2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

FINECYCLE: Towards a Full-Cycle Management Paradigm for Robotic Deployment and Development

Haolin Wang, Wang Xi, Zhiyuan Zhu, Chongrong Fang, and Jianping He

***Abstract*— Typical robotic workflows involve deploying applications from servers to robots for testing or distributing validated applications across a fleet to unify capabilities. Because these processes are often slowed by tedious environment configurations, we need a way to improve deployment and development efficiency. Existing solutions typically simplify deployment through containerization; however, they often lack integrated development environments, necessitating repetitive packaging for code modifications and hindering iterative efficiency. This paper proposes FINECYCLE, a management paradigm that encapsulates the entire software stack into a *robotic image*. By facilitating a complete "deploy-develop-store" cycle, FINECYCLE streamlines the transition between cross-host deployment and iterative refinement. Additionally, we open-source image templates compatible with this paradigm to reduce time costs for researchers and foster collaborative progress in the robotics community.**

I. INTRODUCTION

Cross-host deployment, or running the same application across various device types, is an inevitable step in the large-scale deployment of robotic applications. For instance, in scenarios such as smart factories and warehouses, numerous heterogeneous computing hosts are involved in different roles, ranging from servers to onboard computers. Furthermore, robotic application software usually needs to be deployed across these hosts to unify fleet capabilities [1]. However, cross-host deployment typically requires repetitive configuration work on the target hosts, and this process is tedious. Furthermore, differences exist across layers, ranging from hardware structure to software environment, and these differences make deployment problematic.

Existing approaches address cross-host deployment in two ways. Source code–based solutions enable cross-host deployment of application source code but fail to deploy dependencies, thereby still requiring manual configuration and offering little improvement in efficiency. Solutions based on OS-level virtualization (e.g., containerization) enable the deployment of packages containing robotic applications and their dependencies to eliminate manual configuration, but lack integrated development environments, necessitating container rebuild and redeployment for each development [2]–[4]. Overall, these approaches either fail to address pre-configuration challenges or do not maintain the efficiency of development.

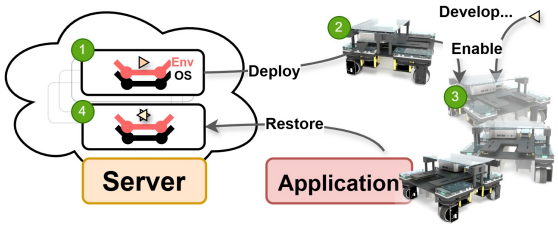

Fig. 1: Example of full-cycle local development process for robotic applications: (1) Image storage on server (operating system "OS" + development environment "Env"); (2) Image deployment to robot host; (3) Development and validation of functional applications; (4) Restoration to server for subsequent reuse.

Therefore, we argue that merely packaging application source code or bundling applications with their dependencies is insufficient. A unified representation that encapsulates applications together with the operating system, dependencies, and development environment is required to achieve configuration-free deployment and development. Disk images naturally provide such a representation by encapsulating all these required elements into a single entity at the data layer. In this work, we refer to such disk images specifically as *robotic images*.

Building on this, we propose a full-cycle process to streamline robotic deployment and development. Here, full-cycle denotes a closed-loop process encompassing robotic image deployment from servers to robotic hosts, support for both local development on robotic hosts and collaborative development across servers and hosts, and eventual restoration to the server for centralized archiving and version management, as shown in Fig. 1. By leveraging a centralized server for storage and distribution of robotic images, this process avoids the complexities of decentralized cross-host deployment between individual robots, thereby ensuring iterative efficiency and consistency throughout the entire image lifecycle.

However, operationalizing this full-cycle process presents several technical challenges. Establishing a unified image representation that facilitates the centralized storage of diverse robotic images is a primary hurdle. Given the shift to robotic images as the primary carrier, establishing new efficient bidirectional deployment and restoration strategies

H. Wang, W. Xi, Z. Zhu, C. Fang, and J. He are with the School of Automation and Intelligent Sensing, Shanghai Jiao Tong University, the Key Laboratory of System Control and Information Processing, Ministry of Education of China, and Shanghai Key Laboratory of Perception and Control in Industrial Network Systems, Shanghai, 200240, China. E-mails: {21-whl, bddwyx, nanjingzhuzhiyuan, crfang, jphe}@sjtu.edu.cn

This work was supported in part by the National Natural Science Foundation of China under Grants 62473260 and 62373247, and in part by the Young Elite Scientist Sponsorship Program by the Cast of China Association for Science and Technology, China under Grant YESS20220357.

between servers and hosts is also critical. Furthermore, for resource-intensive tasks that require server-side development and on-robot testing, a new development strategy is necessary to avoid the prohibitive overhead of redeploying entire images after minor code modifications.

To achieve the full-cycle process, this paper introduces FINECYCLE, a management paradigm for robotic deployment and development. FINECYCLE (i) utilizes hardware virtualization to provide a consistent storage foundation for diverse robotic images; (ii) implements tiered deployment strategies to facilitate bidirectional image circulation; and (iii) leverages mobile disk media to ensure iterative efficiency by bridging high-performance computation with direct on-host execution.

The contributions of this paper are summarized as follows:

1) We propose a full-cycle management paradigm for robotic deployment and development. The paradigm integrates unified storage, bidirectional deployment, and iterative development procedures for robotic images, enabling a full-cycle process without pre-configuration and thereby improving the efficiency of robotic deployment and development.
2) We evaluate various image deployment strategies and validate their suitability for diverse requirements. Additionally, we analyze virtualization overhead and validate the feasibility of virtualized robotic hosts.
3) We provide an open-source, standardized robotic image template. This pre-configured baseline eliminates repetitive dependency configuration, offering researchers a consistent environment, thereby reducing development time and fostering collaborative progress in the robotics community.

II. RELATED WORKS

Robotic application management approaches are expected to enhance the efficiency of full-cycle deployment and development. Early work primarily relied on source code management [5], [6], which provided foundational development solutions such as Continuous Integration for automated code merging and testing [7], Continuous Delivery for streamlined release preparation [8], and Continuous Deployment for direct application updates [9]. However, due to their lack of dependency encapsulation, these approaches could only effectively improve the efficiency of application deployment and development when all required dependencies were pre-configured manually. This fundamental limitation prevents full-cycle deployment and development without configuration, thereby limiting the efficiency of these processes.

Alternatively, virtualization serves as a robust approach to enhancing deployment efficiency. By encapsulating the entire software environment into portable units, virtualization effectively eliminates the need for repetitive manual setup on target hosts. Broadly, virtualization can be classified into two categories. The first is OS-level virtualization, represented by container technologies such as Docker [10], which package robotic applications and their dependencies into containers to ensure they run consistently across different hosts. The second is hardware-level virtualization [11], which encapsulates full robotic images. This enables a single host to run diverse images simultaneously. Furthermore, hardware-level virtualization allows various hosts employing this technology to be organized into a cluster. By utilizing shared storage to maintain a unified image repository, such clusters enable robotic images to be migrated seamlessly between hosts, thereby enhancing their portability.

Due to the availability of cluster orchestration tools for containers, such as Kubernetes [12], prior work has largely focused on OS-level virtualization. Container-based approaches enable automated ROS environment setup [13], [14] and cluster orchestration [2]–[4]. These contributions have substantially improved the deployment efficiency of robotic applications encapsulated in containers. However, as these lightweight containers do not integrate operating systems and development environments, robotic application development cannot be performed within a deployed container, resulting in each development cycle requiring container rebuilding and redeployment. Consequently, they cannot enhance the efficiency of development.

To enhance the efficiency of the full-cycle process, we propose a full-cycle management paradigm for robotic applications. It integrates hardware virtualization and disk cloning technologies to realize unified storage, bidirectional deployment, and iterative development.

III. ROBOT SYSTEM MANAGEMENT PARADIGM

A. Theoretical Paradigm

The FINECYCLE paradigm integrates three synergetic components as illustrated in Fig. 2. Unified Storage (steps 1 and 4) centrally manages robotic images in the form of VMs through cloning and templating. Bidirectional Deployment (steps 2 and 3) employs deployment strategies for image deployment and restoration. Iterative Development (steps 5 and 6) supports local and collaborative workflows, leveraging mobile disk media for server-side development and on-robot validation.

1) Unified Storage: Within the proposed full-cycle management paradigm, a key challenge lies in establishing a unified form of robotic images. To overcome this, the paradigm leverages the hypervisor layer to encapsulate robotic applications and their required components into complete disk images. By utilizing the hypervisor's ability to virtualize the underlying hardware, the paradigm provides the required virtual hardware for images and enables images to be executed as virtual machines (VMs) consistently across heterogeneous hosts.

Moreover, leveraging the hypervisor layer, the paradigm supports VM templating and VM cloning. VM templating allows developed images to be preserved as a VM template for versioned management. Complementary VM cloning then facilitates rapid provisioning by replicating these VM templates into ready-to-use VMs, ensuring that identical images can be executed without repetitive manual setup. In this way, the paradigm not only provides a consistent

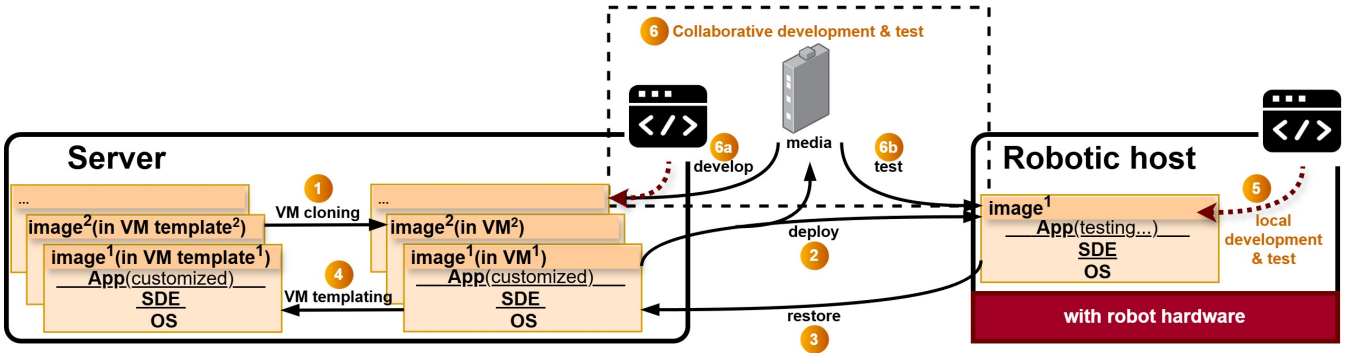

Fig. 2: Full-cycle management paradigm for robotic applications: (1) **VM Cloning** rapidly instantiates robotic images from standardized templates; (2) **Forward Deployment** distributes images from the server to robotic hosts; (3) **Backward Restoration** transfers validated applications back to the repository; (4) **VM Templating** saves updated images as new baselines for version management; (5) **Local Development and Test** are performed directly on the robotic host; and (6) **Collaborative Development and Test** leverage mobile disk media to bridge high-performance server computation (**6a**) with subsequent on-robot hardware validation (**6b**).

storage foundation but also establishes the technical basis for subsequent image deployment and application development.

2) Bidirectional Deployment: As an integral phase of full-cycle management, image deployment facilitates the circulation of robotic images across the system. Since the central server stores robotic images in the form of VMs, the deployment process essentially entails transferring these images from the VMs to the target robotic hosts. To accommodate varying hardware support, robotic hosts are categorized into two types: *virtualized hosts*, which integrate a hypervisor and can join the server's virtualization cluster; and *bare-metal hosts*, which operate without a virtualization layer and reflect the common state of most robotic hosts today.

To accommodate both host types, the paradigm employs three complementary deployment strategies—Cluster Live Migration (CLM), Cluster Disk Cloning (CDC), and Direct Disk Cloning (DDC), primarily realized through VM migration and disk cloning, as shown in Table I.

TABLE I: Summary of deployment strategies

Strategy	Brief Description
CLM	VM migration within cluster, fast relocation on virtualized hosts.
CDC	Full disk cloning within cluster, stable I/O on virtualized hosts.
DDC	Block-level disk cloning to bare-metal hosts ensures native execution.

CLM leverages hypervisor-native live migration to transfer only VM states such as CPU registers, memory pages, and I/O buffers. This achieves sub-second relocation with negligible downtime but depends on cluster shared storage, making it ideal for quickly redeploying non–real-time images.

While CLM allows robotic images to execute on the virtualized host, the image data remains on shared storage, which limits I/O efficiency. CDC addresses this by cloning the entire robotic image from shared storage to the host's local storage, thereby completing the full deployment process for virtualized hosts. By resuming execution with full local disk access, CDC ensures stable I/O performance, making it particularly suited for real-time or I/O-sensitive tasks when high-bandwidth interconnects are available.

DDC operates without a virtualization layer by cloning images block by block—including boot sector, file system, and application binaries—onto bare-metal hosts. This ensures native performance and predictable hardware behavior. Unlike CLM and CDC, which rely on the virtualization layer to abstract heterogeneous hardware, DDC operates directly on physical hardware. Therefore, DDC requires compatible host architectures to ensure successful deployment. In practice, adopting hardware virtualization to transform robotic hosts into virtualized hosts is recommended, as this alleviates hardware-specific constraints during deployment.[14].

By balancing speed, consistency, and native execution, these strategies provide the technical foundation for efficient image deployment without manual reconfiguration. Specifically, when deploying images from the central server to robotic hosts, the approach is tailored to the host type: virtualized hosts leverage CLM or CDC, while bare-metal hosts utilize DDC to ensure native performance.

Conversely, the paradigm supports the restoration of validated images from robotic hosts back to the server. By utilizing the same set of strategies, finalized robotic images are stored back in the central server, thereby ensuring efficient bidirectional deployment that facilitates the complete full-cycle process.

3) Iterative Development: Building upon the established deployment strategies, the paradigm facilitates two primary development workflows.

Local development is achieved by deploying robotic images directly onto robotic hosts using the deployment strategies. This approach enables developers to refine and validate applications within the actual target robotic hosts, ensuring that software behavior remains consistent with the robot's physical hardware constraints.

Collaborative development leverages the high-performance hardware of the central server for resource-intensive tasks. In this workflow, the robotic image is deployed onto mobile disk media using DDC. The media is first connected to the central server, where the image is accessed via hardware pass-through for rapid development and compilation. Subsequently, the media is physically transferred and booted on the robotic host for immediate testing, thereby streamlining the develop-on-server and test-on-host iteration loop.

Notably, the use of mobile disk media is specifically optimized for development iterations rather than long-term deployment. Due to the I/O throughput and latency limitations inherent in USB interfaces and external storage media, this method is designed to provide a portable environment for rapid testing. Once the development phase is finalized, robotic images are instead migrated to the host's internal storage via DDC to ensure optimal performance and stability for long-term operational tasks.

B. Performance and Overhead Discussion

1) Deployment Efficiency: The deployment strategies can be evaluated in terms of deployment time and disk I/O performance. CLM minimizes deployment time by migrating only the operational state; since the image data is not migrated, the application continues to rely on the shared storage, which reduces I/O throughput and increases I/O latency. This makes it suitable for robotic applications that perform non–real-time tasks and prioritize rapid redeployment over disk performance.

In contrast, disk-based deployment entails the transfer of the entire disk image, thereby incurring longer deployment times. The performance of both CDC for virtualized hosts and DDC for bare-metal hosts is predominantly constrained by network bandwidth. Nevertheless, upon completion of the deployment, images operate directly on local storage rather than shared storage, which ensures stable disk I/O performance and consistent system responsiveness.

Taken together, the combination of different strategies supports both rapid redeployment and high-performance execution, thereby ensuring effective full-cycle management of robotic applications for diverse robotic conditions.

2) Virtualization Overhead: Virtualized hosts employ hardware virtualization technology, which introduces an inherent virtualization layer (the hypervisor) between guest images and the physical hardware. As a result, robotic applications running inside virtual machines must access disk, CPU, and network resources through this mediation layer, which inevitably produces additional latency and resource bookkeeping compared with bare-metal execution.

To reduce overhead, we apply best practices: use of virtio drivers for block and network I/O, VFIO-based PCI/USB device passthrough for sensors/actuators, and tuning of I/O schedulers.

Taken together, these design choices ensure that robotic images running on virtualized hosts can utilize hardware and interact with the environment in a manner closely comparable to native execution on bare-metal hosts. While some virtualization overhead is unavoidable in theory, our experimental evaluation demonstrates that this residual overhead is minimal and does not significantly impact the performance of typical robotic applications. Minimal and does not significantly impact the performance of typical robotic applications.

C. Image Template

To lower the entry barrier for developers and further enhance development efficiency, the paradigm introduces a standardized image template. The template is a pre-configured robotic image that encapsulates the operating system (Ubuntu 22.04), Robot Operating System (ROS 2), and essential development tools, as detailed in Table II. By providing a ready-to-use baseline, these templates eliminate repetitive environment setup and allow researchers to focus solely on application functionality.

TABLE II: Standard Contents of the FINECYCLE Image Template

Category	Component	Description / Version
Operating System	Ubuntu 22.04 LTS	Long-term support base for stability.
Middleware	ROS 2 Humble	Standardized robotic communication.
Build System	CMake	Optimized for large-scale C++ projects.
Development IDE	CLion / Pycharm	Pre-configured for ROS 2 indexing.
Simulation	Gazebo / RViz2	Essential for offline testing and visualization.
Tools & Libs	Git, Eigen	Comprehensive toolkit for robotic algorithms.

Developers can obtain and leverage the open-source image template[1] through two primary technical pathways. For physical execution, the template can be directly deployed onto bare-metal hosts, enabling rapid hardware setup. Alternatively, the template can be restored to the server by DDC to establish a standardized VM Template. This integration allows developers to utilize VM cloning for the rapid provisioning of multiple images with consistent configurations, facilitating parallel testing and development.

When integrated with the full-cycle management paradigm, the template circulates seamlessly between central servers and distributed robotic hosts, executing immediately upon arrival without manual reconfiguration. This dual-path approach significantly reduces setup overhead, enhances the efficiency of the entire robotic application lifecycle, and promotes high reproducibility and open collaboration within the robotics community.

D. Domain-Specific Deployment for Robotic Environments

To validate the practicality of the proposed full-cycle management paradigm, domain-specific configurations were applied to adapt it to robotic environments. KVM [15]

[1]The open-source image templates are available at `https://github.com/FINS-Fines/FineCycle/`

was chosen as the underlying hardware virtualization technology, with Proxmox Virtual Environment (PVE) as the management layer, providing a robust platform for virtual machine deployment, monitoring, and management across heterogeneous robotic hardware.

To ensure reliable operation, virtualized hosts allocate all CPU, memory, and storage resources to the migrated system, while also passing through all interaction interfaces—including sensors, actuators, and communication devices—directly to the virtual machine. This design allows the migrated robotic application to fully utilize host resources and interact with the external environment in a manner equivalent to running on a bare-metal host. Furthermore, to address the limitation that PVE natively supports only wired connections, which is unsuitable for mobile robotic scenarios, wireless connectivity is enabled by passing through a USB wireless adapter to the virtual machine, while the host maintains connectivity through its built-in wireless card. This approach avoids the configuration complexity of traditional wireless bridging and the strict requirements on wireless access points, while ensuring hardware-level isolation, making it suitable for mobile robots that require independent network access.

IV. Experiments

To validate the feasibility of the proposed full-cycle management paradigm, three categories of experiments were conducted. The first set evaluates the deployment time and disk performance under different deployment schemes, aiming to identify their suitability for distinct deployment scenarios and requirements. The second set measures the virtualization overhead of virtualized hosts to verify that the introduction of the virtualization layer does not compromise the hardware performance required for robotic applications. The third set presents a case study on a robotic navigation application, demonstrating the practical feasibility of the full-cycle management paradigm and its effectiveness in enhancing deployment and development efficiency. The evaluation points for these experiments are summarized in Table III.

TABLE III: Summary of Experimental Evaluation Metrics

Experiment	**Key Metrics**	**Evaluation Target**
Deployment Performance	Time throughout	Objectively evaluating the efficiency of different deployment strategies across varied conditions.
Virtualization Overhead	CPU, Network	Objectively assessing the execution performance and overhead of virtualized hosts compared to bare-metal hosts.
Case Study	Launch status	Qualitatively verifying the feasibility and continuity of the end-to-end, full-cycle management workflow.

1) Experimental Setup: The paradigm was implemented using KVM as the virtualization technology and *PVE* as the management layer.

2) Hardware Configuration: All hosts were equipped with identical hardware to eliminate hardware bias: an *Intel i7-1260P* processor (12 cores, 16 threads), 16GB 3200MT/s RAM, and a 500GB SSD mini PC.

3) Software Configuration: Each robotic image was configured with Ubuntu 22.04 LTS as the primary operating system. The software stack included essential system dependencies and development tools, such as ROS 2 Humble and the CLion IDE, providing a standardized baseline for both local and collaborative development tasks.

A. Deployment Performance of Image

This experiment evaluates the deployment performance of a standardized robotic image template on two types of robotic hosts: bare-metal hosts without a pre-installed system and virtualized hosts managed by PVE within a cluster. The evaluation focuses on deployment time and post-deployment disk I/O performance to characterize the difference between bare-metal and virtualization-based deployments, and to assess their suitability for real-time and non-real-time robotic applications.

1) Experimental Design: Three deployment approaches were evaluated: CLM, CDC, and DDC. For each approach, two key metrics were measured: total deployment time and post-deployment disk I/O performance. The latter was assessed using `fio` with a 10 GB test file under sequential read/write mode (block size 1 MB, direct I/O, single job).

2) Results: As shown in Table IV, deployment performance was quantified using two metrics. Deployment time was measured end-to-end from initiation to image availability on the target host. Disk I/O performance was obtained from `fio` results, where throughput (MB/s) reflects sustained sequential transfer rate and latency (µs) reflects per-operation response time.

Experimental results indicate that CLM achieves the fastest deployment, completing in under 1s, since it transfers only the operational state of the image. However, as disk data is not migrated, I/O performance is significantly reduced, making CLM suitable only for non–real-time applications.

In contrast, both CDC and DDC deployment times are primarily constrained by network bandwidth. Under identical disk sizes and network conditions, the difference in deployment time between the two approaches remains within 5%, indicating that their performance is largely comparable. Since both methods involve full disk cloning, images are able to operate directly on local storage after deployment. Moreover, the differences in I/O throughput and latency remain within 8%, demonstrating that images deployed via CDC on virtualized hosts can achieve disk performance comparable to those deployed via DDC on bare-metal hosts. These findings suggest that both CDC and DDC are suitable for real-time applications while offering complementary options for diverse deployment scenarios.

TABLE IV: Comparison of Image Deployment Performance

Deployment type	Deployment time	Write speed (Write Latency)	Read speed (Read Latency)
CLM	< 1 s	47.6 MB/s (21997.96 µs)	75.7 MB/s (13848.90 µs)
CDC	7 min 35 s	894 MB/s (1140.94 µs)	1421 MB/s (717.98 µs)
DDC	7 min 22 s	927 MB/s (1056.75 µs)	1527 MB/s (685.42 µs)

TABLE V: Comparison of CPU Performance between Bare-metal and Virtualized Hosts

Host type	HardInfo			UnixBench			Iperf	
	Blowfish	Fibonacci	N-queen	Context Switching	System Call Overhead	File Copy	TCP up	TCP down
Bare-metal Host	0.37s	0.20s	9.72s	6330.1	4930.0	11242.8	228Mbits	60.9Mbits
Virtualized Host	0.40s	0.22s	9.98s	5166.9	4820.7	9318.3	225Mbits	53.8Mbits

Importantly, all images successfully booted and executed immediately after deployed, demonstrating that these strategies enable application deployment without additional preconfiguration. Overall, the proposed paradigm provides multiple deployment strategies, effectively supporting diverse host types and application requirements.

B. Virtualization Overhead Experiment

Virtualization overhead primarily affects three critical aspects of system performance: disk I/O performance, CPU performance, and network communication efficiency. While the disk I/O performance differences between bare-metal and virtualized hosts were evaluated in the previous image deployment experiments, this experiment focuses on quantifying the overhead in CPU computation and end-to-end network performance. A standardized robotic image template was deployed on both host types. By conducting benchmark tests, we aim to determine whether the hypervisor-induced latency and throughput degradation impact typical robotic applications, such as real-time sensor data transmission and heavy algorithmic processing.

1) Experimental Design: To evaluate the performance impact of virtualization, three widely used benchmarking tools were employed. HardInfo[2] was used for algorithmic tasks, and UnixBench for system-level operations. To analyze network communication, Iperf was utilized to measure end-to-end throughput between the host and a remote client, simulating common robotic telemetry and control scenarios. The reported metrics include execution time (s), relative throughput scores, and network bandwidth (Gbps).

2) Results and Discussion: As shown in Table V, lower values indicate better performance for HardInfo algorithmic benchmarks (shorter execution time), while higher values represent better performance for UnixBench system-level tests and iperf network throughput.

The HardInfo benchmarks show minimal slowdown in CPU-intensive computations due to virtualization: Blowfish execution time increased by 8.1%, Fibonacci by 10%, and the N-queen by 2.7%. UnixBench measurements indicate moderate overhead for system-level operations, with context switching reduced by 18.4% and file copy throughput decreasing by 17.1%.

Regarding network performance, the iperf results demonstrate that virtualization introduces varying degrees of overhead for communication. The TCP uplink throughput experienced a negligible decrease of 1.3%, indicating that data transmission from the robot to the server is highly efficient. In contrast, the TCP downlink throughput showed a reduction of 11.7%.

These results demonstrate that while CPU performance and network throughput are affected by virtualization—primarily in system-level operations and data reception—the impact on pure algorithmic computations remains marginal. The residual overhead is small enough to confirm that virtualized hosts are practical for both deployment and development in robotic applications, providing a balance between flexibility and performance.

C. Case Study: Full-Cycle Management Validation with Navigation Application

To validate the closed-loop feasibility of the proposed full-cycle management paradigm in a realistic robotic scenario, we conduct an end-to-end case study using a navigation application. Specifically, the application is based on the Nav2 tb3_simulation_launch demo; the development task focuses on the installation and verified execution of this simulation demo within the robotic image. As previous sections already provide quantitative evaluations of deployment performance and virtualization overhead, this case study focuses on verifying that the complete workflow—*deploy* $\rightarrow$ *develop* $\rightarrow$ *restore* $\rightarrow$ *redeploy*—can be executed without any preconfiguration, as shown in Fig. 3.

1) Implementation Procedure: The procedure consists of three main steps:

Image Provisioning: The standardized VM template is cloned as a VM on the storage server as the baseline image.

Deployment and Development: The image within the VM is deployed to a virtualized robotic host, where the navigation application is developed. Domain-specific configurations are applied to adapt the image to the host hardware.

Restoration and Redeployment: After development, the updated image is restored to the storage server and redeployed onto a bare-metal host. This step demonstrates

[2]HardInfo is an open-source benchmarking tool for Linux systems, available at `https://github.com/lpereira/hardinfo`.

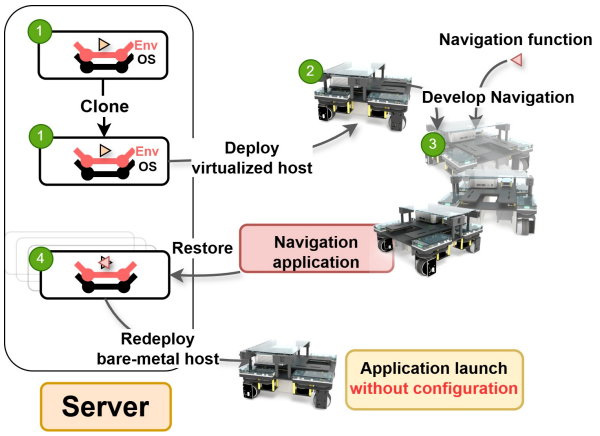

Fig. 3: Full-cycle workflow for the navigation application case study: (1) image instantiation via cloning on the server; (2) deployment to a virtualized host; (3) in-place navigation development and refinement; (4) restoration of the updated image to the server; and final redeployment to a bare-metal host for configuration-free execution test.

that the full cycle—deployment, development, restoration, and redeployment—can be completed without any pre-configuration.

TABLE VI: Navigation Application Launch Feasibility Across Hosts

Host Type	Launch Feasibility
Virtualized host	✓
Bare-metal host	✓

2) Results: Table VI shows that the navigation application successfully launched on both virtualized and bare-metal hosts. This confirms that the full-cycle management paradigm preserves environment consistency across heterogeneous hosts, eliminates manual setup, and supports seamless execution of the complete deployment–development–restoration workflow.

V. Conclusion

This paper proposes FineCycle, a management paradigm that utilizes virtualization-based packaging for unified storage, enabling bidirectional deployment and iterative development across servers and robotic hosts. Experimental results demonstrate that virtualization introduces only limited CPU, disk I/O overhead, and network communication efficiency, which does not materially affect the performance of robotic images. Deployment strategies—including Cluster Live Migration, Cluster Disk Cloning, and Direct Disk Cloning—provide complementary options that balance deployment speed and post-deployment performance for diverse robotic scenarios.

A case study with a navigation application further validates the paradigm, confirming that the complete workflow (*deploy* $\rightarrow$ *develop* $\rightarrow$ *restore* $\rightarrow$ *redeploy*) can be executed without pre-configuration while maintaining environment consistency across heterogeneous hosts. Overall, the proposed paradigm enables seamless full-cycle deployment and development while reducing manual configuration overhead. Furthermore, by releasing a standardized robotic image template as open source, it promotes reproducibility, fosters collaborative development, and supports a sustainable ecosystem for robotics research and applications.

References

[1] Bastian Lampe, Lennart Reiher, Lukas Zanger, Timo Woopen, Raphael Van Kempen, and Lutz Eckstein. Robotkube: Orchestrating large-scale cooperative multi-robot systems with kubernetes and ros. In *International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

[2] Francesco Lumpp, Franco Fummi, Hiren D Patel, and Nicola Bombieri. Containerization and orchestration of software for autonomous mobile robots: A case study of mixed-criticality tasks across edge-cloud computing platforms. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[3] Francesco Lumpp, Franco Fummi, Hiren D Patel, and Nicola Bombieri. Enabling kubernetes orchestration of mixed-criticality software for autonomous mobile robots. *IEEE Transactions on Robotics*, 2023.

[4] Francesco Lumpp, Marco Panato, Nicola Bombieri, and Franco Fummi. A design flow based on docker and kubernetes for ros-based robotic software applications. *ACM Transactions on Embedded Computing Systems*, 2024.

[5] Serhat Uzunbayir and Kaan Kurtel. A review of source code management tools for continuous software development. In *International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2018.

[6] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 2017.

[7] Martin Fowler and Matthew Foemmel. Continuous integration, 2006.

[8] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[9] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 2017.

[10] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.

[11] Robert J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 1981.

[12] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson. *Kubernetes: up and running: dive into the future of infrastructure*. " O'Reilly Media, Inc.", 2022.

[13] Pedro Melo, Rafael Arrais, Sérgio Teixeira, and Germano Veiga. A container-based framework for developing ros applications. In *International Conference on Industrial Informatics (INDIN)*. IEEE, 2022.

[14] Shunki Shibuya, Kazuyuki Kobayashi, Tomoyuki Ohkubo, Kajiro Watanabe, Kaiqiao Tian, Nashwan J Sebi, and Ka C Cheok. Seamless rapid prototyping with docker container for mobile robot development. In *Annual Conference of the Society of Instrument and Control Engineers (SICE)*. IEEE, 2022.

[15] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Linux Symposium*, 2007.