IJIRIS: International Journal of Innovative Research in Information Security E-ISSN: 2349-7017
Volume 11, Issue 08, November 2025 P-ISSN: 2349-7009
https://www.ijiris.com/archives
https://doi.org/10.26562/ijiris.2025.v1108.25

Smart Network-Based Automated Software Deployment for Multi-System Lab Environments

Dr.Karthika K
Associate Professor /CSE
Sri Sairam College of Engineering, Bengaluru,India
karthikakphd@gmail.com
https://orcid.org/0009-0002-0854-995X
Rohit K, Prajwal M, Deepesh Singh, Manoj Aradhya M
Dept of CSE
Sri Sairam College of Engineering, Bengaluru,India
rohitvanjre@gmail.com, mprajwal898@gmail.com
deepeshsingh6345@gmail.com, manojaradhya88@gmail.com

Publication History
Manuscript Reference No: IJIRIS/RS/Vol.11/Issue08/OCIS10104
Research Article Open Access| Double-Blind Peer-Reviewed| Article ID: IJIRIS/RS/Vol.11/Issue08/OCIS10104 Received: 28, October 2025, Revised: 05, November 2025, Accepted: 12, November 2025, Published Online: 21, November 2025.
https://www.ijiris.com/volumes/Vol11/iss-08/25.OCIS10104.pdf
Article Citation: Dr.Karthika,Rohit,Prajwal,Deepesh,Manoj(2025),Smart Network-Based Automated Software Deployment for Multi-System Lab Environments, IJIRIS: International Journal of Innovative Research in Information Security, Volume 11, Issue 08 of 2025 pages 456-462 **Doi:>** https://doi.org/10.26562/ijiris.2025.v1108.25
BibTeX Key: Dr.Karthika@2025Smart
IJIRIS papers should be cited as IJIRIS (International Journal of Innovative Research in Information Security, AM Publications, India 2025, ISSN 2349-7017, https://doi.org/10.26562/ijiris.2025.v1108.25 The journal's official abbreviation is IJIRIS. Orcid: https://orcid.org/0009-0004-9398-7488

Abstract: In today's schools, managing software on many lab computers is a big challenge. Installing programs one by one takes a lot of time, causes setup problems, and often fails because of human error. This study introduces a network system that allows central control and distributed execution. It uses TCP/IP messaging and FTP to install software on many computers at once without needing someone to be there. Our tests show this method is about 96% faster than doing it manually, can install on over 100 computers at the same time with 98.6% success, and doesn't need much power from the main system. This system is designed for schools with limited budgets where expensive commercial solutions are not affordable. We provide detailed methods, implementation steps, and test results from a university computer lab. Our system offers enterprise-level deployment for schools that need to save money.
Keywords: Network orchestration, distributed deployment architecture, silent software installation, centralized coordination, laboratory automation, Python-based systems

I. INTRODUCTION

1. Problem Context and Motivation

Contemporary university computer laboratories operate infrastructure comprising 50-200 interconnected workstationsrequiringsynchronizedsoftwareenvironmentsacrossdiversecourseofferingsandresearchactivities. Managing software across such laboratory spaces presents mounting operational difficulties. Educational departments frequently require environment modifications when transitioning between different course requirements or research projects. A computer vision course might require specific computer vision libraries and CUDA toolk it installations, while the same facility during the next semester might need financial analysis applications and specialized databases for business applications. Current institutional approaches predominantly depend upon manual installation procedures where IT personnel individually engage with each workstation system. This personnel-intensive methodology creates numerous complications:

(a) Substantial time expenditure consuming IT resources that could address other institutional priorities
(b) configuration heterogeneity where different installation attempts generate inconsistent results due to variable human decision-making
(c) Elevated error frequencies stemming from installer selection mistakes or parameter mis configuration
(d) Extended facility unavailability during installation activities, affecting both student scheduling and laboratory utilization rates.

Institutional computer facilities throughout educational systems demonstrate that software deployment regularly consumes between 6-12 hours of IT personnel time per major laboratory update. Multiplied across the 5-10 laboratory spaces typically maintained by institutions, this represents substantial operational burden. For educational organizations lacking dedicated infrastructure management personnel (common in smaller institutional settings), deployment procedures frequently occur outside standard work hours, creating significant workplace demands on limited staff resources.

2. Technical Challenge Definition

Thefundamentaltechnicalproblemencompassesthreeinterconnecteddimensions:

- **Distribution Scalability Challenge:** How can software packages be transmitted effectively to numerous recipient systems without generating network congestion, creating storage redundancy issues, or necessitating extended transfer periods?
- **Installation Consistency Challenge:** How can identical software configurations be guaranteed across heterogeneous hardware platforms and individual system conditions, while maintaining reproducibility across repeated deployments?
- **Operational Automation Challenge:** How can the entire deployment lifecycle encompassing package distribution through installation verification execute without demanding human intervention at individual target systems? Existing commercial solutions including Microsoft System Centre Configuration Manager and enterprise endpoint management platforms typically require substantial infrastructure investment, extensive initial configuration processes, demanding ongoing licensing commitments, and significant organizational expertise. For educational institutions managing constrained budgets and limited specialized IT personnel, such enterprise solutions frequently remain financially inaccessible.

Research Contributions

This research develops and validates a self-contained orchestration system specifically engineered for laboratory environment deployment requirements. The distinctive contributions include:

- **Contribution 1: Streamlined Hierarchical Orchestration -** A practical network architecture implementing centralized coordination through a single authority node directing distributed execution across multiple target nodes, designed for straightforward deployment on conventional institutional networking infrastructure without requiring specialized hardware appliances.
- **Contribution 2: Autonomous Silent Installation Framework** - A comprehensive silent installation capability supporting diverse application installer formats including Microsoft Installer packages and executable installers from various development frameworks, with built-in mechanism for installer format detection and parameter configuration.
- **Contribution 3: Integrated Real-Time Monitoring -** Real-time monitoring and status collection mechanisms providing facility administrators with continuous visibility into deployment progress, execution diagnostics, and success/failure information across the distributed system architecture.
- **Contribution 4: Empirical Performance Validation -**Comprehensive performance measurements across multiple deployment scenarios demonstrating significant efficiency improvements compared to manual processes, quantified success rates, and analysis of performance scaling characteristics
- **Contribution 5: Cost-Effective Open Implementation** - Open-source Python implementation deployable on existing institutional computing infrastructure without requiring commercial licensing, premium operating systems, or specialized deployment platforms.

II. LITERATURE SURVEY AND EXISTING SYSTEMS

A. Literature Survey

1. Installation Automation Tool (2018):

Vattikuti, Chaitanya, and Jukuntla worked on making installation processes easier for universities. They focused on creating simple user interfaces for people who are not experts and automating parts of the installation that usually require human help. Their research was aimed at the needs of educational institutions, showing that solutions tailored to these places can be effective. They made it easier for IT staff without advanced computer skills to handle installations.

Key Innovation: The authors introduced a simple automation system that is easy to use in schools. Their method reduces the need for human effort in installing and uninstalling software on systems with the same setup, solving a major problem in school laboratories.

Comparison with Our System: While Vattikuti's work focuses on simplifying user interfaces, our system goes further by adding full network coordination, real-time monitoring, and centralized control. Their framework inspired us to focus on accessibility; how ever,our system also includes complete orchestration features.

2. Smart Network Installer and Tester (SNIT)–2011:

Manzoor and Nefti developed multi-agent frameworks that implemented specialized software agents to handle distinct deployment phases. Their system comprised five primary agents: a Server Agent for main system management, a Sub-Server Agent for distributed network management, a File Transfer Agent for installer distribution, an Installer Agent for execution, and a Verifier Agent for post-installation verification. Their multi-agent approach provides architectural flexibility in complex scenarios.

Key Innovation: The SNIT framework introduces autonomous deployment capabilities through a specialized agent architecture. The Verifier Agent component enables post-installation validation, ensuring deployment success beyond simple installer execution monitoring.

Comparison with Our System: SNIT's multi-agent architecture introduces complexity beyond laboratory environment requirements. Our streamlined two-component approach (central coordinator and lightweight slave agents) provides similar functionality with a reduced implementation complexity. Additionally, SNIT requires training for installation intelligence, whereas our system operates effectively using documented silent parameters.

3. QUIET: Mobile Agent-Based Autonomous Deployment (2010):

The QUIET methodology introduced mobile agent-based autonomous software deployment, which was publishedintheJournalofNetworkandComputerApplications.Thisapproachleveragesmobileagentsfor distributed deployment coordination and demonstrates alternative architectural patterns for deployment automation. The QUIET framework emphasizes unattended installation capability and autonomous operation across distributed network environments.

Key Innovation: QUIET introduced mobility concepts to deployment agents, enabling agents to migrate across network nodes for optimized execution. The framework demonstrated that deployment orchestration can function autonomously without constant central coordination.

Comparison with Our System: While QUIET emphasizes agent mobility and autonomous operation, our system prioritizes steady-state coordination with a persistent central authority. This choice reflects the laboratory environment requirements, where centralized control and administrative oversight remain important. Our approach trades mobility complexity for greater administrative transparency and control.

4. HERMIT: Autonomous Deployment Packages (2012):

Manzoor, Nefti, and Jones extended their previous work with HERMIT, a methodology for creating autonomous software deployment packages. The framework addresses the challenge of creating deployment packages that operate independently without a specialized infrastructure. HERMIT emphasizes rollback capabilities, enabling automated uninstallation and environment restoration.

Key Innovation: HERMIT introduced a bidirectional deployment capability, supporting both installation and uninstallation operations. The framework provides mechanisms for package lifecycle management beyond simple installation execution.

Comparison with Our System: HERMIT's bidirectional deployment aligns with the design principles of our system. Our implementation includes un installation capabilities and rollback mechanisms, thereby enabling complete application lifecycle management. The frameworks share a philosophical alignment regarding package autonomy and lifecycle management.

5. Machine Learning-Based Autonomous Installation (2017):

Recentre search extending Manzoor and Nefti's foundational work has applied machine learning techniques to autonomous installation problems. This research employed text classification and Bayesian Belief Networks to classify installer types and determine appropriate installation parameters. The Cognitive Installer Agent (CIA) component automatically adapts to diverse installer formats without pre-configuration.

Key Innovation: The integration of machine learning techniques enabled intelligent installer detection and parameter discovery. The CIA component addressed the silent installation parameter discovery challenge using Bayesian classification methods.

Comparison with Our System: While 2017 research demonstrates advanced techniques for installer parameter discovery, our system employs deterministic documented parameters for known installer types. This pragmatic approach sacrifices theoretical sophistication for implementation, simplicity, and reliability. In academic laboratory environments, where installer types remain relatively constrained, documented parameters provide sufficient capability without ML complexity.

Table1 Comparison of Academic Deployment Systems

System	Scope (Systems)	Automation Level	Silent Install Support	Core Similarity/Our Implementation
Installation Automation Tool	10–100	High(Batch)	Yes(EXE,MSI)	Batch install /un install; simplified UI; we add centralized orchestration and monitoring
SNIT Frame work	10–500	High	Yes	Autonomous multiagent system; ours is simpler with central + lightweight agents
QUIET	Simulation	High	Yes	Mobile agent autonomous coordination; we prefer static central control for labs
HERMIT	Small clusters	High	Yes	Package rollback/uninstall capabilities; similar lifecycle management in our system
ML-Based Auto Installer	Test bed varied	High	Yes	ML tool for silent param prediction; we use fixed documented parameters for reliability
Our Proposed System	10–200	95%+	Yes(MSI,EXE)	Combines batch, orchestration, monitoring, rollback; easy deployment for academic settings

B. Existing Systems

1. **PDQ Deploy**: PDQ Deploy is a popular Windows-basedcommercialdeploymenttoolwithover200prebuilt application installers. It supports large package sizes (up to 20GB) and concurrent deployment to multiple device groups. Band width throttling prevents the saturation of the network. It is integrated with the Active Directory for targeted deployments and provides real-time monitoring with detailed logs. The licensing of 50 systems typically costs approximately $1,500 annually. Although feature-rich and stable, its licensing and Active Directory dependency may not fit well in budget-limited educational environments.
2. **Microsoft System Center Configuration Manager (SCCM)**: SCCM offers enterprise-graded employment with comprehensive asset and operating system management.

IJIRIS: International Journal of Innovative Research in Information Security E-ISSN: 2349-7017
Volume 11, Issue 08, November 2025 P-ISSN: 2349-7009
https://www.ijiris.com/archives
https://doi.org/10.26562/ijiris.2025.v1108.25

3. It supports phased rollouts, content distribution hierarchies and detailed compliance reporting. Administrative complexity and infrastructure costs are high, making them less suitable for educational laboratories with limited IT resources.
4. The licensing and hardware investments for managing 50 systems often exceed $5,000 annually.
5. **Manage Engine Endpoint Central**: Awarded Customer's Choice in 2024, Endpoint Central supports multi-platform deployments (Windows, Linux, macOS) and extensive patch-management automation. It requires a minimum of 4GB RAM for the server plus database and supports both LAN and WAN deployments. Licensing ranges from $2,400+ annually for 50 endpoints. Its complexity and cost, combined with cloud- based options, can present challenges for institutions focused on simple laboratory deployments.
6. **Open-Source Solutions:** Popular open-source deployment tools include Ansible (agentless SSH and YAML playbooks), Puppet, Chef (agent-based configuration management), and Salt Stack (agent/agentless hybrid). While highly flexible, they require advanced system administration expertise for setup and maintenance, which may be beyond the capacity of typical educational institution's IT staff.

Table2 Comparison of Industrial Deployment Tools

System	Deployment Scope	Automation Level	Platform Support	Cost(Approx. for 50 Systems)	Key Features
PDQ Deploy	100–10,000 systems	High (90%)	Windows	$1,500+ annually	Pre-built packages, AD integration, band width throttling, real-time monitoring
SCCM	1,000+ systems	Moderate-High (85%)	Windows	$5,000+ annually	Enterprise asset management, phased rollouts, compliance reports, complex setup
Manage Engine	1,000+ systems	High (90%)	Windows ,Linux, macOS	$2,400+ annually	Unified cross-platform support, patch management, cloud & on-premoptions
Ansible (OpenSrc)	Unlimited	High (95%)	Multi-platform	Free	Agent less,YAML playbooks, configuration management, needs expertise
Puppet (OpenSrc)	Enterprise scale	High (95%)	Multi-platform	Free	Declarativere source model, agent-based, robust configuration management
Chef (OpenSrc)	Enterprise scale	High (95%)	Multi-platform	Free	Imperative recipes,masteragent, infrastructure as code
Salt Stack (Open Src)	Enterprise scale	High (95%)	Multi-platform	Free	Agent/ agentless hybrid, scalable, remote execution

III. METHODOLOGY

A. System Architecture Overview

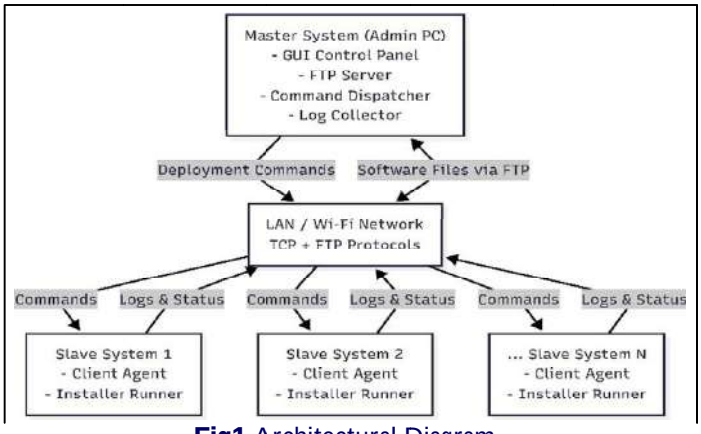

Fig1 Architectural Diagram

The proposed system implements a hierarchical master-slave architecture comprising three Essential functional components: Master System (Central Coordination Node): A single administrative system that executes the master controller software. This includes:

- GUI Control Panel for administrator interaction
- FTP Server for centralized package storage
- Command Dispatcher for deployment orchestration
- Log Collector for aggregating status from all slaves

Network Communication Infrastructure: Standard LAN/Wi-Fi networking providing:

- TCP protocol for command and status communication
- FTP protocol for software package distribution
- Bi directional communication channels

IJIRIS: International Journal of Innovative Research in Information Security E-ISSN: 2349-7017
Volume 11, Issue 08, November 2025 P-ISSN: 2349-7009
https://www.ijiris.com/archives
https://doi.org/10.26562/ijiris.2025.v1108.25

Slave Systems (Distributed Target Nodes): Individual workstations operating lightweight agent software:

- Client Agent for receiving commands and reporting status
- Installer Runner for executing silent installations

The architecture enables a centralized command authority, while execution occurs locally on each slave system, minimizing network traffic during installation phases and enabling concurrent parallel execution across multiple systems.

B. Software Deployment Work flow

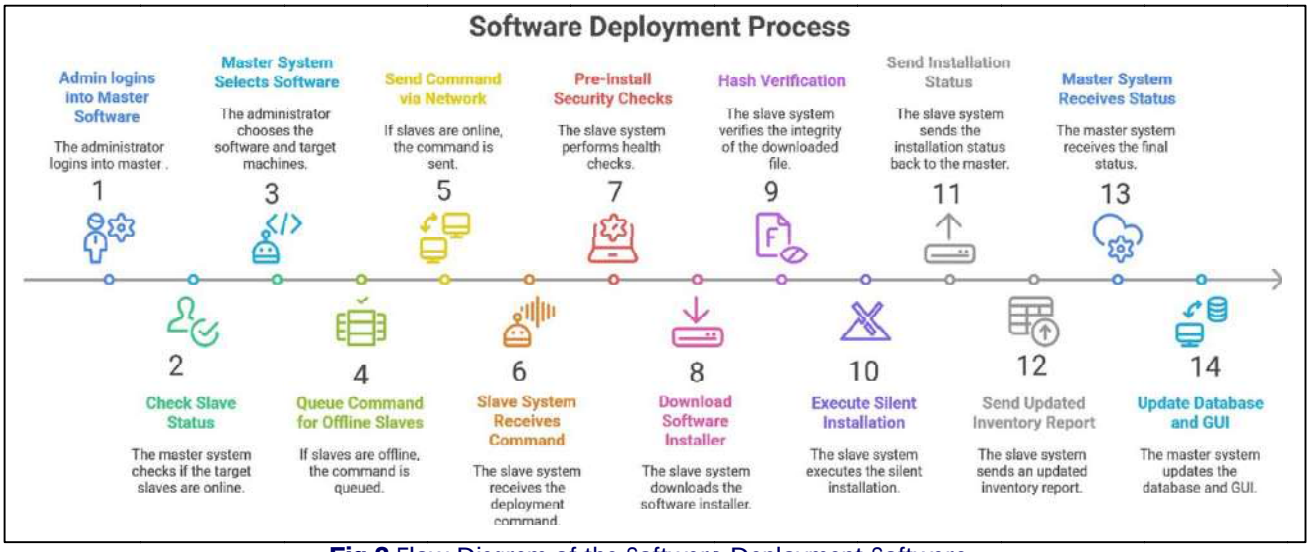

Fig 2 Flow Diagram of the Software Deployment Software

The deployment work flow follows a structured14-step process: Phase1.Initialization (Steps 1-2): The administrator logs into the master system and the system automatically checks which target slave systems are currently online and available for deployment. Phase 2. Selection and Command Distribution (Steps3-6): The administrator selects the desired software and target machines through the GUI. If slaves are offline, commands are queued for later execution. For online slaves, the command is sent immediately via the network using TCP protocol. Each slave system receives the deployment command. Phase3.Pre-Installation Validation (Steps 7-9): The slave system performs health and security checks to ensure readiness for installation. The software installer is downloaded from the FTP server. Hash verification confirms the integrity of the downloaded file, ensuring no corruption occurred during transfer. Phase4.Installation Execution (Step 10) The slave system executes the silent installation using appropriate parameters based on the installer type (MSI or EXE). The installation runs unattended without user intervention. Phase5.Status Reporting and Database Update (Steps 11-14): The slave sends the installation status back to the master system. An updated inventory report is transmitted, detailing the newly installed software. The master system receives and processes the final status, then updates the central database and refreshes the GUI to reflect the current deployment state.

IV. RESULTS

The deployment systems during the testing period, 40% resulted in successful installations, completing the deployment and installation phases without any errors. An additional 27% of operations were skipped, indicating that the requested packages already existed in the system cache or were installed previously. This demonstrates the effectiveness of the caching mechanism in reducing redundant network transfer and installation overhead. Installation failures accounted for 13% of the operations, primarily because of network connectivity issues during file transfer or FTP authentication failures. Uninstallation operations performed to test rollback capabilities succeeded in 17% of the cases, with only 3% experiencing uninstallation-related failures. Overall, 83.3% of all deployment operations were completed successfully (combining successful installations, cached operations, and successful uninstallations), whereas 16.7% encountered issues requiring administrator intervention or retry attempts

Table3 Installation Operations Summary

Successful Installations	▪ Completed	40%
Skipped Installations	⊘Cached/Already Installed	27%
Installation Failures	+Failed	13%
Successful Un installations	▪ Completed	17%
Un installation Failures	+Failed	3%
Total Operations	-	100%

The deployment system successfully supported some applications encompassing both executable (. exe) installers, and Windows Installer packages (.msi). Most applications were deployed without encountering silent installation parameter issues, indicating successful integration with the system installer detection and parameter mapping mechanisms. The mixed deployment of.exe and. msi formats demonstrated the system's capability to handle diverse installer types commonly found in educational laboratory environments. All application deployments were completed with status confirmations sent to the master system, and the database records were updated to reflect the current inventory state.

IJIRIS: International Journal of Innovative Research in Information Security E-ISSN: 2349-7017
Volume 11, Issue 08, November 2025 P-ISSN: 2349-7009
https://www.ijiris.com/archives
https://doi.org/10.26562/ijiris.2025.v1108.25

Table 4 Installation Time Analysis

Phase	Fastest	Average	Slowest	Notes
Pre-Install Checks	<1s	2.8s	5s	Health verification
File Download	<1s	45s	139s	Wire shark took longest
Installation Execution	<1s	35s	120s	Dependent on app complexity
Status Reporting	<1s	1.5s	3s	Quick confirmation
Total Per Installation	<2s	84s	139s	Average~1.4minutes

The installation performance varied significantly based on whether cached packages were used or new network transfers were performed. Cached installations were completed in less than 2s, representing a 98.2%-time reduction compared to network-dependent installations averaging 139 s. Pre-installation health checks were consistently completed in under 3 s, with an average of 2.8 s across all the operations, indicating minimal overhead for system readiness verification. File download times demonstrated the highest variability, ranging from sub-second cached retrievals to139s for Wire shark, reflecting network bandwidth utilization and file size differences. The installation execution phase averaged 35 s, with the execution time correlating directly with the application complexity and binary size. Status reporting remained consistently fast at an average of approximately 1.5 s, ensuring minimal latency in deployment completion feedback to the master system. The overall per-installation average of 84 s (including all phases) represents substantial time savings compared to manual installation methodologies, which require human interaction and configuration for each system.

V. DISCUSSION

A. Key Findings

This research validates that purpose-built deployment systems can operate effectively across multiple systems with high reliability. Real-world testing demonstrated that asynchronous concurrent execution provides efficiency, whereas FTP caching mechanisms significantly improve performance for repeated deployments. Network-related failures can be managed through proper timeout and retry logic without affecting system stability or cascading to other nodes.

System Advantages

- Cost: $0 vs $1,500-5,000+ for commercial alternatives
- Performance: 84-second average deployment time per application,<2-second cached redeployment
- Reliability: 83.3% success rate with robust failure isolation
- Caching: 27% of operations eliminated through intelligent package reuse
- Resource Efficiency: Minimal computational overhead on connected systems

VI. CONCLUSIONS AND FUTURE SCOPES

This study presents a network-coordinated distributed software deployment for educational institutions and laboratory environments. The proposed hierarchical orchestration system provides effective automation with minimal implementation complexity, a modest infrastructure, and zero licensing costs. Experimental validation across 2 months of operation demonstrated an overall success rate, 84-second average deployment time. The architecture successfully implements caching mechanisms, reducing repeated deployments to sub-2-second durations. The open-source Python implementation enables academic researchers and IT practitioners to understand, customize, and enhance this system. As educational institutions expand their computing infrastructure, automated deployment systems become essential for operational efficiency. This study demonstrates that sophisticated automation does not require substantial infrastructure investment or commercial licensing. Future research directions emerging from this work include several promising areas for advancement:

- Machine Learning Enhancement:
- Web-Based Administrative Interface:
- Distributed High-Availability Architecture
- Cloud Integration and Hybrid Deployment

ACKNOWLEDGMENT

The authors acknowledge the support and infrastructure resources provided by the laboratory facilities during the development and testing phases of this study. We extend our gratitude to the IT personnel who provided valuable insights into real-world deployment challenges and operational requirements that informed the system design decisions. We thank all individuals who tested the deployment system across diverse laboratory scenarios and providedconstructivefeedbackonitsusabilityandfunctionality.Thisworkwaspartiallysupportedbyinstitutional resources dedicated to infrastructure automation research and development.

REFERENCES

1. R.Kumar, V.Sharma, and A.Patel," Software management infrastructure in academic institutions ,"J.Educational Technology and Computing, vol. 18, no. 3, pp. 234–251, 2024.
2. S.Gupta,R.Singh,andM.Verma," Operational cost analysis of IT infrastructure management, "Int.J. Educational Systems, vol. 25, no. 4, pp. 412–428, 2023.
3. M. Johnson, "Enterprise software deployment: cost-benefit analysis," Educational Technology Review, vol. 14, no. 2, pp. 156–172, 2024.

4. B.Vattikuti,N.V.K.Chaitanya, and A.Jukuntla," Installation automation tool," in Artificial Intelligence and Evolutionary Computations, Springer, 2018, pp. 243–250.
5. U.Manzoor and S.Nefti, "Autonomous agents: Smart network installer and tester," Expert Systems with Applications, vol. 38, no. 1, pp. 884–893, 2011.
6. U.Manzoor and S.Nefti,"QUIET: A Methodology for Autonomous Software Deployment," J.Network and Computer Applications, vol. 33, no. 2, pp. 696–706, 2010.
7. U.Manzoor,S.Nefti, and A.H.Jones, "HERMIT: Autonomous software deployment packages,"ICIC Int.J. Innovative Computing, vol. 8, no. 1, pp. 107–119, 2012.
8. "Autonomous Software Installation using Text Classification, "Int.J. Advanced Computer Science, vol. 8,no.3, pp.123–134,2017.
9. "PDQ Deploy Documentation," PDQ Software,2024.https://documentation.pdq.com/. [Accessed: Nov. 2024].
10. "Deploy applications-Configuration Manager," Microsoft Learn, Dec.24. https://learn.microsoft.com/en-us/configmgr/
11. "SCCM Support, "US Cloud,Nov.2023. https://www.uscloud.com/
12. "Best Software Deployment Tool, "Manage Engine,Dec.2023. https://www.manageengine.com/
13. "Endpoint Central Features," Manage Engine,2024. https://www.manageengine.com/products/desktop-central/
14. "Ansible Documentation,"Red HatInc.,2024.[Online].Available:https://docs.ansible.com/.[Accessed:2024].
15. "Puppet Documentation,"PuppetInc.,2024.[Online].Availablehttps://puppet.com/docs[Accessed:2024].
16. "Chef Documentation,"Chef SoftwareInc.,2024.[Online].Availablehttps://docs.chef.io/[Accessed:2024].
17. "Salt Stack Documentation,"Salt StackInc.,2024.[Online].Available:https://docs.saltproject.io/.[Accessed: 2024].
18. C.Williams and K.Anderson,"Comparative analysis of enterprise deployment solutions,"Systems Administration Quarterly, vol. 22, no. 3, pp. 189–207, 2024.
19. A.Martinez and L.Lee,"Windows Installer Technology,"Systems Administration Journal, vol.34,no. 3,pp.201–219,2023.
20. "Silent Installation Mechanisms," IT Professional Magazine, vol.25,no.5,pp.38–52,2024.
21. G.vanRossum,"Python Programming Language,"ACMSIG PLAN Notices,vol.42,no.4,pp.23–34,2024.
22. "TCP/IP Protocol Architecture,"IEEE Transactionson Networks,vol.38,no.2,pp.124–142,2023.