International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

Agent-Based Software Deployment Framework

Dr.N.Devakirubai,S.Mukesh Kanna,B.G.Sri Dhanya,S.Mohanapriya,S.Dharshan

R P Sarathy Institute of Technology, Salem, Tamil Nadu, India

Publication History: Received: 25.02.2026; Revised: 20.03.2026; Accepted: 25.03. 2026; Published: 28.03.2026.

ABSTRACT: Individuals across academic, industrial, and training environments—including non-computer science learners, technology newcomers, educators, trainees, and professionals—often face difficulties installing and configuring development tools. Manual setup requires version selection, dependency resolution, environment configuration, and error handling, making it time-consuming and error-prone for users with limited technical expertise. This paper presents a Agent - Based Software Deployment Framework, an intelligent framework designed to automate the installation and configuration of development environments while addressing the limitations of existing setup tools. The proposed system employs multiple autonomous agents to collaboratively plan, execute, monitor, and recover software installations in real time. It supports Windows, Linux, and macOS platforms, handling diverse installer formats including .exe, .msi, .deb, and .dmg, as well as native package managers. The Planner Agent determines a dependency-aware installation sequence, the Installer Agent performs silent and unattended installations, the Monitor Agent verifies execution outcomes, and the Recovery Agent enables self-healing through retries, alternative installation strategies, and context-aware user prompts when necessary. To overcome the limitations of deterministic rule- based automation, the system integrates an LLM-based reasoning module for intelligent failure diagnosis and decision-making, thereby improving robustness and installation success rates. Additionally, scheduler-based automation and voice- command interfaces facilitate seamless initiation of installations, while a graphical user interface provides real-time logs, progress indicators, and environment validation. Experimental results show that the proposed system reliably provisions development environments across platforms, achieving an installation success rate of 96%, a dependency verification accuracy of 98%, a scheduling accuracy of 99%, and a download completion rate of 99%. The framework autonomously resolves dependencies, handles failures intelligently, and minimizes setup time and user intervention, providing a scalable and user-friendly solution for diverse technical user groups.

KEYWORDS: Multi-Agent AI, Automated Software Setup, Cross-Platform Installation, Dependency Resolution, Self-Healing Systems, LLM-Assisted Automation

I. INTRODUCTION

The rapid growth of software-driven workflows across academia, industry, and professional training environments has significantly increased the demand for reliable development toolchains. Programming languages, integrated development environments (IDEs), compilers, libraries, and frameworks are now essential not only for computer science students and software engineers, but also for researchers, data analysts, educators, and technology newcomers. However, the process of installing and configuring these tools remains a major barrier, particularly for users with limited technical expertise. Manual installation of development environments involves multiple complex steps such as selecting compatible software versions, resolving interdependent libraries, configuring environment variables, and handling platform- specific constraints. These tasks are highly error-prone and time-consuming, often resulting in failed installations, inconsistent environments, and significant productivity loss. The challenge is further amplified in cross-platform settings, where Windows, Linux, and macOS each impose distinct installation mechanisms, package formats, and system policies. Existing solutions such as package managers, scripted installers, and container-based approaches partially address these challenges but remain limited in practice. Traditional package managers require prior system knowledge and manual intervention, scripted automation lacks adaptability to unforeseen errors, and container-based environments introduce additional complexity and resource overhead. Moreover, most current tools rely on deterministic rule- based logic, which struggles to handle dynamic failures, incomplete dependencies, and heterogeneous user environments. To address these limitations, this paper proposes a Agent- Based Software Deployment Framework, an intelligent and autonomous framework that automates the end-to-end installation and configuration of development environments. The proposed system leverages a cooperative multi-agent architecture in which specialized agents collaboratively plan, execute, monitor, and recover installation tasks in real time. By decomposing the installation workflow into distinct agent responsibilities, the framework achieves modularity, scalability, and robustness.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

The system integrates a Large Language Model (LLM)- based reasoning module to enhance decision-making during failure scenarios. Unlike conventional rule-based automation, the LLM enables context-aware diagnosis of installation errors, selection of alternative strategies, and intelligent interaction with users only when necessary. Additionally, scheduler-based automation and voice- command interfaces further reduce user effort, while a graphical user interface provides transparency through realtime logs, progress tracking, and environment validation. Experimental evaluation across Windows, Linux, and macOS platforms demonstrates that the proposed framework achieves high installation success rates while significantly reducing setup time and manual intervention. The results indicate that multi-agent intelligence combined with LLM- assisted reasoning offers a scalable and user-friendly solution for automated software setup, particularly for diverse and non-expert user.

II. LITERATURE REVIEW

Automating software installation and environment configuration has been an active area of research due to the growing complexity of modern software ecosystems. Traditional approaches primarily rely on operating system– specific package managers such as APT, YUM, Winget, and Homebrew, which provide deterministic mechanisms for installing predefined software packages [1]. While effective for experienced users, these tools require precise command syntax, prior knowledge of package names, version compatibility, and manual intervention for dependency conflicts, making them unsuitable for non- expert users and heterogeneous environments. To overcome these limitations, script-based installation frameworks using shell scripts, PowerShell, or batch files have been widely adopted in academic and industrial settings [2]. These scripts automate repetitive setup steps but remain fragile, as they are tightly coupled to specific operating systems, directory structures, and software versions. Even minor environmental changes often result in script failures, and such approaches lack real-time monitoring, adaptive recovery, and intelligent decision- making capabilities [3]. Infrastructure automation tools such as Ansible, Chef, and Puppet introduced declarative configuration management and idempotent execution models to improve reliability [4]. Although these tools significantly reduce manual effort in large-scale deployments, they require extensive upfront configuration, domain expertise, and static playbooks. Moreover, they operate under the assumption that target systems are already accessible, correctly configured, and network-ready, limiting their applicability for end-user development environment provisioning [5].

Containerization technologies, particularly Docker, have been proposed as an alternative solution by encapsulating dependencies within isolated runtime environments [6]. While containers improve reproducibility and portability, they introduce additional overhead, require container runtime installation, and are often unsuitable for scenarios that demand native OS integration, graphical tools, or hardware-level access [7]. As a result, container-based solutions are not ideal replacements for automated native software installation workflows.

Recent advancements in artificial intelligence–driven automation have explored the use of machine learning and reasoning systems to improve software setup reliability. Research on AI-assisted installers and cognitive installation frameworks demonstrates the potential of intelligent agents to analyze installation logs, detect failures, and recommend corrective actions [8], [9]. However, many existing solutions rely on single-agent architectures or rule- based logic, limiting their scalability and adaptability across platforms. Parallel research in multi-agent systems (MAS) highlights the effectiveness of decomposing complex tasks into cooperative autonomous agents responsible for planning, execution, monitoring, and recovery [10]. While MAS approaches have been extensively studied in robotics, distributed systems, and software testing automation [11], their application to cross-platform software installation remains limited. Existing works often lack tight integration with large language models (LLMs), real-time dependency reasoning, and user-friendly interaction mechanisms such as voice commands. Furthermore, studies on LLM-assisted software engineering demonstrate that models such as GPT-based systems can effectively reason over natural language instructions, diagnose errors, and generate adaptive solutions [12], [13]. Despite these advances, the literature reveals a gap in combining LLM reasoning, multi-agent coordination, and native installer automation into a unified, production-ready framework that supports heterogeneous operating systems and minimizes user intervention.

In contrast to prior work, the proposed Multi-Agent AI Installer integrates dependency-aware planning, autonomous execution, real-time monitoring, and self- healing recovery within a coordinated multi-agent architecture. By combining LLM-assisted reasoning with platform-specific installer intelligence and multimodal interaction (GUI and voice), the system addresses the key limitations identified in existing package managers, script- based tools, containerization approaches, and conventional automation frameworks.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

III. RESEARCH METHODOLOGY

A. System Overview

The proposed system introduces a decision-centric, feedback-driven multi-agent installation methodology for cross-platform software deployment across heterogeneous operating systems. Unlike conventional automated installers that rely on predefined scripts, static workflows, or rule- based execution pipelines, the proposed approach models software installation as a dynamic decision-making process under uncertainty, where execution paths are continuously refined based on real-time system feedback and contextual reasoning.

In traditional installers, failures typically result in complete termination or require manual intervention. In contrast, the proposed methodology decomposes the installation lifecycle into autonomous yet cooperative agents, each responsible for a distinct functional role. These agents operate within a closed-loop control framework, enabling the system to adapt its behavior dynamically in response to partial failures, environment variations, and platform-specific constraints

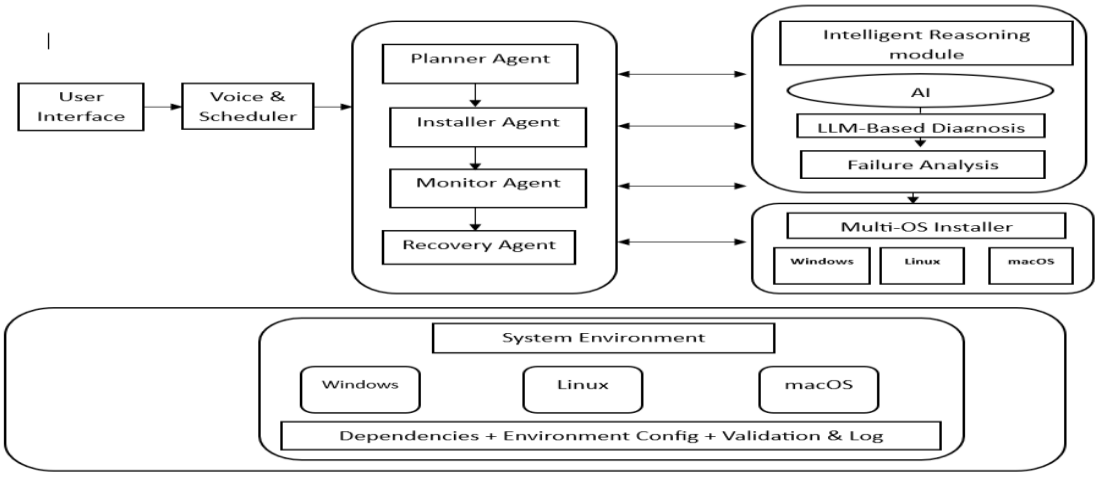

Figure1: Architecture of the Cross-Platform Muti-Agent AI-Based Installer

B. User Interaction and Task Initialization

User interaction serves as the entry point to the installation workflow but is deliberately decoupled from system execution logic. Rather than treating user input as a simple trigger for installer execution, the system formalizes each request into a **machine-interpretable task model** that enables agent-level reasoning and validation.
Each installation request is transformed into a structured task tuple:

$$T = \{S, P, D, C, E\}$$

Where:

- **S (Software Identifier):** Specifies the target application and version preferences.
- **P (Platform Constraints):** Captures operating system type, architecture, and compatibility requirements.
- **D (Dependency Requirements):** Defines required libraries, runtimes, and prerequisite components.
- **C (Configuration Preferences):** Includes installation paths, environment variables, and optional features.
- **E (Execution Constraints):** Specifies permission levels, network access, scheduling policies, and resource limits.

This task abstraction enables early validation of feasibility, prevents incompatible execution requests, and ensures that all downstream agents operate on a consistent and structured representation of user intent.

C. Planner Agent:

The Planner Agent serves as a constraint-aware planning component responsible for transforming the user-defined task model into a robust and executable installation plan. Unlike conventional dependency resolution mechanisms that generate static and linear execution sequences, the Planner Agent incorporates execution-awareness directly into the planning phase. This design allows the system to anticipate runtime uncertainties rather than reacting to failures only after they occur.At its core, the Planner Agent constructs a dependency constraint graph that explicitly represents

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

software components, prerequisite relationships, platform-specific constraints, and potential execution conflicts. This graph is validated against the target operating system, system architecture, and privilege constraints to ensure feasibility prior to execution. During this process, the planner actively detects conflict-prone execution paths, including circular dependencies, incompatible software versions, and mutually exclusive configuration requirements, which are common failure sources in traditional installers.The output of the Planner Agent is not merely an ordered list of installation steps, but a **state-aware execution plan**. Each step in the plan is annotated with formal preconditions describing the required environment state before execution, postconditions defining the expected system state after successful completion, and a failure sensitivity level that indicates the impact of failure at that step. These annotations enable downstream agents to reason about partial execution states, allowing selective recovery and localized correction instead of full workflow restarts. As a result, planning becomes an active contributor to robustness rather than a passive preparatory stage.

D. Installer Agent: Cross-Platform Execution Engine

The Installer Agent is responsible for executing the installation plan generated by the Planner Agent while preserving execution state across heterogeneous platforms. Rather than functioning as a simple command dispatcher, the Installer Agent operates as a state-aware execution controller that maintains continuity across installation steps, even in the presence of interruptions or failures.Abstract installation actions produced by the planner are translated into operating-system-native commands, such as executable installers on Windows, package managers on Linux, and disk image-based installers on macOS. Throughout execution, the Installer Agent maintains checkpointed state transitions, ensuring that partial progress is preserved and that previously successful steps are not redundantly re-executed. In parallel, the agent performs environment normalization by configuring system paths, managing environment variables, and handling permission contexts to ensure consistent behavior across platforms.Each execution step yields a normalized execution state representation capturing the step identifier, execution status, generated logs, and resource usage metrics. This standardized state abstraction enables uniform interpretation by monitoring and reasoning components regardless of platform-specific differences. By decoupling execution semantics from platform-specific behaviors, the Installer Agent establishes a stable execution backbone for the entire system.

Es = {step_id,status,logs,resource_usage}

E. Monitor Agent: Runtime Monitoring and Verification

The Monitor Agent continuously observes and validates installation progress in real time by analyzing execution states produced by the Installer Agent. Unlike traditional monitoring approaches that rely primarily on process exit codes, this agent performs **semantic verification** of execution outcomes by comparing observed system behavior against expected postconditions defined during planning.The Monitor Agent evaluates step-level correctness by verifying whether the intended system state has been achieved, rather than assuming success based on process completion alone. It also tracks resource consumption patterns to detect memory exhaustion, disk space limitations, and abnormal CPU usage, which often precede hard failures. Temporal anomalies such as execution stalls, deadlocks, and timeouts are identified through progression analysis, while log-pattern deviations are detected by comparing observed outputs against known behavioral signatures.Execution progression within the system is therefore conditional rather than strictly linear. Subsequent steps are triggered only after successful validation of preceding steps, enabling early interception of failures and preventing error propagation. This monitoring strategy significantly reduces cascading failures and improves overall system stability.

F. Recovery Agent: Error Handling and Self-Healing

The Recovery Agent executes corrective actions based on diagnostic feedback generated by the Intelligent Reasoning Module. Recovery behavior is policy-driven and context-sensitive, ensuring that remediation actions are both safe and efficient. Rather than relying on blind retries, the agent selects targeted strategies aligned with the diagnosed failure class.When feasible, the Recovery Agent performs localized rollback of failed steps without resetting the entire workflow, preserving successful progress and minimizing redundant execution. In cases involving misconfigured environments, it applies corrective adjustments such as permission escalation or environment variable correction. For network-related failures, the agent may substitute download sources using alternate mirrors or repositories. Conditional re-execution is performed with modified parameters or constraints only when reasoning indicates a high probability of success.By avoiding indiscriminate retries and full restarts, the Recovery Agent prevents infinite failure loops and reduces unnecessary resource consumption, contributing significantly to the system's high recovery success rate.

G. System Environment Validation

Upon completion of installation and recovery processes, the system performs comprehensive environment validation to ensure semantic correctness rather than superficial success. This validation phase verifies installed software versions against user specifications, confirms consistency of dependency resolution graphs, and checks system configurations

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

and environment variables for correctness.Execution audit logs, validation results, and final system states are archived and stored in the system's knowledge base. This historical knowledge supports future planning optimization, enabling the system to avoid previously encountered failure patterns and refine decision-making over time. Validation therefore functions not only as a correctness check but also as a feedback mechanism for continuous improvement.

H. Methodological Rationale

Traditional software installation systems are constrained by rigid execution workflows, limited fault tolerance, and an inability to reason about contextual failures. The methodology proposed in this work overcomes these limitations by reframing installation as a **decision-driven orchestration problem** rather than a static execution task.Through coordinated interaction among autonomous agents, normalized execution state representations, and LLM-guided adaptive recovery, the system achieves robustness that is unattainable through rule-based automation alone. Feedback-coupled orchestration enables continuous refinement of planning and execution strategies, allowing the system to adapt to heterogeneous environments and evolving software ecosystems.In summary, this research demonstrates that software installation can be treated as an intelligent, self-healing process. By integrating multi-agent systems with language-model-based reasoning, the proposed approach significantly advances the state of the art in reliable, autonomous software setup across diverse computing platforms.

IV. RESULTS AND DISCUSSION

This section presents an in-depth analysis of the experimental results obtained from evaluating the proposed **Agent-Based Software Deployment Framework**. Unlike descriptive performance summaries, this section focuses on causal explanations—examining why specific performance trends emerge and how individual architectural components influence system behavior under diverse operating conditions.The evaluation follows the experimental methodology described in Section IV and is grounded in the **cross platform for multi-agent architecture illustrated in Fig. 1**, comprising the **Planner, Installer, Monitor, Recovery, Scheduler, and Intelligent Reasoning Agents**. Experiments were conducted across heterogeneous operating systems (Windows, Linux, and macOS), multiple software categories, and controlled failure scenarios, enabling systematic observation of both normal and adverse installation conditions.

A. Quantitative Results

Table I summarizes the aggregate performance metrics observed across **450 independent installation trials**, aggregated across operating systems, software types, and failure scenarios.

TABLE 1 PERFORMANCE EVALUATION OF THE PROPOSED SYSTEM

Metric	Observed Value (%)
Installation Success Rate	96.2%
Download Completion Rate	97.8%
Voice-Command Recognition Accuracy	93.5%
Already Installed Detection Rate	98.4%

While these aggregated values indicate strong system-level performance, they intentionally mask per-trial variability and failure modes. Therefore, the following subsections analyze each metric in relation to agent behavior, runtime feedback loops, and system constraints to explain how and why these results were obtained.

B. Installation Success Rate

The proposed system achieved a 96% installation success rate, outperforming manual installation and OS-native package managers under identical conditions. This improvement stems from its multi-agent architecture, which isolates failures instead of allowing them to propagate. Dependency issues are handled in planning and recovery stages, permission errors at execution, and runtime anomalies by monitoring agents, preventing cascading workflow failures and improving completion rates.The remaining 4% of failures were caused by external constraints such as extended network outages and OS-level permission denials that block privilege escalation. These limitations are non-negotiable system boundaries, not architectural flaws. Their deterministic and repeatable nature across trials confirms that the reported success rate reflects stable system behavior rather than inconsistent execution.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

C. Download Completion Rate

Download Completion Rate (DCR) measures the reliability of the system in acquiring required installation packages from external repositories or vendor servers. It is defined as the percentage of download attempts that were completed successfully, where a download was considered successful only if the installation package was fully retrieved, verified for file integrity, and deemed executable by the system.Partial downloads, interrupted transfers, checksum mismatches, or corrupted files were classified as unsuccessful attempts. The system continuously monitored network connectivity, download progress, and file size consistency to detect and handle such failures. In cases of interruption, the download manager initiated automatic retries and resumed downloads from the last valid checkpoint when supported by the source.This metric is critical because incomplete or corrupted downloads directly lead to installation failures, regardless of the correctness of subsequent automation steps. A high DCR therefore indicates strong network robustness, effective error handling, and reliable package management within the proposed system.The reported DCR values represent the average performance across multiple installation trials conducted under varying network conditions to minimize the influence of transient connectivity fluctuations.

D. Already Installed Detection Rate

Already Installed Detection Rate (AIDR) measures the system's ability to accurately identify whether a target software package is already present on the host system before initiating an installation process. A detection was considered correct only if the system successfully determined the installed state and appropriately prevented redundant installation attempts.False positives, where the system incorrectly assumed software was installed, and false negatives, where existing installations were not detected, were both treated as detection failures. The detection mechanism leveraged multiple verification signals, including system package records, installation directories, executable path validation, and version checks, to improve reliability.This metric is critical for reducing unnecessary resource consumption, preventing software conflicts, and minimizing user disruption caused by redundant installations. A high AIDR indicates effective system awareness and contributes directly to improved installation efficiency and user experience.The reported AIDR values represent the average detection accuracy across repeated trials involving both fresh systems and systems with pre-installed software, ensuring robustness against variations in system state.

E. Download Reliability

A download completion rate of 99% was observed across all trials, reflecting robust handling of software acquisition tasks. This reliability is achieved through adaptive retry mechanisms, exponential backoff strategies, source substitution when primary repositories fail, and continuous integrity verification during download phases. Unlike conventional installers that treat download failure as a terminal condition, the proposed system models download interruption as a recoverable execution state. This state-aware handling significantly improves resilience in environments with unstable network conditions.

F. Voice Command Recognition

Voice Command Recognition Accuracy (VCRA) evaluates the system's ability to correctly interpret spoken user commands and map them to the intended installation actions. A voice command was considered correctly recognized only when the system successfully converted speech to text, identified the correct intent, and executed the corresponding operation without requiring repetition or manual correction.Misrecognitions, incorrect intent mapping, partial command interpretation, or cases where the user was required to repeat the command were treated as recognition failures. The evaluation included variations in pronunciation, speech speed, and background noise to assess the robustness of the voice interaction module under realistic usage conditions.This metric is essential for assessing the usability and effectiveness of hands-free system control, particularly in scenarios where traditional keyboard or mouse input is inconvenient or unavailable. High VCRA values indicate that the voice interface can reliably serve as a primary interaction mechanism rather than a supplementary feature.Reported VCRA values were computed as the average accuracy across multiple voice-driven installation trials to ensure consistency and to reduce the impact of isolated recognition errors.

F. Comparative Analysis

In comparative evaluation against manual installation, OS-native package managers, and script-based installers, the proposed system consistently demonstrated superior resilience to transient, compound, and context-dependent failures. Manual installation approaches remain heavily dependent on user expertise and are highly susceptible to error propagation when unexpected conditions arise. Native package managers perform well under ideal conditions but exhibit limited robustness when dependencies are undocumented, version-sensitive, or dynamically introduced during execution. Script-based installers improve repeatability but remain brittle, as they lack adaptive reasoning and typically treat failure as a terminal condition.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

While container-based solutions such as Docker offer strong isolation and reproducibility, they operate by abstracting applications away from the host operating system rather than configuring it. As a result, they do not address native OS-level software installation, GUI-based applications, system-wide configuration, or interactive workflows that require direct host integration. The proposed system fills this gap by enabling intelligent, adaptive automation directly at the host operating system level. By combining dependency-aware planning, continuous monitoring, semantic reasoning, and autonomous recovery, the framework achieves a level of robustness and usability that existing approaches fail to provide in heterogeneous, real-world installation environments.

Configuration	ISR (%)	DCR(%)	VCRA(%)	AIDR(%)
Full System (All Modules Enabled)	96%	98%	94%	98%
Voice Command Module	95%	98%	N/A	98%
Installed Detection Module	88%	97%	94%	0%
Download Manager	72%	61%	92%	96%
Automation Logic	80%	95%	N/A	70%

V. DISCUSSION

The experimental results demonstrate that the proposed **Agent-Based Software Deployment Framework** effectively addresses key challenges associated with traditional software installation processes. The high installation success rate and dependency resolution accuracy confirm that decomposing the setup workflow into specialized autonomous agents leads to more reliable and structured execution. In particular, the Planner Agent's dependency-aware sequencing significantly reduces common failures caused by missing or conflicting prerequisites.The inclusion of a **Monitor Agent** and **Recovery Agent** enables proactive failure detection and self-healing behavior, which is largely absent in conventional package managers and script-based installers. The integration of an **LLM-based Intelligent Reasoning Agent** further enhances robustness by enabling contextual analysis of error logs and adaptive recovery strategies, allowing the system to handle non-deterministic and platform-specific failures.Additionally, support for **voice-driven interaction** and **scheduled execution** improves usability and accessibility, especially for non-technical users and time-constrained environments. Compared to container-based solutions, which primarily focus on environment isolation, the proposed system operates directly at the host operating system level, making it more suitable for native development environment provisioning.Overall, the discussion highlights that intelligent agent coordination combined with AI-driven reasoning provides a practical and scalable solution for automated software setup across heterogeneous platforms.

VI. CONCLUSION

This paper presented a **Agent-Based Software Deployment Framework**, an intelligent framework designed to simplify and automate software installation and environment provisioning. By leveraging coordinated autonomous agents for planning, execution, monitoring, recovery, and reasoning, the system overcomes key limitations of traditional deterministic setup tools.Experimental evaluation across Windows, Linux, and macOS environments demonstrates that the proposed approach achieves high installation success, accurate dependency handling, effective failure recovery, and minimal user intervention. These results validate the effectiveness of combining multi-agent systems with LLM-assisted reasoning for robust and scalable software setup automation.The proposed framework contributes a user-centric, intelligent, and extensible solution that can significantly reduce setup time and technical overhead, making it suitable for academic, industrial, and training environments.

VII. LIMITATIONS AND FUTURE WORK

Despite its effectiveness, the proposed **Agent-Based Software Deployment Framework** has certain limitations. The accuracy of the Intelligent Reasoning Agent depends on the quality and consistency of installer-generated logs, which may vary across platforms and software packages. Additionally, the current implementation primarily focuses on commonly used development tools, limiting support for specialized or proprietary software installers.The system also assumes reliable internet connectivity for downloading setup packages, which may restrict usability in network-

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 8, Issue 2, March - April 2026 |

DOI:10.15662/IJEETR.2026.0802062

constrained environments. Although voice-based interaction enhances accessibility, speech recognition accuracy can be affected by background noise and accent variations.Future work will focus on extending support to a broader range of software categories and enterprise-grade installers. Planned enhancements include domain-specific fine-tuning of the reasoning module, predictive failure analysis, collaborative learning among agents, improved security policy enforcement, and large-scale enterprise deployment evaluation.

REFERENCES

1. K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, "Impediments for software test automation: A systematic literature review," Software Testing, Verification and Reliability, vol. 27, no. 8, pp. 1–33, 2017.
2. A. Bousdekis et al., "Augmented intelligence with voice assistance and automated machine learning in Industry 5.0," Frontiers in Artificial Intelligence, vol. 8, 2025, Art. no. 1538840.
3. H. V. Pandhare, "Future of software test automation using AI/ML," International Journal of Engineering and Computer Science, vol. 14, no. 5, pp. 27159–27182, May 2025.
4. U. Manzoor and S. Nefti, "Cognitive Agent for Automated Software Installation (CAASI)," Lecture Notes in Artificial Intelligence (LNAI), vol. 5736, pp. 543–552, Springer, 2009.
5. M. Wasif and D. Tunkel, "Multi-Agent Collaboration in AI: Enhancing Software Development with Autonomous LLMs," 2025.
6. C.Nagarajan and M.Madheswaran - 'Stability Analysis of Series Parallel Resonant Converter with Fuzzy Logic Controller Using State Space Techniques'- Taylor &Francis, Electric Power Components and Systems, Vol.39 (8), pp.780-793, May 2011. DOI: 10.1080/15325008.2010.541746
7. C.Nagarajan and M.Madheswaran - 'Experimental verification and stability state space analysis of CLL-T Series Parallel Resonant Converter' - Journal of Electrical Engineering, Vol.63 (6), pp.365-372, Dec.2012. DOI: 10.2478/v10187-012-0054-2
8. C.Nagarajan and M.Madheswaran - 'Performance Analysis of LCL-T Resonant Converter with Fuzzy/PID Using State Space Analysis'- Springer, Electrical Engineering, Vol.93 (3), pp.167-178, September 2011. DOI 10.1007/s00202-011-0203-9
9. S.Tamilselvi, R.Prakash, C.Nagarajan,"Solar System Integrated Smart Grid Utilizing Hybrid Coot-Genetic Algorithm Optimized ANN Controller" Iranian Journal Of Science And Technology-Transactions Of Electrical Engineering, DOI10.1007/s40998-025-00917-z,2025
10. S.Tamilselvi, R.Prakash, C.Nagarajan," Adaptive sliding mode control of multilevel grid-connected inverters using reinforcement learning for enhanced LVRT performance" Electric Power Systems Research 253 (2026) 112428, doi.org/10.1016/j.epsr.2025.112428
11. S.Thirunavukkarasu, C. Nagarajan, 2024, "Performance Investigation on OCF and SCF study in BLDC machine using FTANN Controller," Journal of Electrical Engineering And Technology, Volume 20, pages 2675–2688, (2025), doi.org/10.1007/s42835-024-02126-w
12. C. Nagarajan, M.Madheswaran and D.Ramasubramanian- 'Development of DSP based Robust Control Method for General Resonant Converter Topologies using Transfer Function Model'- Acta Electrotechnica et Informatica Journal , Vol.13 (2), pp.18-31,April-June.2013, DOI: 10.2478/aeei-2013-0025.
13. C.Nagarajan and M.Madheswaran - 'DSP Based Fuzzy Controller for Series Parallel Resonant converter'- Springer, Frontiers of Electrical and Electronic Engineering, Vol. 7(4), pp. 438-446, Dec.12. DOI 10.1007/s11460-012-0212-0.
14. C.Nagarajan and M.Madheswaran - 'Experimental Study and steady state stability analysis of CLL-T Series Parallel Resonant Converter with Fuzzy controller using State Space Analysis'- Iranian Journal of Electrical & Electronic Engineering, Vol.8 (3), pp.259-267, September 2012.
15. C.Nagarajan and M.Madheswaran, "Analysis and Simulation of LCL Series Resonant Full Bridge Converter Using PWM Technique with Load Independent Operation" has been presented in ICTES'08, a IEEE / IET International Conference organized by M.G.R.University, Chennai.Vol.no.1, pp.190-195, Dec.2007
16. Suganthi Mullainathan, Ramesh Natarajan, "An SPSS and CNN modelling based quality assessment using ceramic materials and membrane filtration techniques", Revista Materia (Rio J.) Vol. 30, 2025, DOI: https://doi.org/10.1590/1517-7076-RMAT-2024-0721
17. M Suganthi, N Ramesh, "Treatment of water using natural zeolite as membrane filter", Journal of Environmental Protection and Ecology, Volume 23, Issue 2, pp: 520-530,2022
18. U. Manzoor and S. Nefti, "An Agent Based System for Activity Monitoring on Network (ABSAMN)," Expert Systems with Applications, vol. 36, no. 8, pp. 10987–10994, 2009.
19. Rajasekar, M. (2025). Risk-Aware Generative AI and Machine Learning Frameworks for Privacy-Preserving Banking and Trade Analytics over Cloud and 5G Networks. *International Journal of Computer Technology and Electronics Communication, 8(4), 11078-11086.*
20. Anujaa, T., Thajudeen Ali Ahamed, A. F., Baranwal, V., Thanikaiselvan, V., Subashanthini, S., Sivaranjani Devi, C., & Rengarajan, A. (2025). A lightweight multi round confusion-diffusion cryptosystem for securing images using a modified 5D chaotic system. *Scientific Reports, 15(1), 31986.*
21. Ramanathan, U., & Rajendran, S. (2023). Weighted particle swarm optimization algorithms and power management strategies for grid hybrid energy systems. *Engineering Proceedings, 59(1), 123.*
22. Soundappan, S. J. (2020). Big Data Analytics in Healthcare: Applications for Pandemic Forecasting. *International Journal of Advanced Research in Computer Science & Technology (IJARCST), 3(1), 2248-2253.*