Communication

Reproducibility of Open-Source Virtual Surgery Frameworks

Adhirit Prabhugee and Anton V. Sinitskiy *

Posted Date: 15 June 2026

doi: 10.20944/preprints202606.1100.v1

Keywords: reproducibility audit; robotic surgery; negative results; scientific computing

 doi:10.20944/preprints202606.1100.v1

Communication

Reproducibility of Open-Source Virtual Surgery Frameworks

Adhirit Prabhugee [1] and Anton V. Sinitskiy [2,3,*]

[1] Algonquin Regional High School, Northborough, MA 01532, USA
[2] College of Professional Studies, Northeastern University, Boston, MA 02115, USA
[3] ML LC, Southborough, MA 01772, USA
* Correspondence: a.sinitskiy@northeastern.edu

Abstract

Virtual surgery environments are increasingly used in surgical training, robotics, and reinforcement learning, yet their practical reproducibility remains insufficiently studied. Prior work focuses primarily on simulation capabilities and task performance, with limited attention to installation effort, dependency stability, and long-term usability. This paper presents a reproducibility audit of several open-source frameworks, including Surgical Gym, iMSTK, and SOFA, based on hands-on installation and execution attempts in up-to-date computing environments. The results reveal substantial heterogeneity in both architecture and reproducibility outcomes. Surgical Gym failed due to reliance on a deprecated Isaac Gym interface that is no longer supported in current Isaac Sim releases. iMSTK failed to build because of unresolved dependencies, and its archived status makes an upstream fix unlikely. SOFA installation was hindered by extensive system-level dependencies and distribution-specific package mismatches. Across all systems, failures were driven primarily by external dependency ecosystems rather than core simulation logic. These findings indicate that reproducibility in virtual surgery frameworks is strongly shaped by architectural design and dependency surfaces. The study highlights a gap between open-source availability and practical usability, underscoring the need for improved dependency management, environment specification, and reproducibility reporting in surgical simulation research.

Keywords: reproducibility audit; robotic surgery; negative results; scientific computing

Introduction

Surgical care is a fundamental component of modern healthcare systems and plays a critical role in treating trauma, cancer, cardiovascular disease, and numerous other life-threatening conditions. Despite its importance, access to surgical care remains uneven worldwide. Estimates suggest that roughly five billion people lack access to safe and affordable surgical services, and more than a hundred of millions of additional procedures are required annually to meet global health needs.[1] The gap is driven by shortages of trained surgeons, infrastructure limitations, and uneven distribution of healthcare resources. These systemic constraints increase the importance of scalable training methods that can improve surgical skill acquisition and evaluation.

Training surgeons is particularly challenging because surgery requires complex psychomotor skills, extensive procedural knowledge, and real-time decision making. Traditional surgical education follows an apprenticeship model in which trainees gradually acquire expertise through supervised participation in operations. However, modern healthcare constraints limit opportunities for hands-on practice. Studies of surgical residency programs report that trainees frequently experience limited access to operative cases, which reduces the number of procedures they can perform during training and can leave graduates feeling insufficiently prepared for independent practice.[2] At the same time, surgical training is costly, requiring specialized equipment, operating room access, and expert supervision. The need to balance patient safety with educational

 doi:10.20944/preprints202606.1100.v1

requirements further restricts the amount of experimentation and repetition that trainees can perform during real procedures.

Computational simulation and virtual surgery environments offer a promising solution to these challenges. By modeling anatomy, tools–tissue interaction, and surgical workflows in software, simulation platforms enable trainees and researchers to experiment in a controlled and repeatable environment without risk to patients. Virtual environments also support automated data collection, objective performance evaluation, and large-scale experimentation with robotic control algorithms or artificial intelligence methods. The field of virtual surgery environments has been consistently growing over the last few decades (Figure 1). Simulation-based training approaches, including extended-reality systems and virtual environments, have been shown to improve surgical skill acquisition while providing a lower-risk and more accessible training setting than traditional operating-room experience.[3,4] These capabilities motivate continued development of virtual surgery frameworks and highlight the importance of assessing their practical usability and reproducibility in research settings.

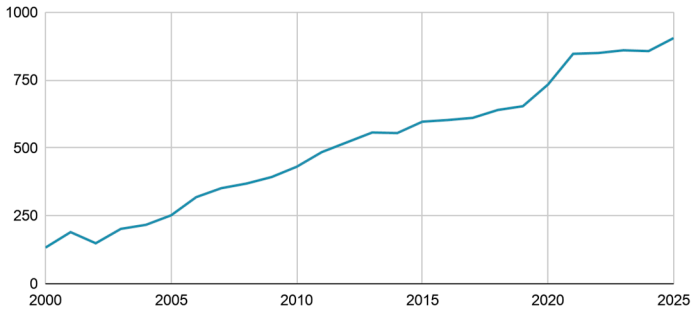

Figure 1. The number of papers on VSE has been steadily growing over the last 25 years. (The number of articles is measured in Google Scholar by searching for "Virtual Surgery" within the corresponding year.).

Virtual simulation has become an important tool for surgical training and research, particularly in robotic surgery and reinforcement-learning–based automation. Real surgical training opportunities are limited by cost, ethical constraints, and the scarcity of expert supervision, motivating the development of virtual surgery environments (VSEs) that simulate surgical tasks and tools–tissue interaction. A number of open-source frameworks have been developed to support surgical simulation. Major examples include SOFA, iMSTK, OpenSurgSim, and AMBF, which provide reusable components for physics simulation, collision handling, and haptic interaction.[5–8] These platforms form the basis of many interactive surgical simulators but differ substantially in architecture, dependencies, and usability. SOFA is a framework for real-time deformable simulation with a modular scene-graph architecture. While flexible, this design can require significant configuration effort because simulations must combine numerous components such as solvers, collision models, and mappings between physical and visual representations.[8,9] Similarly, iMSTK provides a C++ toolkit for surgical simulation and haptics integration, but requires a complex build toolchain and multiple dependencies. The project was archived in 2025, raising concerns about long-term maintainability.[10] More recently, reinforcement-learning–oriented environments have emerged for surgical robotics research. Platforms such as SurRoL, LapGym, Surgical Gym, and ORBIT-Surgical provide benchmark tasks and simulation environments for training control policies.[11–13] These systems often rely on robotics simulators such as PyBullet, SOFA, or NVIDIA Isaac Sim. Despite enabling large-scale experiments, these environments introduce new reproducibility challenges. Several platforms rely on deprecated APIs or tightly pinned software versions. For example, SurRoL depends on the deprecated Gym API and requires Python 3.7,[14] increasing compatibility issues for newer software stacks. Other platforms depend on unmaintained physics engines or GPU-specific toolchains, such as NVIDIA FleX or CUDA-based simulation libraries.

 doi:10.20944/preprints202606.1100.v1

Hardware constraints also affect reproducibility. GPU-accelerated simulation frameworks such as Isaac Gym or Isaac Sim enable faster reinforcement-learning experiments but require specific NVIDIA GPU hardware and driver configurations, limiting portability across computing environments.[12,15] Recent comparative studies of surgical simulation environments emphasize these issues but rarely quantify installation effort. Instead, they report qualitative indicators such as dependency fragility, hardware lock-in, and software version constraints.[6,15,16] Consequently, while many open-source surgical simulators exist, the literature provides little systematic evaluation of how difficult these systems are to install, configure, and run on modern computing platforms. This gap motivates reproducibility-focused audits of surgical simulation frameworks, examining not only their simulation capabilities but also the practical effort required to deploy them on contemporary hardware and software stacks.

Results

Available open-source surgical virtual reality software systems are highly diverse in their simulation focus, technical stacks, licensing, and intended architectural roles. Across the audited set (Figure 2), we observed at least three distinct "center-of-gravity" archetypes (Table 1). Surgical Gym targets GPU-accelerated reinforcement learning and robotics-centric interaction loops, implemented on Omniverse Isaac Sim stack from NVIDIA and explicitly built on top of Isaac Sim's omni.isaac.core and omni.isaac.gym frameworks (Python-first orchestration with simulator-coupled extensions). In contrast, iMSTK positions itself as a C++ toolkit for interactive, multi-modal surgical simulation scenarios, emphasizing modular real-time simulation components and integration with haptics and rendering backends. SOFA occupies a broader multi-physics engine role, using an open-core, modular architecture and prioritizing real-time continuum mechanics and deformable simulation. These differences were also reflected in our hands-on audit workflow, where each system demanded a different execution substrate (simulator runtime versus native toolchain compilation) and a different notion of "success" (policy rollouts versus interactive scenes).

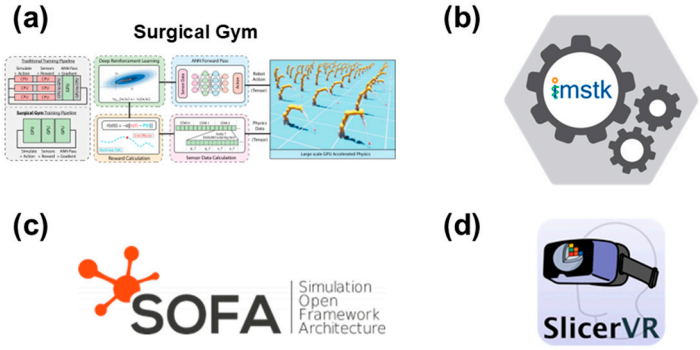

Figure 2. Multiple VSE frameworks, including (a) Surgical Gym, (b) iMSTK, (c) SOFA, (d) Slicer VR, have been considered in this work.

Table 1. Surgical VR Frameworks Comparison.

VR Framework	Physics / simulation focus	Language / stack	License
Surgical Gym	Robotics-centric simulation using NVIDIA Isaac Sim	Python + NVIDIA Omniverse / Isaac Sim	MIT
iMSTK	Real-time medical simulation (tools, deformable tissue, interaction)	C++ modular toolkit	Apache 2.0

 doi:10.20944/preprints202606.1100.v1

SOFA	Multi-model physics engine (deformables, constraints, contact)	C++ core + plugin ecosystem	LGPL-2.1
IMHOTEP	Visualization-focused (patient anatomy, imaging data)	Unity + C#	BSD
Slicer VR	Imaging visualization & scene interaction (VTK/Slicer modules)	C++ / VTK / 3D Slicer ecosystem	Apache 2.0

Licensing and distribution models further reinforced this heterogeneity. Surgical Gym is distributed as a research codebase layered over an external proprietary simulator runtime (Isaac Sim), whereas iMSTK is released under Apache 2.0 according to Kitware public descriptions, and SOFA core is LGPL with carve-outs for specific subdirectories. Visualization-oriented VR in the medical imaging ecosystem offers yet another pattern: SlicerVirtualReality is an extension to 3D Slicer with its own release and platform constraints, and community reports indicate Windows-only availability in at least some recent configurations, with breakage across Slicer major versions. This breadth supports the interpretation that open-source virtual surgery frameworks is not a single interchangeable category, but a spectrum spanning robotics training, physics engines, and clinician-facing visualization.

From a reproducibility perspective, the diversity is consequential because each archetype couples to a different dependency surface. Simulator-bound systems concentrate risk in vendor release cadence and extension availability, while C++ toolkits concentrate risk in compiler, dependency discovery, and transitive third-party source availability. In our audit, these differences translated into qualitatively different failure modes and remediation costs, which motivates reporting results per framework rather than aggregating into a single "ran or did not run" outcome.

This diversity is not only technical but also architectural (Table 2): the audited systems occupy distinct strata in a virtual-surgery software stack, from reinforcement-learning backends for robotic policy training (Surgical Gym) to mid-level C++ toolkits for assembling interactive procedure simulators (iMSTK) and general-purpose deformable physics engines (SOFA). It also includes clinician-facing VR front ends centered on Unity-based interaction and medical data ingestion (IMHOTEP) and VR extensions that deliberately reuse existing clinical imaging infrastructure rather than re-implement it (SlicerVirtualReality within 3D Slicer). Collectively, these frameworks are largely complementary rather than interchangeable, so reproducibility must be interpreted relative to the intended architectural role of each system.

Table 2. Comparative architectural strategies, implementation shapes, and intended system roles of audited open-source virtual surgery frameworks.

Framework	**Core strategy (specific)**	**Architectural shape**	**Best architectural role**
Surgical Gym	Optimize **large-scale policy training for surgical robots** by standardizing tasks, observations, and reward structures on top of Isaac Sim	Python package following Isaac Sim / Gym task abstractions	Backend for autonomous surgical robot learning
iMSTK	Enable **rapid assembly of interactive surgical skill simulators** by exposing mid-level abstractions for tools, tissue, and interaction loops	C++ simulation toolkit with examples and CMake build	Custom interactive surgical simulators (research-focused, legacy)

 doi:10.20944/preprints202606.1100.v1

SOFA	Provide a **generalizable, research-grade deformable physics engine** where complex biomechanical behavior is composed from modular simulation components	Large multi-library engine with plugin ecosystem	Physics and biomechanics backbone
IMHOTEP	Deliver a **clinician-facing VR workflow for pre-operative planning** by centering the codebase around Unity scenes, interaction tools, and medical data ingestion	Unity application project	Front-end VR planning and visualization
Slicer VR	Extend **3D Slicer imaging and segmentation pipeline into VR** without re-implementing medical data infrastructure	Host-extension module with superbuild	VR interface for clinical imaging workflows

This taxonomy also motivated deliberate deprioritization of some candidates as primary physics-based surgical simulation baselines. IMHOTEP is presented primarily as a Unity-based VR framework for visualizing and interacting with multimodal patient data in support of planning and education, rather than as a validated tissue–tool interaction simulator with haptics-centric design goals. In addition, its public-facing descriptions emphasize surgeon workspaces and medical data ingestion, with no explicit positioning as a robotics integration framework (for example, ROS interfaces or closed-loop surgical robot control), so robot control integration is appropriately rated low unless substantial external integration is added. SlicerVirtualReality is similarly valuable but maintainers explicitly document that the extension currently functions on Windows, with Linux support described as experimental and macOS lacking backend support. In addition, users sometimes report the extension broken for current stable or preview versions, and install availability gaps for specific Slicer releases. Finally, users report rendering instability, including blank or black VR scenes, interaction problems, and lighting regressions across backends.[17]

Surgical Gym did not run because it relies on a deprecated Isaac Gym integration that is no longer included in currently maintained Isaac Sim distributions. Note that NVIDIA Isaac Sim and Isaac Gym occupy different positions in NVIDIA robotics ecosystem. Isaac Sim is an Omniverse-based robotics simulation reference application for developing and testing AI-driven robots in physically based virtual environments, and it is distributed as an Omniverse application with an extension system and a bundled Python runtime. Isaac Gym, by contrast, was NVIDIA physics simulation environment oriented toward reinforcement learning research workflows; it is now explicitly labeled “Now Deprecated” and described as legacy software that is no longer supported.[18] NVIDIA recommended successor pathway is Isaac Lab, an open-source robot learning framework built on the Isaac Sim platform, and the Isaac Lab documentation states that it replaces earlier Isaac Gym frameworks and OmniIsaacGymEnvs.

In our reproduction attempt, we first had to reconcile two incompatible execution contexts: (i) Isaac Sim bundled Python environment, which is necessary for importing Omniverse modules (for example omni.isaac), and (ii) a conventional virtual environment used to install missing Python packages required by Surgical Gym scripts. Practically, we created a venv using Isaac Sim launcher Python (via python.sh -m venv ...) and then sourced Isaac Sim environment setup script so that the python from venv could resolve Omniverse modules. This step was necessary because, without the environment variables of Isaac Sim, standard Python invocations could not import omni at all, whereas after sourcing the setup, import omni.isaac succeeded. A second class of issues arose earlier in the stack, where the provided random_policy.py script failed on missing Hydra and OmegaConf dependencies; we attempted to address these via direct installation of missing packages into the venv and by rewriting the entry script to bypass Hydra-based configuration.

 doi:10.20944/preprints202606.1100.v1

After resolving these surface issues, execution consistently failed at a deeper integration boundary: Surgical Gym vectorized RL environment depends on omni.isaac.gym.vec_env (for example VecEnvBase), and importing this module raised ModuleNotFoundError: No module named 'omni.isaac.gym' even when omni.isaac itself imported correctly. This indicates that the required Isaac Sim extension or package namespace was absent from the installed Isaac Sim distribution, not merely misconfigured in Python path resolution. This observation is coherent with NVIDIA broader platform transition, since Isaac Lab is positioned as the unified robot learning framework for Isaac Sim, and Isaac Sim has also undergone extension namespace and API evolution (for example the ongoing renaming away from omni.isaac.* prefixes in newer releases). Taken together, our attempts suggest that Surgical Gym failure is not a conventional "missing pip dependency" problem, but a structural compatibility break caused by reliance on an RL interface layer that has been deprecated or removed from current Isaac Sim distributions. The net result is that Surgical Gym reproducibility is dominated by external platform compatibility rather than by conventional package installation or configuration issues.

iMSTK did not run because the build failed around the VegaFEM dependency and the project is archived, making upstream fixes unlikely. In the audit environment, iMSTK build process progressed through initial toolchain setup but failed at the dependency stage involving VegaFEM, where the build system attempted to resolve or fetch third-party components and encountered a non-recoverable error. This outcome is consistent with iMSTK design choice to automate dependency acquisition and compilation via CMake-driven external projects, which increases convenience when URLs and upstream repositories remain stable but creates a single point of failure when external artifacts move, become rate-limited, or vanish. The original version of VegaFEM has not been updated since 2018, and iMSTK installation depends on an unofficial fork maintained by another group,[19] which is an indicator of ecosystem fragmentation that can exacerbate the dependency rot for scripted download steps. The second factor, project lifecycle status, materially reduces the likelihood of resolution through upstream maintenance. Kitware GitLab page states that support and development for iMSTK has been discontinued as of May 2, 2025. Additionally, the GitHub mirror indicates that the repository was archived on September 29, 2025 and is read-only.[10] Together, these status signals imply that even if the specific Vega-related failure were diagnosable, a durable fix would most likely require local patching or community forking rather than routine upstream issue resolution. Consequently, iMSTK non-execution in this audit is best interpreted as a reproducibility failure driven by a combination of (i) transitive dependency acquisition assumptions embedded in the build and (ii) an inactive maintenance state. This combination is particularly problematic for scientific reproducibility, because it undermines the expectation that a third party can rebuild the system at a later date using documented procedures and canonical sources.

SOFA Framework did not run because it relies on a large, distribution-specific system dependency stack and legacy setup assumptions, indicating limited portability. The SOFA reproduction attempt was dominated by dependency provisioning and build-environment alignment rather than by SOFA source code defects. In the audit workflow, translating recommended installation steps to the target environment required substantial manual mapping between package managers and library variants, and the accumulation of prerequisites created repeated opportunities for version conflicts and missing packages. This profile reflects SOFA nature as a large C++ framework with optional modules and GUI-related dependencies (for example Boost and Qt), where build success depends on consistent discovery of a broad toolchain and library surface. The official documentation further supports the observation that SOFA build instructions are closely coupled to specific host distributions and contemporary toolchain baselines. The Linux build page states an explicit policy of supporting only the latest Ubuntu LTS and prescribes Ubuntu-centric package installation commands, alongside compiler and CMake requirements. Such guidance is reasonable for a project optimizing for a narrow CI target, but it reduces portability when the audit environment diverges from Ubuntu, because dependency names, versions, and default ABI choices differ across distributions. Notably, the SOFA community itself highlights the need for dependency isolation

 doi:10.20944/preprints202606.1100.v1

strategies, including using conda packages to avoid mixing system-wide libraries and to centralize dependency resolution. In the audit, the practical implication was that "run from source" reproduction required either (i) adopting the documented, distribution-specific assumptions (which were misaligned with the target environment) or (ii) introducing an additional environment manager layer (conda), which shifts complexity but does not eliminate it. The observed non-execution therefore substantiates the conclusion that SOFA reproducibility in heterogeneous environments is constrained primarily by dependency-stack portability rather than by algorithmic correctness.

Discussion

Reproducibility in open-source virtual surgery frameworks should be understood as an architectural property rather than a binary outcome. The audited systems span distinct layers of a broader software stack, including reinforcement-learning backends, mid-level simulation toolkits, general-purpose physics engines, and clinician-facing visualization interfaces. Each layer imposes different assumptions about execution context, success criteria, and interaction models. As a result, reproducibility cannot be meaningfully evaluated by treating these systems as interchangeable instances of a single category. Instead, successful reproduction must be interpreted relative to the intended role of each framework, whether that involves executing policy rollouts, compiling and running interactive simulations, or rendering clinical data in virtual reality. This perspective clarifies why identical evaluation procedures yield qualitatively different outcomes across systems.

Across all frameworks, reproducibility failures were driven primarily by the structure of their dependency ecosystems rather than by defects in core simulation logic. Platform-coupled systems such as Surgical Gym depend on external simulator infrastructures whose APIs and extension layers evolve rapidly, producing incompatibilities that cannot be resolved through conventional debugging or package installation. In contrast, toolchain-driven systems such as iMSTK fail during dependency acquisition and compilation, where assumptions about the availability and stability of third-party components no longer hold. Despite these differences in manifestation, both classes of failure share a common origin in external dependency instability. This pattern suggests that reproducibility risk is largely externalized, and that modern surgical simulation frameworks inherit fragility from the broader ecosystems on which they are built. As these ecosystems evolve, the gap between research code and sustainable execution environments continues to widen.

The case of SOFA highlights a distinct but related challenge in which generality and modularity come at the expense of portability. As a large, extensible physics engine designed to support a wide range of biomechanical simulations, SOFA requires a substantial and tightly coordinated system dependency stack. Successful installation depends on consistent alignment between compilers, libraries, and operating system distributions, which can be difficult to achieve outside the specific environments targeted by project documentation. This reflects a structural trade-off in simulation software design. Frameworks that maximize flexibility and extensibility often require complex configuration and environment management, which in turn increases the likelihood of reproducibility failure in heterogeneous settings.

These findings also motivate a re-evaluation of what constitutes open-source software in the context of surgical simulation. While all audited frameworks provide access to source code, many rely on proprietary runtimes, implicit system assumptions, or unstable external dependencies that are not captured in standard distribution mechanisms. In practice, this results in systems that are open at the code level but not self-contained at the environment level. Reproducibility therefore depends not only on code availability but also on the stability and transparency of the surrounding software stack. Addressing this gap requires approaches that treat environment specification, dependency management, and long-term maintenance as first-class concerns in the design and dissemination of virtual surgery platforms.

Our results indicate that reproducibility in virtual surgery environments is constrained by architectural diversity, dependency fragility, and design trade-offs that are not currently reflected in standard evaluation practices. Future work should prioritize reproducibility-aware development

 doi:10.20944/preprints202606.1100.v1

strategies, including clearer reporting of installation requirements, improved environment encapsulation, and closer alignment between research software and stable infrastructure. Without such efforts, the practical usability of open-source surgical simulation frameworks will remain limited despite their technical sophistication and scientific value.

Methods

Framework selection was guided by recent literature, technical reports, and community usage patterns in surgical simulation and robotic surgery research. Candidate systems were identified from survey-style analyses and project documentation spanning 2023 to 2026, with emphasis on platforms that are actively cited, publicly available, and representative of distinct architectural roles within virtual surgery environments. The final set was selected to cover a spectrum of system types in order to capture variation in intended use and dependency structure.

Reproducibility audit. Evaluation was conducted through a hands-on reproducibility audit designed to approximate the experience of an independent researcher attempting to deploy each framework. Rather than relying on synthetic benchmarks or code inspection, the methodology required complete installation and execution using publicly available documentation. For each framework, the audit proceeded from a clean environment and followed the recommended setup path, including cloning repositories, installing dependencies, and executing provided example scripts.

Operational definition of reproducibility. Reproducibility was defined relative to an intended function of each framework rather than through a uniform benchmark. For reinforcement-learning environments, success was defined as executing a minimal control loop or policy rollout within the simulator. For simulation toolkits, success required successful compilation and launch of an interactive example. For physics engines or visualization systems, success corresponded to rendering and interacting with a canonical scene. This role-dependent definition ensured that evaluation reflected the practical goals for which each system is designed.

Experimental environments. All experiments were conducted in controlled computing environments. Where required, GPU-enabled Amazon Web Services (AWS) cloud instances were used to satisfy hardware constraints imposed by simulation platforms. Particular attention was given to environment configuration, including the interaction between system-level dependencies, Python virtual environments, and simulator-specific runtimes. Simulator-bound frameworks required the use of bundled Python interpreters and environment setup scripts to enable access to internal modules, while additional packages were installed within compatible virtual environments layered on top of these runtimes. Such hybrid configurations were necessary because many frameworks span multiple execution contexts and dependency domains.

Failure mode identification. Failure modes were recorded throughout the audit and categorized according to their dominant cause. Categories included dependency resolution errors, deprecated or missing APIs, build and compilation failures, and platform incompatibilities. Classification was based on the primary blocking issue encountered during installation or execution, even when multiple contributing factors were present. Given the architectural heterogeneity of the audited systems, results were analyzed on a per-framework basis rather than aggregated into a single metric. Each framework was evaluated within its own dependency context, execution model, and definition of success.

References

1. Meara JG, Leather AJ, Hagander L, et al. Global Surgery 2030: evidence and solutions for achieving health, welfare, and economic development. *Lancet*. 2015; 386(9993):569-624. doi:10.1016/S0140-6736(15)60160-X
2. George BC, Bohnen JD, Williams RG, et al. Readiness of US General Surgery Residents for Independent Practice. *Ann Surg*. 2017; 266(4): 582-594. doi:10.1097/SLA.0000000000002414

 doi:10.20944/preprints202606.1100.v1

3. Gao R, Kurenov S, Black EW, Peters J. Adding Safety Rules to Surgeon-Authored VR Training. *arXiv*. 2021; doi:10.48550/arxiv.2111.02523
4. Khan ZA, Kamal N, Hameed A, et al. SmartSIM - a virtual reality simulator for laparoscopy training using a generic physics engine. *International Journal of Medical Robotics and Computer Assisted Surgery*. 2017; 13(3): e1771. doi:10.1002/rcs.1771
5. Popovici DM, Hamza-Lup FG, Seitan A, Bogdan CM. Comparative Study of APIs and Frameworks for Haptic Application Development. *2012 International Conference on Cyberworlds*. 2012: 37-44. doi: 10.48550/arxiv.1903.02691
6. Hamza-Lup FG, Seitan A, Petre C, Polceanu M, Bogdan CM, Popovici DM. Haptic User Interfaces and Practice-based Learning for Minimally Invasive Surgical Training. *arXiv*. 2019; doi:10.48550/arxiv.1903.04882
7. Ruthenbeck GS, Reynolds KJ. Virtual reality surgical simulator software development tools. *Journal of Simulation*. 2013; 7:101-108. doi:10.1057/jos.2012.22
8. Faure F, Duriez C, Delingette H, et al. SOFA: A Multi-Model Framework for Interactive Physical Simulation. In: Payan Y, ed. *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery Studies in Mechanobiology, Tissue Engineering and Biomaterials*. Springer Berlin Heidelberg; 2012: 283-321. vol. 11.
9. Bao C, Wang B. A Open Source Based General Framework for Virtual Surgery Simulation. *2008 International Conference on BioMedical Engineering and Informatics*. 2008; 1:575-579. doi: 10.1109/bmei.2008.129
10. iMSTK repository. https://gitlab.kitware.com/iMSTK/iMSTK
11. Munawar A, Wu JY, Fischer GS, Taylor RH, Kazanzides P. An Open Simulation Environment for Learning and Practice of Robot-Assisted Surgical Suturing. *IEEE Robotics and Automation Letters*. 2022; 7(2): 3843-3850. doi:10.1109/lra.2022.3146900
12. Moore J, Scheirich H, Jadhav S, et al. The interactive medical simulation toolkit (iMSTK): an open source platform for surgical simulation. *Frontiers in Virtual Reality*. 2023; 4:1130156. doi: 10.3389/frvir.2023.1130156
13. Poliakov V, Tsetserukou D, Poorten EV. Filasofia: A Framework for Streamlined Development of Real-Time Surgical Simulations. *arXiv* 2023; doi:10.48550/arxiv.2311.14508
14. SurRoL repository. https://github.com/med-air/surrol
15. Korzeniowski P. *Modelling and simulation of flexible instruments for minimally invasive surgical training in virtual reality*. Imperial College London; 2015. *PhD Thesis;* doi: 10.25560/58344
16. Hamza-Lup FG, Bogdan C, Seitan A. Haptic Simulator for Liver Diagnostics through Palpation. *arXiv*. 2019; doi: 10.48550/arxiv.1903.03268
17. SlicerVirtualReality repository. https://github.com/KitwareMedical/SlicerVirtualReality
18. Isaac Gym. https://developer.nvidia.com/isaac-gym
19. VegaFEM unofficial repository. https://github.com/FabienPean/VegaFEM