

MARGoS: A Configuration-Driven Platform for Reproducible MARL Experiments in ARGoS

Tom Philipp Backert¹, Mohammad D. Soorati², and Javad Ghofrani³

¹ University of Lübeck, Lübeck, Germany
tom.backert@student.uni-luebeck.de

² University of Southampton, Southampton, UK
m.soorati@soton.ac.uk

³ Bonn-Rhein-Sieg University of Applied Sciences, Sankt Augustin, Germany
javad.ghofrani@h-brs.de

Abstract. Multi-agent reinforcement learning (MARL) has become a prominent research direction, particularly within collective robotics, where the ARGoS simulator is widely used to study emergent swarm behavior. Yet despite this growth, systematic support for reuse and reproducibility in MARL-based swarm robotics remains limited and insufficiently standardized: a single experiment is assembled from manually aligned simulator files, training scripts, configuration values, paths, seeds, and environment details, so reuse and reproduction stay fragile even within one research group. To address this gap, we introduce MARGoS, a lightweight, configuration-driven platform developed in this project that elevates these experiment-defining inputs to first-class, platform-managed artifacts. MARGoS enables reproducible experimentation through configuration hashing, end-to-end seed propagation, frozen (immutable) configurations, structured logging, environment fingerprinting, and exportable experiment bundles that facilitate sharing, replay, and verification of results across machines and over time. A five-dimension evaluation (architecture, efficiency, reproducibility, usability, and collaboration) supports our claim that managing experiment variation as a platform responsibility makes ARGoS+MARL experiments easier to reuse, reproduce, and hand off. Our code is available at <https://github.com/tombackert/margos>.

Keywords: Variability · Configuration · Reproducibility · Multi-agent reinforcement learning · Swarm robotics · ARGoS

1 Project Context, Goals, and Scope

Multi-agent reinforcement learning (MARL) is a prominent paradigm for training collective robot behavior, in which large numbers of simple, embodied agents

This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this contribution is published in *Software and Systems Reuse, Product Lines, and Configuration – VARIABILITY 2026 Satellite Events* (LNCS 17104), and is available online at https://doi.org/10.1007/978-3-032-40421-3_24.

produce group-level behavior through decentralized local interaction rather than through a central controller [16, 6]. Because experimenting with physical swarms is costly and slow, this training is simulation-first, and ARGoS is a widely used modular simulator for exactly this setting [15, 1]. Running MARL training on ARGoS, which we refer to as ARGoS+MARL, is therefore a natural and increasingly common way to study learned swarm behavior. Its experiment workflow, however, is largely manual: a single run is assembled by hand from separately edited scenario files, training code, parameters, seeds, output paths, and environment assumptions, so small drifts can silently redefine the experiment and reuse or reproduction stays fragile even within one research group. We present MARGoS, a small, configuration-driven platform built in this project to make reuse and reproduction of ARGoS+MARL experiments a managed property of the platform rather than a matter of user discipline.

The recurring inputs above are variant-bearing experiment artifacts: some define a run, while others capture the state needed to rerun and compare it. When they vary independently, configuration, reproduction, and handoff become manual coordination tasks rather than managed properties. MARGoS treats these experiment-defining inputs as first-class, platform-managed artifacts through configuration hashing, end-to-end seed propagation, frozen configurations, structured logging, environment fingerprinting, and exportable experiment bundles. It thereby turns unmanaged artifact variation into platform-owned managed variation and frames ARGoS+MARL experimentation as a configuration and reuse problem rather than an algorithmic one: the contribution is the platform and its integration of variation-control responsibilities, not a new learning algorithm or swarm behavior.

The gap this addresses is deliberately bounded. Among the adjacent platforms surveyed in Section 2, none is both ARGoS-native and covers configuration $\rightarrow$ orchestration $\rightarrow$ logging $\rightarrow$ analysis $\rightarrow$ export and import as a single pipeline. MARGoS explores integrated experiment-lifecycle support for this stack by centralizing the variant-bearing artifacts, and it was completed as a small-scale university research project sized for meaningful evaluation rather than as a general-purpose product: enough infrastructure to measure workflow efficiency, self-reproducibility, analytical usability, and handoff. Section 2 positions the platform against those adjacent systems, Section 3 defines the evaluation it enables, and Section 4 reports the lessons it produced.

To show that this problem is real and that MARGoS addresses it correctly, we evaluate the platform along the five dimensions summarized in Table 1, which states each guiding question and the evidence type. RQ1 is enablement-only, validated by whether the platform executes the RQ2-RQ5 evaluation protocols, while RQ2-RQ5 are the four empirically tested dimensions that recur in the evaluation design (Section 3) and the consolidated results (Section 4). Throughout, the evaluation is supporting evidence for the platform contribution, not the contribution itself.

The scope is deliberately narrow: collaboration means asynchronous bundle handoff between prepared environments, not cloud workspaces, shared state, or

Table 1. Project goals and evaluation dimensions. Each dimension maps to a guiding question and to the evidence used to answer it in this paper; RQ1 is enablement-validated through protocol execution, RQ2-RQ5 are threshold-tested.

Dimension	Question	Evidence used in this paper
RQ1 Architecture (enablement)	Can the platform execute the evaluation protocols?	Protocol enablement across RQ2-RQ5
RQ2 Efficiency	Does integration reduce repeated workflow effort?	Time and step reduction vs manual workflow
RQ3 Reproducibility	Can identical experiments be rerun reliably on the same hardware?	Reference run plus 20 automated reproductions
RQ4 Usability	Does the design reduce interaction complexity?	Heuristic audit and KLM analysis
RQ5 Collaboration	Can experiment bundles be handed off reliably to a prepared collaborator?	Export/import handoff trials

live co-editing. The project also excludes algorithmic contributions, unrestricted cross-lab portability, and new experiments for this paper, reusing the executed project evaluation only. Each exclusion keeps the evidence tied to managed variation rather than product breadth. The next section turns the variant-bearing artifacts introduced here into platform-owned managed variation.

2 Platform Overview

MARGoS is structured as a five-stage pipeline: configuration $\rightarrow$ orchestration $\rightarrow$ logging $\rightarrow$ analysis $\rightarrow$ export/import. Each stage centralizes one or more of the experiment-defining artifacts named in Section 1. The pipeline is the concrete mechanism by which those unmanaged inputs become platform-managed variation: configurations define experiment state, orchestration executes it, logs and reports preserve observed behavior, and export/import turns a run into a handoff bundle.

To position this design, we compare it against representative recent platforms that combine multi-agent or multi-robot learning infrastructure with simulator integration. The comparison uses three criteria tied to the gap claim: whether the system is ARGoS-native, whether it covers the full configuration $\rightarrow$ orchestration $\rightarrow$ logging $\rightarrow$ analysis $\rightarrow$ export/import lifecycle, and whether export/import is part of the reproducibility or handoff path rather than a separate manual step. Five comparators cover adjacent multi-agent or multi-robot slices: distributed many-agent infrastructure on Alchemist [5], JAX-vectorized learning on the Robotarium testbed [7], IsaacSim-based MARL self-play [8], Gazebo/ROS autonomous-driving MARL [9], and AirSim/UE4 UAV swarms [10]. ORBIT [12] and IsaacLab [11] cover integrated single-robot

learning. Against the three criteria, the surveyed systems provide useful related-work context but do not cover the complete ARGoS+MARL lifecycle evaluated here; the gap claim opened in Section 1 is therefore bounded to this comparator set rather than to the literature as a whole. ORBIT and IsaacLab are used only as adjacent feasibility evidence for integrated robot-learning infrastructure, while the MARGoS pipeline is derived from the research questions and evaluation metrics for ARGoS+MARL. The contribution claimed here is the RQ-driven integration of variation-control responsibilities for this stack.

The CLI is the user-facing entry point to the pipeline and exposes it as five commands: **run**, **show**, **compare**, **export**, and **import**. These commands are entry points into the pipeline rather than one-to-one stage labels: **run** spans configuration loading, seed propagation, run-state capture, orchestration, and per-iteration logging; **compare** operates over stored logs and configuration hashes; and **show**, **export**, and **import** operate on stored runs. At the orchestration boundary, ArgosToZoo provides the ARGoS-to-MARL bridge: ARGoS communicates through ZeroMQ with a PettingZoo-compatible Python environment that RLlib training scripts can consume. There is no separate **reproduce** command: where later metrics label a reproduction task, the platform path is **import** followed by **run**, with **compare** used to verify the rerun against the reference. All five commands are invocable with a single positional argument under a convention-over-configuration default, so the surface a user must traverse to move an experiment through the pipeline is fixed at a five-verb interface. This five-command surface is itself a design decision, and it is the user-facing measurement surface that the efficiency, usability, and collaboration evaluations in Section 4 measure against.

Configuration hashing, seed propagation, frozen configs, structured logs, environment fingerprints, and exported source copies are the core reusable assets through which the platform turns experiment variants into managed artifacts. The hash is computed over the normalized frozen configuration. Because that configuration contains file references rather than embedded file bodies, the config hash alone is not a source-file versioning mechanism. Local rerun identity therefore assumes unchanged referenced sources; the platform makes that boundary explicit rather than claiming content-level version control. Drift in referenced simulator files and training scripts is controlled at the bundle boundary: the completed run records the frozen config, integrity hash, environment fingerprint, and logs, while the exported bundle copies the scenario and training-script sources and import rewrites references to those bundled copies. A received bundle therefore reproduces the captured variant rather than resolving mutable paths in the sender’s working tree.

The asset-to-outcome mapping is deliberately narrow. Configuration hashing, seed propagation, and frozen configs support RQ3 same-hardware rerun identity; logs and summaries support RQ2 workflow-effort measurement and result inspection; the RQ4 analytical-usability claim comes primarily from the five-command surface and measured task paths rather than from logs alone; and export/import bundles assembled from frozen configs, logs, source copies, and

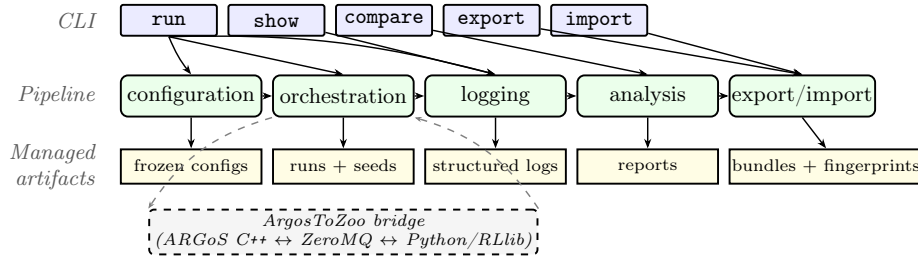


Fig. 1. MARGoS architecture. Control flow runs top-down from the five-command CLI through the five-stage pipeline (configuration → orchestration → logging → analysis → export/import); managed-artifact flow runs from each stage into the bottom row (frozen configs, runs and seeds, structured logs, reports, exportable bundles with environment fingerprints). The `run` command intentionally fans out because it spans configuration, orchestration, and logging rather than naming one stage. The ArgosTo-Zoo bridge couples the orchestration stage to ARGoS via ZeroMQ.

fingerprints support the RQ5 handoff evaluation. Figure 1 summarizes this control and managed-artifact flow across the five pipeline stages, and Section 3 picks up the asset-to-outcome mapping as the input to the evaluation design.

3 Evaluation Design

Section 2 mapped each platform-owned asset to an outcome it enables; this section defines how those outcomes are measured. Evaluation was predefined, metric-driven, and based on counted, timed, or analytical measures: metrics, thresholds, and sample sizes were frozen before data collection, and no new experiments were added for this paper. RQ2, RQ4, and RQ5 compare documented manual workflows against MARGoS paths; RQ3 reruns one same-hardware reference run. In the timed protocols, trial order was documented and conditions were interleaved where the protocol allowed, so learning effects could be checked rather than inferred away. Surveys, interviews, and participant self-report ratings were excluded; RQ4 uses Nielsen-style heuristic evaluation and GOMS/KLM modelling [14, 13, 2, 3], as expert analytical evidence rather than behavioral user evidence. Primary interpretation is threshold/effect-size based; Mann–Whitney and bootstrap checks are reported only as robustness checks for RQ2/RQ5 timing, not as population-level inference.

Table 2 consolidates the four empirical dimensions. It also fixes the operational readings needed to assess fairness: RQ2 uses screen-recorded human-effort time and discrete workflow steps; RQ3 requires reward and AUC within $\pm 1\%$ with matching configuration identity; RQ4 combines 35 heuristic criteria with weighted KLM task paths; and RQ5 includes step counts and timing, with handoff success defined as final-50-reward match within $\pm 1\%$. RQ1 Architecture is answered separately by whether the architecture enables the RQ2-RQ5 protocols to run end-to-end.

Table 2. Executed evaluation design: primary metrics, sample sizes, and predefined practical targets across the four empirically tested dimensions. RQ1 Architecture is omitted because it is validated by protocol enablement: the platform must execute the RQ2-RQ5 protocols end-to-end rather than pass a separate threshold test.

Dimension	Primary metrics	Sample	Target
RQ2 Efficiency	Time-to-Complete; Steps-to-Complete	N=5 per condition	$\geq 50\%$ reduction
RQ3 Reproducibility	Reproduce-Success-Rate; Result-Variance	N=20 automated runs	$\geq 90\%$
RQ4 Usability	Heuristic compliance; KLM reduction	35 criteria; 7 tasks; 1 structured audit	$\geq 80\%$; $\geq 50\%$
RQ5 Collaboration	Steps/time-to-share; first-run/reproduce time; handoff success	10 manual + 10 platform cycles; timed N=10/condition	$\geq 50\%$; $\geq 90\%$ success

Four measurement boundaries scope the Section 4 results. RQ3 is same-hardware only. RQ5 is local-first and asynchronous, but the measured proxy is narrower: a prepared local collaborator account with isolated home directory, fixed repository copies, one shared preinstalled RQ5 virtualenv, and transfer-only artifact access; network and machine provisioning are excluded. Human-effort timing excludes compute wait. Behavioral usability metrics (Time-to-First-Success, Error-Rate, Recovery-Time, Documentation-Lookups) were dropped because the self-as-user setup invalidates behavioral interpretation, so RQ4 reports only analytical methods. The next section reports the measured outcomes against these targets and boundaries as the evidence bundle that anchors the paper’s contribution.

4 Results and Lessons Learned

Table 3 summarizes the empirical outcomes: RQ2-RQ4 meet their predefined practical targets, while RQ5 is partially supported because two timing targets fall short despite reliable handoff. Fig. 2 places the cross-RQ effort-reduction metrics on a single axis, with the dashed line marking the predefined $\geq 50\%$ target. The four lessons below unpack what each judgment means at the project level; together they form the evidence bundle that anchors the configuration-driven contribution introduced in Section 1.

Table 3. Measured outcomes across the four empirically tested dimensions. RQ2-RQ4 meet their predefined practical targets, while RQ5 is partially supported; jointly the four rows are the evidence bundle that platform-owned managed variation translates into measurable workflow effort reduction, deterministic self-reproducibility, analytical usability gains, and verified handoff at a first-run overhead.

Dimension	Metric result	Threshold	Judgment
RQ2 Efficiency	Time -78.2%; steps -66.7%	$\geq 50\%$ reduction	Supported
RQ3 Reproducibility	100% success (20/20); 0.0000 SD	$\geq 90\%$ success	Strongly supported
RQ4 Usability	85.7% heuristic compliance; 56.9% KLM reduction	$\geq 80\%$; $\geq 50\%$	Supported
RQ5 Collaboration	100% handoff success; share time -65.3%; reproduce time -45.9%; first run +7.4% slower	$\geq 50\%$ reductions; $\geq 90\%$ success	Partially supported

Time-to-Complete on the RQ2 reference task dropped from 87.0 ± 15.02 s in the manual condition to 19.0 ± 1.41 s in the platform condition, a 78.2% reduction, and Steps-to-Complete dropped from 6.0 to 2.0, a 66.7% reduction; both exceeded the predefined $\geq 50\%$ threshold (robustness: $U=0$, exact $p=0.011$, bootstrap relative-reduction CI [74.4, 80.9]%). The gains concentrated in operational coordination, including training initiation, monitoring, result extraction, and export, rather than in the underlying robotics or learning algorithms, because the platform replaces a multi-tool, multi-step manual sequence with single-command operations. The project lesson is that integration gives the clearest workflow benefit when it removes repeated manual coordination across tools, not when it adds features. This is a measured reduction on the RQ2 reference task with $N=5$ per condition and a single expert evaluator, not a generalized productivity claim.

All 20 RQ3 reruns matched the reference on final reward and area under the learning curve (100% success, 0.0000 SD), and the Config-Integrity and Seed-Determinism checks passed; this exceeds the predefined $\geq 90\%$ threshold. This is the clearest single-dimension result in the bundle. Within the same-hardware boundary stated in Section 3, the platform’s centralized configuration management and deterministic seed propagation produced perfectly reproducible training outcomes. The project lesson is that bounded self-reproducibility follows directly from making configuration identity, seed propagation, and runtime configuration-integrity checks platform responsibilities rather than relying on user discipline to manually track them; structured logs support auditability of these checks but are not the directly measured mechanism here. Within the evidence bundle this is the rerun-identity strand: managed configuration and seeds preserve rerun identity, while the coordination strand (RQ2) and the handoff strand (RQ5) carry their own outcomes below.

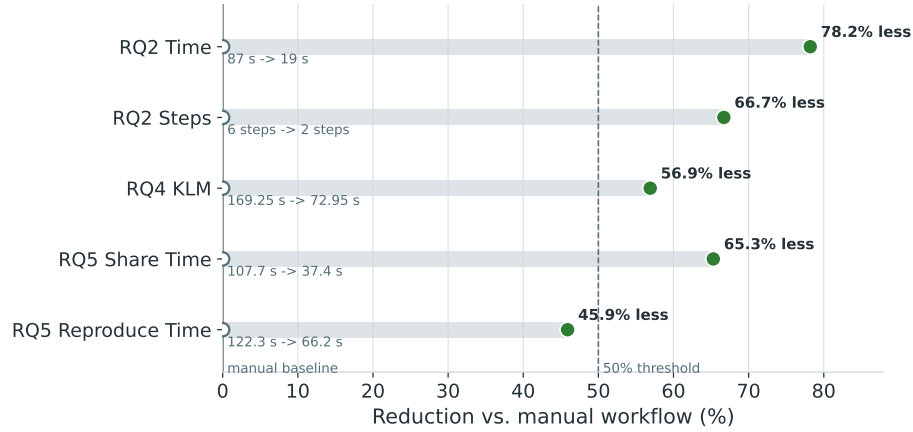


Fig. 2. Cross-dimension effort reductions on the executed project evaluation: every metric is plotted against the manual baseline at 0%, and the predefined $\geq 50\%$ target appears as a dashed line. The chart is a visual comparison, not pooled inference: it juxtaposes workflow-effort gains on the RQ2 reference task, analytical KLM reductions on the RQ4 task set, and sharing-phase gains on RQ5, while showing the RQ5 reproduce-time metric narrowly below the 50% target.

Heuristic-Compliance-Rate was 30/35 (85.7%) against a $\geq 80\%$ threshold, and KLM-Reduction was 56.9% weighted across seven tasks against a $\geq 50\%$ threshold; both targets were met. The KLM reduction is not uniform across the task set: gains concentrated in execution, monitoring, results access, export, and import/run reproduction tasks (T3-T7, 58-88% reductions), while configuration editing tasks (T1-T2) showed 0% reduction because both conditions use the same editor for the same parameters. The project lesson is that analytical usability improves where the platform owns the task path; it does not improve where the user still hand-edits configuration files. The 0% reduction on T1-T2 is a genuine design boundary rather than a measurement artifact, and it identifies the most actionable area for future platform work.

Handoff-Success-Rate was 100% in both conditions (10/10 platform handoffs, 10/10 manual baseline); Steps-to-Share dropped from 8 to 1 (-87.5%); Time-to-Share dropped from 107.7 ± 24.1 s to 37.4 ± 18.2 s (-65.3% ; robustness: $U=0$, exact $p<0.001$, bootstrap relative-reduction CI $[53.8, 75.7]\%$); Time-to-Reproduce dropped from 122.3 ± 30.3 s to 66.2 ± 13.1 s (-45.9%), narrowly below the predefined 50% threshold; and Time-to-First-Run was 7.4% slower in the platform condition (52.6 ± 11.4 s vs 56.5 ± 11.6 s). Because both conditions reached 100% success, RQ5 differentiates effort and overhead rather than a reliability gain; Bundle-Completeness was 8/8, and Setup-Divergence averaged 3.0 package mismatches that did not prevent reproduction. Figure 3 visualizes this tradeoff: large sharing-phase gains and reliable success, set against the first-run overhead. The recorded handoff timings identify the recurring over-

head in the platform’s import processing: bundle extraction, configuration path rewriting, and environment fingerprint comparison. The project lesson is that verified handoff preserves 100% handoff success while reducing sharing effort, at a first-run overhead from bundle validation and import checks; RQ5 is partially supported, and this tradeoff is the project’s most actionable finding for managed-variant infrastructure because it surfaces a real cost of that infrastructure. This leads directly to the validity and transferability boundaries in Section 5.

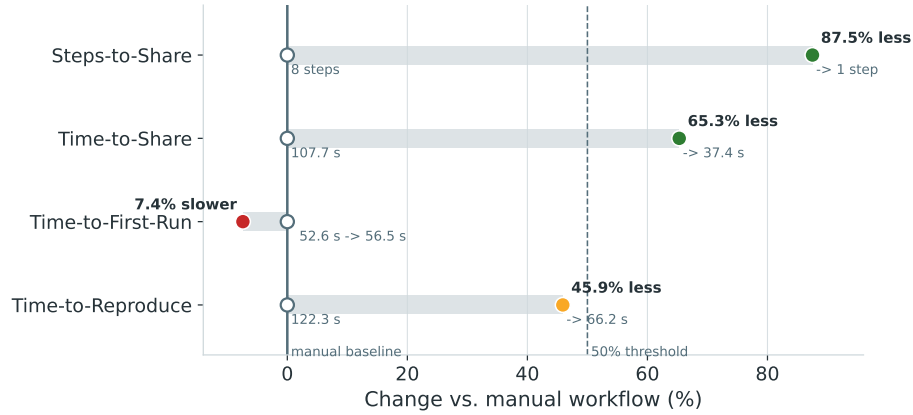


Fig. 3. Collaboration tradeoff (RQ5): the four timing metrics relative to the manual baseline at 0%. Large sharing-phase gains and reliable handoff success are set against the first-run overhead introduced by bundle validation and import checks. The figure shows the RQ5 partial verdict as a single tradeoff in the evidence bundle rather than as a missed target in isolation.

5 Limitations and Transferability

The four-validity-type framework of internal, construct, external, and conclusion validity is the standard taxonomy for validity threats in empirical software-engineering studies and organizes the boundaries under which the Section 4 results hold [4, 17]. The core internal-validity limit is single developer-evaluator, self-evaluation: instrumented metrics reduce evaluator discretion for RQ2, RQ3, and RQ5, while RQ4 remains judgment-heavy and is bounded by documented heuristic-audit and KLM decision rules (Section 3). The core construct-validity limit is that KLM predicts expert error-free interaction time and a heuristic audit is not novice-user behavior, so the RQ4 claim is a design-property claim, not evidence of novice behavior, real-user error patterns, or real-user time savings. The core external-validity limits are the measurement boundaries stated in Section 3: same-hardware self-reproducibility, a prepared collaborator account

despite measured residual library drift, and the ARGoS/RLlib stack and executed task set as the only evaluated context. The conclusion-validity limit is that results are threshold and effect-size judgments under the executed study design, not population-level inference; predefined metrics and thresholds reduce post-hoc selection risk, documented trial order addresses learning effects in the timed protocols, and RQ5’s import-overhead isolation addresses the main remaining handoff confound.

Within those boundaries, treating explicit configuration, deterministic seed propagation, structured logs, and exportable bundles as first-class managed-variation mechanisms is the reusable design decomposition this project advances; it is a candidate pattern for simulator-learning stacks in which those primitives can be exposed, not an empirically validated transfer claim. Direct portability is not claimed without further evidence: transferability here is a pattern claim, not a deployment claim.

6 Conclusion

MARGoS, a configuration-driven platform for ARGoS+MARL experimentation, turns fragmented experiment-defining artifacts into an executable, reusable pipeline. Five research questions evaluate it: the platform answers RQ1 by running the RQ2-RQ5 protocols end-to-end. The measured pattern is sharp but bounded: integration cuts RQ2 workflow time by 78.2%, RQ3 reruns reach 100% success on the same hardware with no observed result variance, RQ4 meets its analytical usability targets, and RQ5 preserves 100% handoff success with 65.3% less sharing time while still missing the reproduce-time target and adding a 7.4% first-run overhead. The resulting next steps are concrete: reduce import-validation overhead and test pattern-level transfer toward cross-lab portability in simulator-learning stacks that expose the same primitives.

References

1. Blais, M.A., Akhloufi, M.A.: Reinforcement learning for swarm robotics: An overview of applications, algorithms and simulators. *Cognitive Robotics* **3**, 226–256 (Jan 2023). <https://doi.org/10.1016/j.cogr.2023.07.004>
2. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. *Communications of the ACM* **23**(7), 396–410 (Jul 1980). <https://doi.org/10.1145/358886.358895>
3. Card, S.K., Moran, T.P., Newell, A.: *The Psychology of Human-Computer Interaction*. Erlbaum, Mahwah, NJ, repr edn. (1983)
4. Cook, T.D., Campbell, D.T.: *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin, Boston (1979)
5. Domini, D., Aguzzi, G., Pianini, D., Viroli, M.: A Reusable Simulation Pipeline for Many-Agent Reinforcement Learning. In: *2024 28th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. pp. 83–90. IEEE, Urbino, Italy (Oct 2024). <https://doi.org/10.1109/DS-RT62209.2024.00021>

6. Gronauer, S., Diepold, K.: Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review* **55**(2), 895–943 (Feb 2022).
<https://doi.org/10.1007/s10462-021-09996-w>
7. Jain, S.A., Liu, J., Kailas, S., Ravichandar, H.: JaxRobotarium: Training and Deploying Multi-Robot Policies in 10 Minutes (Nov 2025).
<https://doi.org/10.48550/arXiv.2505.06771>
8. Ke, Y., Li, S., Li, Z., Li, H., Liu, Y., He, Y.: ArenaSim: A High-Performance Simulation Platform for Multi-Robot Self-Play Learning. *IEEE Robotics and Automation Letters* **10**(6), 5497–5504 (Jun 2025).
<https://doi.org/10.1109/LRA.2025.3557224>
9. Liang, Z., Cao, J., Jiang, S., Saxena, D., Cao, R., Xu, H.: From Agents to Robots: A Training and Evaluation Platform for Multi-robot Reinforcement Learning. In: 2024 IEEE 30th International Conference on Parallel and Distributed Systems (ICPADS). pp. 593–600. IEEE, Belgrade, Serbia (Oct 2024).
<https://doi.org/10.1109/ICPADS63350.2024.00083>
10. Lou, J., Wu, W., Liao, S., Shi, R.: Air-M: A Visual Reality Many-Agent Reinforcement Learning Platform for Large-Scale Aerial Unmanned System. In: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 5598–5605. IEEE, Detroit, MI, USA (Oct 2023).
<https://doi.org/10.1109/IROS55552.2023.10341405>
11. Mittal, M., Roth, P., Tigue, J., Richard, A., Zhang, O., Du, P., Serrano-Munoz, A., Yao, X., Zurbrugg, R., Rudin, N., et al.: Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning (Nov 2025).
<https://doi.org/10.48550/arXiv.2511.04831>
12. Mittal, M., Yu, C., Yu, Q., Liu, J., Rudin, N., Hoeller, D., Yuan, J.L., Singh, R., Guo, Y., Mazhar, H., Mandlekar, A., Babich, B., State, G., Hutter, M., Garg, A.: Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments. *IEEE Robotics and Automation Letters* **8**(6), 3740–3747 (Jun 2023). <https://doi.org/10.1109/LRA.2023.3270034>
13. Molich, R., Nielsen, J.: Improving a human-computer dialogue. *Communications of the ACM* **33**(3), 338–348 (Mar 1990). <https://doi.org/10.1145/77481.77486>
14. Nielsen, J., Molich, R.: Heuristic evaluation of user interfaces. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Empowering People - CHI '90*. pp. 249–256. ACM Press, Seattle, Washington, United States (1990). <https://doi.org/10.1145/97243.97281>
15. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence* **6**(4), 271–295 (Dec 2012).
<https://doi.org/10.1007/s11721-012-0072-5>
16. Şahin, E.: Swarm Robotics: From Sources of Inspiration to Domains of Application. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Şahin, E., Spears, W.M. (eds.) *Swarm Robotics*, vol. 3342, pp. 10–20. Springer Berlin Heidelberg, Berlin, Heidelberg (2005).
https://doi.org/10.1007/978-3-540-30552-1_2
17. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg (2012).
<https://doi.org/10.1007/978-3-642-29044-2>