RoboSimStudio: An Interactive Toolkit for Rapid Real2Sim of Robot Manipulation Tasks

Kaiyuan Chen[2*], **Shuangyu Xie**[2*], **Uksang Yoo**[2*],
Xinyue Zhu[2], Cael Magner[2], Alvin Zhu[2], Daehwa Kim[2],
Eric Heiden[1], Erwin Coumans[1], Miles Macklin[1], Spencer Huang[1], Ken Goldberg[2]
[*]Equal contribution
[1]NVIDIA [2]University of California, Berkeley
Project Website: https://robosimstudio.github.io

Abstract: It has been difficult to accurately model the physics of the interaction between a robot and the objects it manipulates, including contact mechanics, coupled contact dynamics, deformation, stiffness, and friction. We present RoboSimStudio (RSS), an interactive toolkit that makes choices of solver, contact model, and physical parameters explicit and editable, allowing both humans and coding agents to quickly create and iteratively refine simulations based on observed physical behavior. We implement RSS with the open Newton physics simulation system and describe experiments with human-guided Real2Sim using (1) visual feedback and (2) force feedback, and (3) agentic Real2Sim2Real. RSS will be open-sourced; code, videos, and additional information are available on the project website: https://robosimstudio.github.io.

1 Introduction

Simulating robot manipulation requires modeling the interaction between a robot and the objects it manipulates, including robot kinematics, coupled contact dynamics, and deformation (Figure 1). Emerging simulation frameworks, such as Newton [1], Genesis [2], and SuperDex [3], increasingly cover these different forms of physics by supporting rigid, articulated, and deformable-body simulation within a common framework. However, accurately modeling robot manipulation behavior to match real-world physics, or Real2Sim, still requires choosing appropriate physics models, contact models, and physical parameters, and iteratively refining these choices against observed behavior. We observe that humans and LLM coding agents share a fundamental interaction pattern for Real2Sim: they modify a scene, execute physical interactions, observe discrepancies, and refine the model. Humans perform this loop through direct manipulation and immediate visual or force feedback, while coding agents [4, 5] use iterative reasoning and acting [6] to inspect simulator state and revise the scene and robot behavior. Motivated by this, we make multi-physics simulation directly interactive and editable by both humans and coding agents.

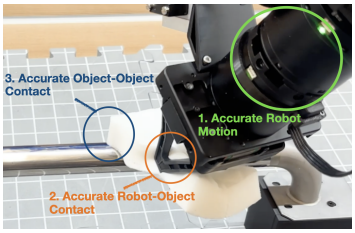


Figure 1: Challenges of simulating accurate robot control under object-object contacts and object-robot contacts in contact-rich and deformable manipulation.

We present RoboSimStudio (RSS), an interactive toolkit for physics-based simulation of robot manipulation built on Newton [1] and Viser [7]. RoboSimStudio supports co-simulation with external physics engines and makes the solver, contact model, and physical parameters explicit and editable. Humans tune them through an interactive display with sliders, visual feedback, and force feedback,

CoRL 2026 Workshop on Physics Simulation and World Models (PhysWM).

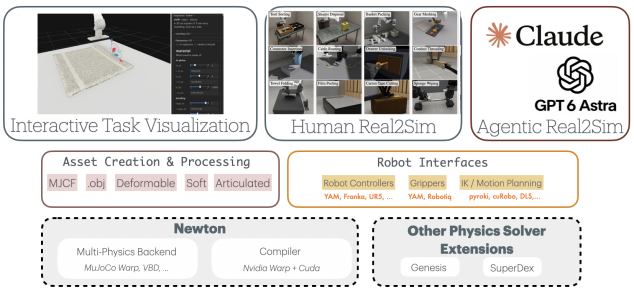


Figure 2: **RoboSimStudio Architecture.** Newton and co-simulated external engines provide the physics; RoboSimStudio composes robots, assets, and solvers into one editable scene, which humans access through the interactive studio and coding agents through an iterative interface.

while coding agents receive simulator state together with spatial-temporal text and visual feedback to observe the consequences of their changes.

2 RoboSimStudio

Figure 2 shows the architecture of RoboSimStudio. Physics is provided by Newton [1] and its solvers, with external engines co-simulated by exchanging contact and forces. RoboSimStudio composes robots, objects, and physics into one simulation whose solver, contact model, and physical parameters stay explicit and editable, and exposes it through code, an interactive studio for humans, and an iterative interface for coding agents.

Robot Interfaces. RoboSimStudio provides a common, extensible interface for Franka Panda, UR5e, and single- and bimanual YAM robots, each specified declaratively by its model, joints, mounting, and hand geometry. It supports interchangeable inverse kinematics solvers such as pyroki [8], joint-space, Cartesian, and guarded contact motions, collision-aware planning such as cuRobo [9], and joint or end-effector action spaces.

Multi-Physics Simulation and Coupling. No single physical model suffices for scenes that combine articulated robots, rigid objects, and deformable materials. In RoboSimStudio, objects specify what they are, while the scene specifies which solver models each type of physics. By default, articulated robots use MuJoCo-Warp [10], cables and cloth use Vertex Block Descent [11], and soft bodies can be simulated by external engines [3, 2] co-simulated with Newton. RoboSimStudio couples these solvers by exchanging contact forces between Newton solvers or poses and contact wrenches with external engines. The solver, coupling, contact model, and material parameters remain explicit scene entries that can be changed without rewriting the task.

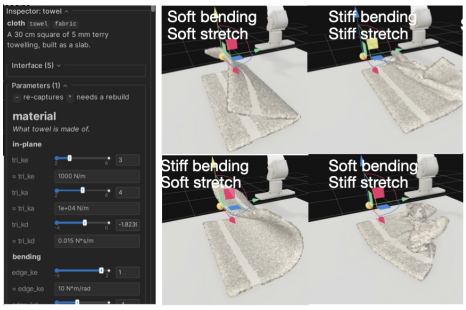


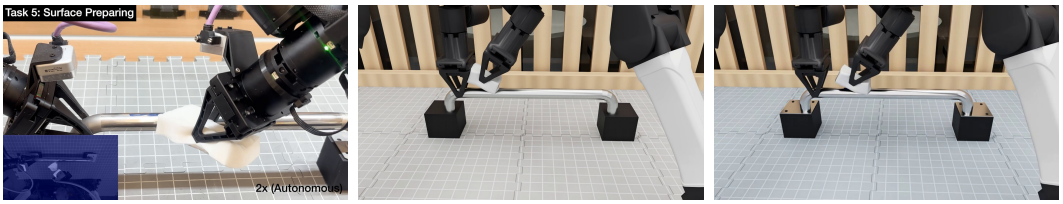
Figure 3: **RoboSimStudio studio.** Left: a towel's inspector; tri_ke and edge_ke set stretch and bending stiffness. Right: the same dragged fold at the same instant under four stiffness settings.

Interactive Interface for Humans. In the browser-based studio built on Viser [7], users can select, drag, and inspect objects and control robot arms directly, probing physical behavior before writing a policy. Each object's inspector exposes its physical parameters with units and ranges; parameters such as stiffness and damping can be changed while the simulation runs, and those that require a rebuild are marked. Figure 3 shows the same towel fold under four stiffness choices: tuning becomes direct interaction with the resulting behavior.

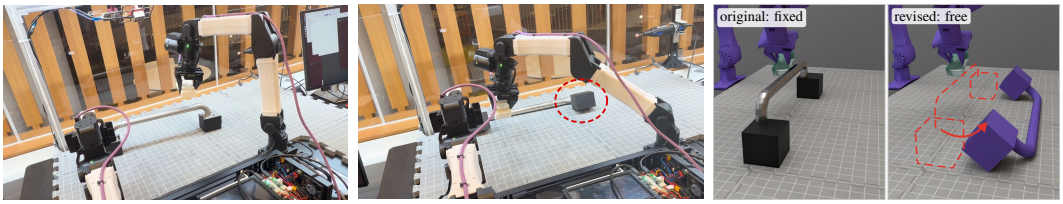
Iterative Interface for Agents. Given a video of a real task, a coding agent tunes a simulation through the same loop: modify the scene, execute a behavior, and observe the result. RoboSimStudio organizes a rollout into robot motion primitives and converts simulator state into spatial-temporal



Figure 4: **18 challenging contact-rich and deformable manipulation task simulations created with RoboSimStudio.**



(a) Real task (phone video). (b) Simulation by Claude Fable 5.1. (c) Simulation by GPT-6 Astra.



(d) Sim2Real policy deployment. (e) Real failure. (f) Agentic simulation refinement.

Figure 5: **Agentic Real2Sim2Real for sponge wiping.** (a) One arm holds a bar while the other wipes a marker stripe with a sponge. (b, c) Simulations built from the same phone video by two coding agents, with a deformable sponge ((c) path traced). (d) The policy improved in simulation, on the real robot. (e) The wipe displaces a bar support that is fixed in simulation. (f) The agent frees the bar; the revised simulation captures it tipping. Appendix Figure 15 shows the complete pipeline.

feedback that describes target error, object motion, contact events, grasping or slipping, and progress toward the task goal. This feedback exposes events such as contact that are hard to infer from images, letting an agent attribute an outcome to its cause and revise the policy or the simulation until its behavior matches the video.

3 Case Studies

3.1 Human Real2Sim with Visual Feedback

From observations of a real task, a user selects robot and object assets, arranges their geometry, and chooses the physics models; the user then executes the task, compares the simulated motion and deformation with the real behavior, and adjusts contact and material parameters until the two qualitatively agree. With this workflow we created the 18 simulations in Figure 4, spanning rigid bodies, articulated mechanisms, cables, cloth, and soft bodies. Three of them pose different modeling challenges:

- *Connector insertion*: sub-millimeter clearance between a USB-C plug and its receptacle, handled by mesh-based contact with adjustable collision resolution.

- *Leaf pressing*: bending and shape recovery of a thin curved leaf, handled by shell-based simulation with adjustable solver and contact parameters.
- *Strip polishing*: material removal driven by contact force and relative motion, modeled with Preston's equation [12] from solver contact forces.

3.2 Human Real2Sim with Force Feedback

Contact is hard to judge visually: when inserting a tool, it helps to feel contact and friction. In RoboSimStudio, a low-cost Gello leader arm [13] drives a simulated robot, and simulated contact is rendered back to the operator through the leader's motors. The leader's joint angles command the simulated arm, and the gap between the commanded and simulated configurations is fed back when contact prevents the simulated robot from following. To avoid rendering simulation lag as contact, RoboSimStudio uses a speed-dependent deadband and releases feedback when the simulation pauses or falls behind; the gripper uses its measured, not commanded, opening, so that grasping is distinguished from closing in free space. Because the feedback comes from this configuration gap rather than from raw contact forces relayed between engines, it reflects the net effect of whichever coupled solvers resolve the contact, and the deadband suppresses small transients. The operator thus feels the behavior of the chosen solver and parameters, which serves both demonstration collection and evaluation of the contact model, for example when inserting a thin tool into a hanger (Figure 6).

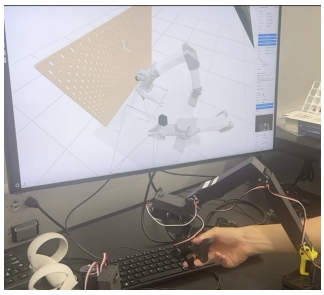


Figure 6: Tool-hanging teleoperation with RoboSimStudio.

3.3 Agentic Real2Sim2Real

We study agentic Real2Sim2Real on bimanual sponge wiping, which combines articulated, rigid-body, and soft-body physics (Fig. 5a). Given a smartphone video without CAD models or calibrated cameras, the RoboSimStudio documentation, and a one-sentence task description, two off-the-shelf coding agents, Claude Fable 5.1 in Claude Code [4] and GPT-6 Astra in Codex [5], each constructed a multi-physics simulation in about an hour (Fig. 5b, 5c); one agent constructed and verified the simulated task within 56 minutes. The agents estimated scene geometry from the video and used visual observations and spatial-temporal feedback (Section 2) to find errors in robot kinematics and contact conditions. We then varied the sponge's stiffness and generated 30 spatial layouts of the bar and sponge, over which Graph-as-Policy (GaP) [14] improved the simulated wiped area from 0% to 100% in 14 iterations. The resulting policy transferred to the physical robot and completed the wiping task (Fig. 5d). One real rollout, however, failed when the wipe displaced a bar support that was fixed in simulation (Fig. 5e). Given the failure video and a description of the cause, the coding agent replaced the fixed constraint with a free rigid body; under an off-center wipe, the revised simulation reproduces the bar tipping, which the original cannot (Fig. 5f).

4 Conclusion

RoboSimStudio makes the solver, contact model, and physical parameters of a manipulation simulation explicit and editable for humans and coding agents. It does not eliminate the Sim2Real gap, but because each constraint is an explicit modeling choice, a real failure such as the displaced bar support directly identifies what to change next. **Limitations.** This paper reports a developing system and a pilot study; Appendices A–H give related work and further details. Real–sim agreement was assessed mainly visually, with a first event-timing comparison (Appendix G.1); future work will compare object trajectories and contact forces against real measurements. The stability of coupling forces between engines with different contact models is not yet characterized beyond the documented limits of Appendix C.1. Real-world transfer was evaluated on one task, and a second task in simulation only (Appendix H); we plan repeated real-world trials across more tasks.

References

[1] The Newton Contributors. Newton: GPU-accelerated physics simulation for robotics and simulation research, Apr. 2025. URL https://github.com/newton-physics/newton. Open-source software (Linux Foundation project initiated by Disney Research, Google DeepMind, and NVIDIA).

[2] Genesis Authors. Genesis: A generative and universal physics engine for robotics and beyond, Dec. 2024. URL https://github.com/Genesis-Embodied-AI/genesis-world.

[3] The Mochi Team. An implicit physics engine for contact-rich simulation, 2026. URL https://github.com/facebookresearch/project_superdex. Project SuperDex.

[4] Anthropic. Claude Code, 2025. URL https://claude.com/claude-code. Agentic coding tool; introduced February 2025. Accessed 2026-10-04.

[5] OpenAI. Codex, 2025. URL https://openai.com/index/introducing-codex/. Coding agent; introduced May 2025. Accessed 2026-10-04.

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[7] B. Yi, C. M. Kim, J. Kerr, G. Wu, R. Feng, A. Zhang, J. Kulhanek, H. Choi, Y. Ma, M. Tancik, and A. Kanazawa. Viser: Imperative, web-based 3D visualization for Python. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, July 2026. doi:10.15607/RSS.2026.XXII.175.

[8] C. M. Kim, B. Yi, H. Choi, Y. Ma, K. Goldberg, and A. Kanazawa. PyRoki: A modular toolkit for robot kinematic optimization. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1312–1319. IEEE, Oct. 2025. doi:10.1109/IROS60139.2025.11246651.

[9] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox. cuRobo: Parallelized collision-free minimum-jerk robot motion generation. arXiv preprint arXiv:2310.17274, 2023. URL https://arxiv.org/abs/2310.17274. Technical report.

[10] Google DeepMind and NVIDIA. MuJoCo Warp (MJWarp): GPU-optimized version of the MuJoCo physics simulator, designed for NVIDIA hardware, 2025. URL https://github.com/google-deepmind/mujoco_warp.

[11] A. H. Chen, Z. Liu, Y. Yang, and C. Yuksel. Vertex Block Descent. *ACM Transactions on Graphics*, 43(4):1–16, July 2024. doi:10.1145/3658179.

[12] F. W. Preston. The theory and design of plate glass polishing machines. *Journal of the Society of Glass Technology*, 11:214–256, 1927.

[13] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel. GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12156–12163. IEEE, Oct. 2024. doi:10.1109/IROS58592.2024.10801581.

[14] K. Chen, S. Xie, L. Fu, J. Yu, W. Pacini, et al. GaP: A Graph-as-Policy multi-agent self-learning harness for variational automation tasks. In *Proceedings of The 10th Conference on Robot Learning (CoRL)*, Austin, TX, USA, 2026. URL https://arxiv.org/abs/2607.05369.

[15] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, Oct. 2012. doi:10.1109/IROS.2012.6386109.

[16] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac Gym: High performance GPU based physics simulation for robot learning. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

[17] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, June 2023. doi:10.1109/LRA.2023.3270034.

[18] M. Mittal, P. Roth, J. Tigue, A. Richard, O. Zhang, et al. Isaac Lab: A GPU-accelerated simulation framework for multi-modal robot learning. arXiv preprint arXiv:2511.04831, 2025. URL https://arxiv.org/abs/2511.04831.

[19] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, et al. Demonstrating GPU parallelized robot simulation and rendering for generalizable embodied AI with ManiSkill3. In *Proceedings of Robotics: Science and Systems (RSS)*, Los Angeles, CA, USA, June 2025. doi:10.15607/RSS.2025.XXI.021.

[20] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, C. Sferrazza, Y. Tassa, and P. Abbeel. Demonstrating MuJoCo Playground. In *Proceedings of Robotics: Science and Systems (RSS)*, Los Angeles, CA, USA, June 2025. doi:10.15607/RSS.2025.XXI.020.

[21] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, K. Lin, A. Maddukuri, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning. arXiv preprint arXiv:2009.12293, 2020. URL https://arxiv.org/abs/2009.12293.

[22] M. Macklin. Warp: A high-performance Python framework for GPU simulation and graphics, Mar. 2022. URL https://github.com/NVIDIA/warp. NVIDIA GPU Technology Conference (GTC).

[23] M. Macklin, M. Müller, and N. Chentanez. XPBD: Position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games (MIG)*, MIG '16, pages 49–54. ACM, Oct. 2016. doi:10.1145/2994258.2994272.

[24] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, et al. Do as I can, not as I say: Grounding language in robotic affordances. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

[25] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as Policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, May 2023. doi:10.1109/ICRA48891.2023.10160591.

[26] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, May 2023. doi:10.1109/ICRA48891.2023.10161317.

[27] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

[28] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. ChatGPT for robotics: Design principles and model abilities. *IEEE Access*, 12:55682–55696, 2024. doi:10.1109/ACCESS.2024.3387941.

[29] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, et al. Inner Monologue: Embodied reasoning through planning with language models. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pages 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

[30] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, et al. Language to rewards for robotic skill synthesis. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 374–404. PMLR, 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

[31] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

[32] K. E. Chen, S. Xie, Z. Ma, P. Sanketi, and K. Goldberg. Robo2VLM: Improving visual question answering using large-scale robot manipulation data. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pages 24644–24679. Curran Associates, Inc., 2025. doi:10.52202/085713-0733. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1f467c3e37abf9f86c78f44c6a27ee7c-Abstract-Datasets_and_Benchmarks_Track.html.

[33] S. Xie, K. Chen, Z. Chen, S. Adebola, Y. Huang, Z. Ma, T. Qiu, W. Yuan, D. Shah, P. R. Sanketi, and K. Goldberg. RoboVista: Evaluating vision language models for diverse robot applications. In *Proceedings of Robotics: Science and Systems (RSS)*, Sydney, Australia, July 2026. doi:10.15607/RSS.2026.XXII.095.

[34] Y. J. Ma, W. Liang, H.-J. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman. DrEureka: Language model guided sim-to-real transfer. In *Proceedings of Robotics: Science and Systems (RSS)*, Delft, Netherlands, July 2024. doi:10.15607/RSS.2024.XX.094.

[35] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang. GenSim: Generating robotic simulation tasks via large language models. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=OI3RoHoWAN.

[36] P. Hua, M. Liu, A. Macaluso, Y. Lin, W. Zhang, H. Xu, and L. Wang. GenSim2: Scaling robot data generation with multi-modal and reasoning LLMs. In P. Agrawal, O. Kroemer, and W. Burgard, editors, *Proceedings of The 8th Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 5030–5066. PMLR, 2025. URL https://proceedings.mlr.press/v270/hua25a.html.

[37] W. Ye, F. Liu, Z. Ding, Y. Gao, O. Rybkin, and P. Abbeel. Video2Policy: Scaling up manipulation tasks in simulation through internet videos. arXiv preprint arXiv:2502.09886, 2025. URL https://arxiv.org/abs/2502.09886.

[38] Z. Mandi, Y. Weng, D. Bauer, and S. Song. Real2Code: Reconstruct articulated objects via code generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=CAssIgPN4I.

[39] M. Ahn, D. Dwibedi, C. Finn, M. Gonzalez Arenas, K. Gopalakrishnan, et al. AutoRT: Embodied foundation models for large scale orchestration of robotic agents. arXiv preprint arXiv:2401.12963, 2024. URL https://arxiv.org/abs/2401.12963.

[40] L. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, L. Fei-Fei, G. Shi, J. Wu, S. S. Sastry, Y. Zhu, K. Goldberg, and L. Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In T. Zhang, M. Dudik, M. Jaggi, A. Agarwal, S. Li, D. Schuurmans, J. Zhu, F. Berkenkamp, H. Dong, and A. Bietti, editors, *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, volume 306 of *Proceedings of Machine Learning Research*, pages 32155–32212. PMLR, 2026. URL https://proceedings.mlr.press/v306/fu26u.html.

[41] J. Shi, R. Yang, K. Chao, S. B. Wan, Y. Shao, J. Lei, J. Qian, L. Le, P. Chaudhari, K. Daniilidis, C. Wen, and D. Jayaraman. Maestro: Orchestrating robotics modules with vision-language models for zero-shot generalist robots. arXiv preprint arXiv:2511.00917, 2025.

[42] R. Lu, Y. Wu, E. Kou, L. Fu, W. Xiao, A. Mandlekar, Y. Xu, G. Shi, K. Goldberg, A. Chen, M. Chowdhury, Y. Zhu, L. Fan, and G. Wang. ASPIRE: Agentic /skills discovery for robotics. arXiv preprint arXiv:2607.00272, 2026.

[43] W. Xiao, J. Xie, T. Zhang, H. Lin, L. Fu, H. Xue, J. Lu, Y. Yang, C. Dai, Z. Wang, J. Wu, G. Wang, S. S. Sastry, K. Goldberg, L. Fan, Y. Zhu, and G. Shi. ENPIRE: Agentic robot policy self-improvement in the real world. In *Proceedings of The 10th Conference on Robot Learning (CoRL)*, 2026.

[44] Y. Wang, Y. Li, L. Fei-Fei, J. Wu, and H. Huang. DexAgent: An agentic Human2Sim2Robot framework for dexterous manipulation with self-evolving tool library. arXiv preprint arXiv:2609.35318, 2026.

[45] Y.-J. Wang, H. Jiang, S. Deng, H. Xue, W. Ye, R. Duan, N. Haghtalab, S. S. Sastry, P. Abbeel, and H. Qi. Reconstruct, practice, go real: Guided self-improvement for embodied agents. arXiv preprint arXiv:2610.02204, 2026.

[46] S. Xie, K. Chen, and K. Goldberg. Revisiting the "Push-T" robot manipulation task with agentic robotics. arXiv preprint arXiv:2608.18227, 2026. URL https://arxiv.org/abs/2608.18227.

[47] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 50528–50652. Curran Associates, Inc., 2024. doi:10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

[48] M. Torne Villasevil, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, Delft, Netherlands, July 2024. doi:10.15607/RSS.2024.XX.015.

[49] Q. Chen, A. Walsman, M. Memmel, K. Mo, A. Fang, D. Fox, and A. Gupta. URDFormer: A pipeline for constructing articulated simulation environments from real-world images. In *Proceedings of Robotics: Science and Systems (RSS)*, Delft, Netherlands, July 2024. doi:10.15607/RSS.2024.XX.124.

[50] T. Dai, J. Wong, Y. Jiang, C. Wang, C. Gokmen, R. Zhang, J. Wu, and L. Fei-Fei. Automated creation of digital cousins for robust policy learning. In P. Agrawal, O. Kroemer, and W. Burgard, editors, *Proceedings of The 8th Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 4912–4943. PMLR, 2025. URL https://proceedings.mlr.press/v270/dai25a.html.

[51] L. Le, J. Xie, W. Liang, H.-J. Wang, Y. Yang, Y. J. Ma, K. Vedder, A. Krishna, D. Jayaraman, and E. Eaton. Articulate-Anything: Automatic modeling of articulated objects via a vision-language foundation model. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=s3FTX4Ay55.

[52] X. Li, K. Hsu, J. Gu, O. Mees, K. Pertsch, et al. Evaluating real-world robot manipulation policies in simulation. In P. Agrawal, O. Kroemer, and W. Burgard, editors, *Proceedings of The 8th Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

[53] M. N. Qureshi, S. Garg, F. Yandun, D. Held, G. Kantor, and A. Silwal. SplatSim: Zero-shot Sim2Real transfer of RGB manipulation policies using Gaussian splatting. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6502–6509. IEEE, May 2025. doi:10.1109/ICRA55743.2025.11128339.

[54] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 51936–51983. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24cc.html.

[55] Y. Yang, F.-Y. Sun, L. Weihs, E. Vanderbilt, A. Herrasti, W. Han, J. Wu, N. Haber, R. Krishna, L. Liu, C. Callison-Burch, M. Yatskar, A. Kembhavi, and C. Clark. Holodeck: Language guided generation of 3D embodied AI environments. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16277–16287. IEEE, June 2024. doi:10.1109/CVPR52733.2024.01536.

[56] A. Goldberg, K. Kondap, T. Qiu, Z. Ma, L. Fu, J. Kerr, H. Huang, K. Chen, K. Fang, and K. Goldberg. Blox-Net: Generative design-for-robot-assembly using VLM supervision, physics simulation, and a robot with reset. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15493–15500. IEEE, May 2025. doi:10.1109/ICRA55743.2025.11127489.

[57] Google. Gemini 4 Argon, 2026. Released to developers October 13, 2026.

[58] Blender Online Community. Blender: A 3D modelling and rendering package, 2026. URL https://www.blender.org. Blender Foundation, Amsterdam.

[59] M. Li, Z. Ferguson, T. Schneider, T. Langlois, D. Zorin, D. Panozzo, C. Jiang, and D. M. Kaufman. Incremental Potential Contact: Intersection- and inversion-free, large-deformation dynamics. *ACM Transactions on Graphics*, 39(4), Aug. 2020. doi:10.1145/3386569.3392425.

A Related Work

Table 1: **Comparison of robotics simulation frameworks.** Comparison of physics capabilities, interactive interfaces, and Real2Sim workflows. ✓ indicates built-in support, ∼ indicates partial support, and "Custom" indicates that additional implementation is required.

Feature	MuJoCo	IsaacLab	ManiSkill3	Genesis	Newton	RoboSimStudio
Physics Simulation						
Rigid/articulated bodies	✓	✓	✓	✓	✓	✓
Deformable objects	✓	✓	∼	✓	✓	✓
Coupled multi-physics solvers	–	∼	–	✓	∼	✓
Interactive Interfaces						
Interactive scene inspection	✓	✓	✓	✓	✓	✓
Physics parameter tuning	∼	✓	∼	∼	∼	✓
Structured feedback for coding agents	Custom	Custom	Custom	Custom	Custom	✓
Shared human-agent refinement	Custom	Custom	Custom	Custom	Custom	✓
Real2Sim Workflows						
Video-driven agentic Real2Sim	–	–	–	–	–	✓
Force-feedback Real2Sim	–	∼	–	–	–	✓
Failure-driven simulation refinement	–	–	–	–	–	✓

Physics Simulation for Robot Learning. Physics simulation has enabled large-scale robot learning by providing environments for policy training and evaluation. MuJoCo [15] supports articulated robot dynamics and contact-rich manipulation, while GPU-accelerated frameworks such as Isaac Gym and Isaac Lab [16, 17, 18], ManiSkill3 [19], MuJoCo Playground [20], and Genesis [2] support parallel simulation for robot learning. Frameworks such as robosuite [21] further provide standardized manipulation environments. More recently, Newton [1], built on Warp [22], integrates multiple physics solvers, including MuJoCo-Warp [10], Vertex Block Descent (VBD) [11], and XPBD [23]. Other physics engines, such as SuperDex [3], provide specialized models for contact-rich soft-body simulation. While these frameworks provide increasingly diverse physics capabilities, constructing manipulation environments still requires selecting appropriate solvers, configuring contact models, and tuning physical parameters. RoboSimStudio builds on Newton and supports co-simulation with external physics engines, allowing different physics models to be composed within the same scene and interactively refined by humans and coding agents.

Agentic Robotics. Large language models (LLMs) are now being used in robotics for task planning, robot control, and policy generation. Early approaches use language models to ground instructions in robot affordances [24], generate plans and executable robot programs [25, 26, 27, 28], and improve robot behavior through feedback [29]. More recent works use LLMs to generate reward functions [30, 31], visual question answering [32, 33], domain randomization strategies [34], and simulation tasks for robot learning [35, 36, 37]. With iterative reasoning and acting frameworks such as ReAct [6], general-purpose coding agents [4, 5] can modify code, execute programs, and refine their solutions based on feedback. Such agents have been applied to reconstructing articulated objects [38], robot data collection [39], and iterative policy improvement, such as CaP-X [40] and GaP [14]. MAESTRO [41] orchestrates robotics modules through VLM-written programs, ASPIRE [42] grows a reusable code skill library across many simulated tasks, and ENPIRE [43] lets coding agents improve policies through real-world trial and error. DexAgent [44] turns a single egocentric human video into dexterous robot trajectories, and RPG [45] practices tasks in simulation and revises skills from diagnosed failures before real-world deployment. A coding agent has also solved the Push-T benchmark without demonstrations by running its own experiments in simulation [46]. The effectiveness of these agents also depends on their ability to interact with external environments and access execution feedback [47]. RoboSimStudio builds on this insight by providing coding agents with an interactive physics simulation environment, where scene configurations,

physics models, and simulation feedback are directly accessible for agentic Real2Sim construction and refinement.

Real2Sim. Real2Sim approaches reconstruct simulation environments from real-world observations to support robot policy training and evaluation. Prior work has explored constructing simulations of real scenes, and variations of them, from images, videos, and other observations [48, 49, 50, 51, 52, 53]. Other approaches use generative models and LLMs to automatically construct simulation tasks, environments, and reward functions [35, 54, 55, 31], or generate robot policies directly [25, 14]. Blox-Net [56] uses vision-language models to design block assemblies, refines the designs through perturbation analysis in simulation, and builds them with a real robot. These approaches reduce the effort required for scene construction and policy development, but often focus on specific reconstruction or generation pipelines. Recent frontier models, such as Claude Fable 5.1 [4], GPT-6 Astra [5], and Gemini 4 Argon [57], can assist in reconstructing visually plausible 3D scenes from videos, including producing Blender [58] renderings. However, visual realism does not guarantee physical accuracy, particularly for contact-rich interactions and deformable objects such as cables, cloth, and sponges. RoboSimStudio therefore exposes editable physics models, contact parameters, and simulator feedback, enabling humans and coding agents to iteratively refine simulated physical behavior to match real-world observations rather than appearance alone.

B System Details

This appendix expands Section 2: how a scene is described, which parameters are exposed, and what the studio offers beyond the material controls of Figure 3.

Describing a scene. A scene in RoboSimStudio is a short program (Fig. 7). It names a work cell, adds a robot, and places objects on the work surface. An object states what it is—a sheet of cloth, a foam sponge, a rigid tool—and never how it is simulated. Building the scene partitions it among solvers, couples them, and returns a running simulation. One more line opens it in the studio. The same description is what the studio displays and what an agent edits, so the three interfaces cannot drift apart.

Physics as a modeling choice. Real manipulation scenes mix materials. A wiping task couples an articulated arm, a compliant sponge, and a rigid bar; a routing task couples a gripper with a cable that bends and twists. No single solver handles all of these well. RoboSimStudio groups the objects in a scene by the physics they need and gives each group a suitable solver: articulated robots to MuJoCo-Warp [10], cables and cloth to Vertex Block Descent [11] or XPBD [23], and soft bodies to finite-element or incremental-potential-contact [59] engines co-simulated beside Newton [3, 2]. The engine behind an object is one entry in the scene description, not a rewrite of the scene. Figure 8 shows the same cable pick with the cable modeled three ways: as a chain of capsules in Newton, as a Cosserat rod in an external finite-element engine, and as a tetrahedral bar in an incremental-potential-contact engine.

Tuning physics interactively. A simulation is rarely right on the first build. Cloth that is too stiff will not drape; a sponge that is too slippery will not wipe. RoboSimStudio exposes the parameters that decide these outcomes—about a hundred in total—in five groups: material properties of each asset (stiffness, damping, density), contact (friction, restitution, margins), solver settings (rates, iterations), coupling between solvers, and robot drives. Each parameter appears in the studio as a control with its unit and range, generated from the same description the solver reads (Fig. 9a). Material stiffness and damping, the cable latch, and soft contact take effect on the next substep of the running scene, so their effect is visible immediately (Fig. 3). Other changes re-capture the step program or rebuild the scene, which takes seconds, and the studio marks which is which. The studio is built on Viser [7] and runs in a browser. The simulation can run on a remote GPU while the page is open on a laptop, or shared with a collaborator through a link.

```python
import robosimstudio as rss

scene = rss.Scene(cell="bench_075")    # a work cell
scene.add(rss.robot("yam_bimanual"))   # two arms
scene.add(rss.sheet(name="towel", size=(0.3, 0.3),
                    at=(-0.30, -0.15, 0.02)))
scene.add(rss.sponge(name="sponge", at=(0.18, 0.0, 0.0)))

sim = scene.build()   # partition, couple, schedule
rss.view(sim)         # open the studio in a browser
```

```
physics: proxy over 3 island(s)
  island mujoco:   robot  (16 bodies, 16 joints)
  island superdex: sponge (440 particles)
  island vbd:      towel  (961 particles)
  channel mujoco -> vbd: 6 bodies (two_way)
```

```python
from robosimstudio.specs.physics import backend

physics = rss.replace(rss.GENERIC, backends=(
    ("articulated", backend("mujoco")),
    ("cloth", backend("superdex", substeps=2)),  # new
    ("soft",  backend("superdex", substeps=2)),
    ("*",     backend("vbd")),
))
scene = rss.Scene(cell="bench_075", physics=physics)
```

```
physics: none over 2 island(s)
  island mujoco:   robot  (16 bodies, 16 joints)
  island superdex: towel, sponge (1401 particles)
```

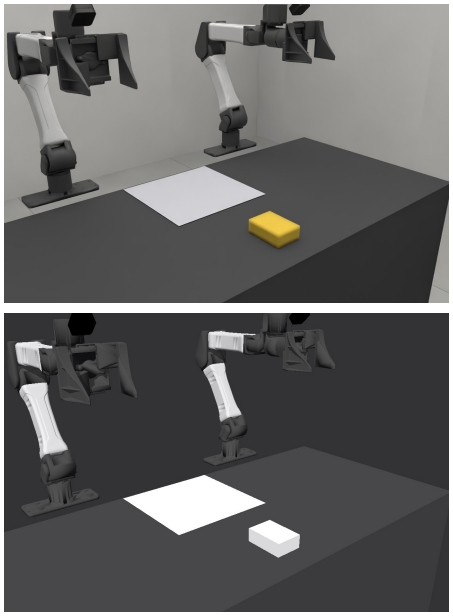


Figure 7: **A scene in a few lines.** Top left: a work cell, two arms, a towel, and a sponge. The objects state what they are and where they sit on the work surface; building the scene assigns each a solver and couples them (grey: the composer's summary, abridged). Bottom left: one added line moves the towel to another engine, and the composer regroups the scene from three islands to two. Right: the scene as built, path traced (top) and rasterized (bottom).

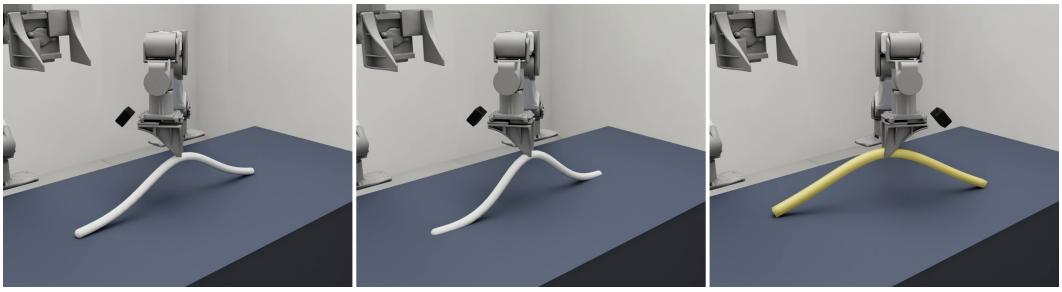


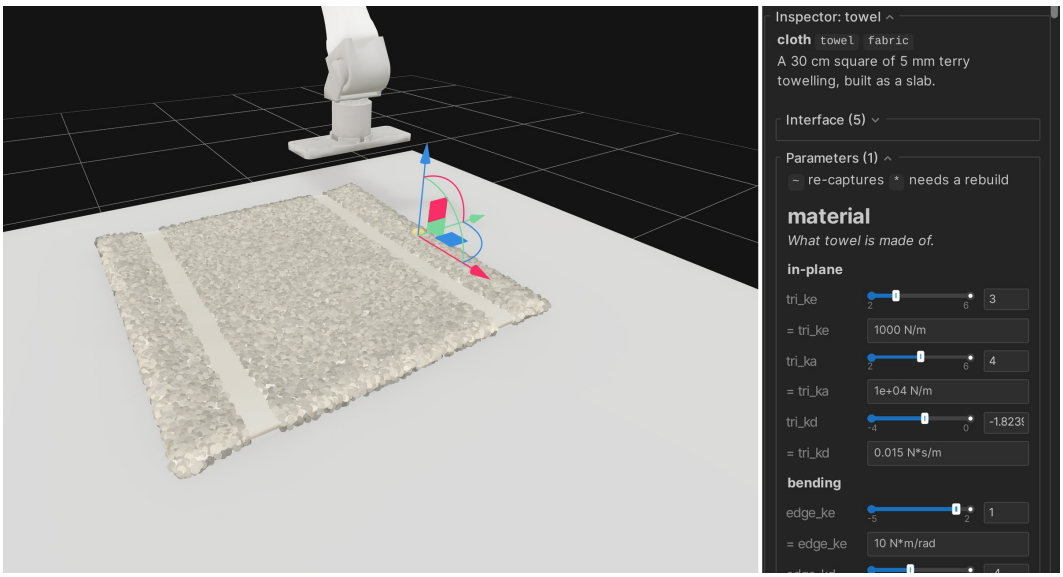
Figure 8: **One cable, three engines.** Switching the engine behind one object is a single change to the scene; the arm stays in MuJoCo-Warp throughout. From left: Newton's Vertex Block Descent, SuperDex, and Genesis.

Parameters. Table 2 breaks down the parameters the studio exposes. A parameter is counted only if it is described: it has a unit, a range worth sweeping, and a stated cost of changing it. Solvers and couplers that register no description still run; the studio shows their settings as plain inputs that rebuild the scene.

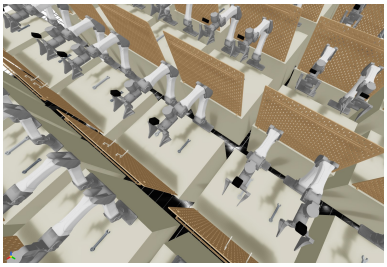
Interacting with the scene. Every object in the studio can be grabbed and dragged. Every arm can be driven by a handle on its end effector or, with a low-cost leader arm [13], by direct teleoperation (Fig. 9c). The leader arm is active. Its joint angles drive the simulated arm, and RoboSimStudio reads back where the simulated arm actually is. When contact holds the simulated arm back—a jaw closed on a part, a link against a fixture—the gap between the two is rendered on the leader's own motors, and the operator feels the scene push back. In free space the motors stay off: a gap below a threshold that grows with the hand's speed is the simulated arm's lag, not contact. The leader runs on its own clock, faster than the scene, and its motors are released whenever the scene pauses or falls behind. Interaction serves two purposes. It lets a person probe the physics—push the sponge,

Table 2: **Tunable parameters by group.** Each parameter carries a unit and a useful range, from which the studio generates its controls. Thirteen take effect on the next substep; the others re-capture the step program or rebuild the scene.

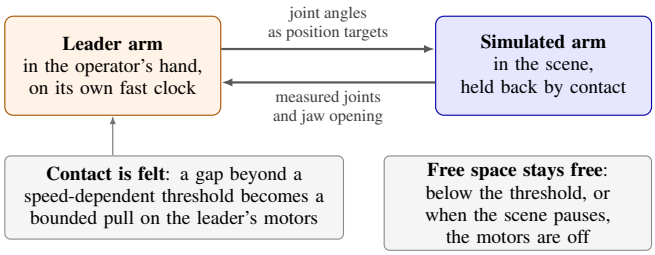
Group	What it covers	Count
Asset	Cloth in-plane and bending stiffness, damping, and density; foam elasticity, density, and damping; cable bending and stretch stiffness; rigid mass	38
Contact	Friction, restitution, contact stiffness and damping; collision margins and detection settings	19
Solver	Control and substep rates; iteration counts, tolerances, and contact buffers of each Newton solver	30
Coupling	Proxy mass scaling, passes, and relaxation; constraint-based coupling penalties	13
Robot	Joint drive gains, armature, friction, and limits; cable latch gains and force bound	9
Total		109



(a) The studio: a towel selected in the scene, and its inspector with material controls (enlarged).



(b) Batched worlds of one task.



(c) Teleoperation with an active leader arm.

Figure 9: **The interactive studio.** (a) Every object can be selected, dragged, and inspected; its parameters are controls with units, and a legend marks the ones that need a re-capture or a rebuild. (b) The same page shows a fleet of batched worlds. (c) A leader arm drives the simulated arm; where contact holds the simulated arm back, the gap between the two is rendered on the leader's motors, so the operator feels the scene.

drop the cable—before writing any policy, and it produces demonstrations recorded with the state

of every body, joint, and particle. Recorded runs can be replayed and re-rendered offline with a path tracer for photorealistic video (Fig. 5c).

Authoring tasks. A scene becomes a task when it has a goal. RoboSimStudio describes tasks declaratively: the objects, the distribution of their initial conditions, and a goal stated as relations among objects measured against thresholds—a wrench enclosed by its peg and at rest, a crate turned over and contained in a box. An evaluator checks the goal at every control step, and the policy cannot override it: a policy may ask to end an episode, but it cannot declare its own success. Variation is part of the description. One task yields many instances with different object poses, counts, and difficulty levels (Fig. 15e). The current gallery of about twenty tasks (Fig. 4, Appendix E) spans tool sorting and hanging, cable routing and weaving, connector insertion, gear meshing, sharps disposal, tape and film peeling, cloth folding, surface wiping, and crate flipping, on Franka Panda, UR5e, and bimanual YAM arms.

Running at scale. The same scene runs as one interactive world or as many batched worlds on one GPU. Batched execution supports data generation and evaluation over task variations, and the studio can show hundreds of worlds at once (Fig. 9b). Rigid and articulated scenes batch today; scenes with cables and cloth still run one world at a time. Appendix D reports throughput.

C Multi-Physics Coupling

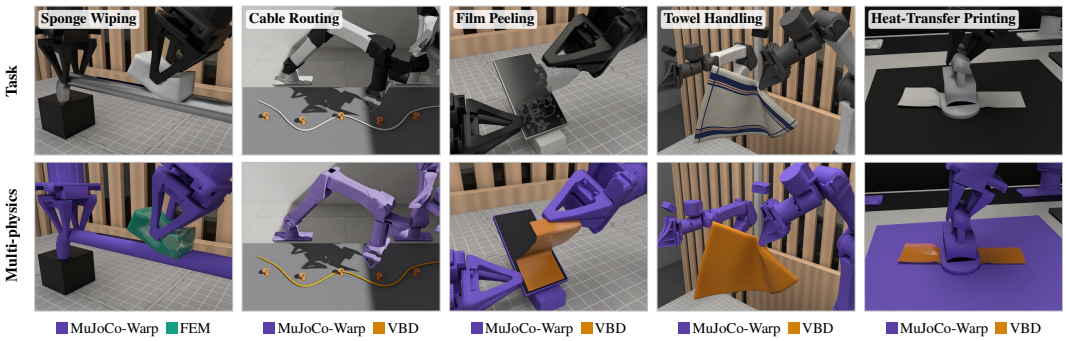


Figure 10: **Multi-physics coupling in RoboSimStudio.** Five manipulation tasks using different physics solvers and coupling methods. Top: rendered simulation scenes. Bottom: visualization of the physics solvers used for each object, including MuJoCo-Warp (violet), finite-element simulation (FEM, teal), and Vertex Block Descent (VBD, amber). All tasks use the same robot interface and scene description framework in RoboSimStudio.

Figure 10 shows five manipulation tasks that use different physics solvers and coupling methods. For sponge wiping, the sponge is a finite-element soft body in the external SuperDex engine [3], co-simulated with Newton by passing rigid-body poses to SuperDex and returning contact forces to the rigid-body solver at each control step. For cable routing, the cable is a chain of rigid capsules in Vertex Block Descent, and the robot hands enter that solver as proxy bodies, so that contact forces from the cable are transferred back to the robot. The choice of solver, coupling method, and physical parameters is changed in the scene description, without changing the robot interface or the task.

An arm wants a rigid-body solver with accurate joints, limits, and drives. A cable or a towel wants an implicit solver that stays stable on thin, stiff deformables. A sponge wants a volumetric model. RoboSimStudio lets each object be simulated by the solver that suits it, and couples the solvers where the objects touch. This appendix describes how a scene is partitioned, what crosses each boundary, and how the result is scheduled (Fig. 11).

Who states what. RoboSimStudio keeps apart three statements that simulation code usually mixes. An *asset* states what it is and what it may touch: a cable is a rod that hands must be able

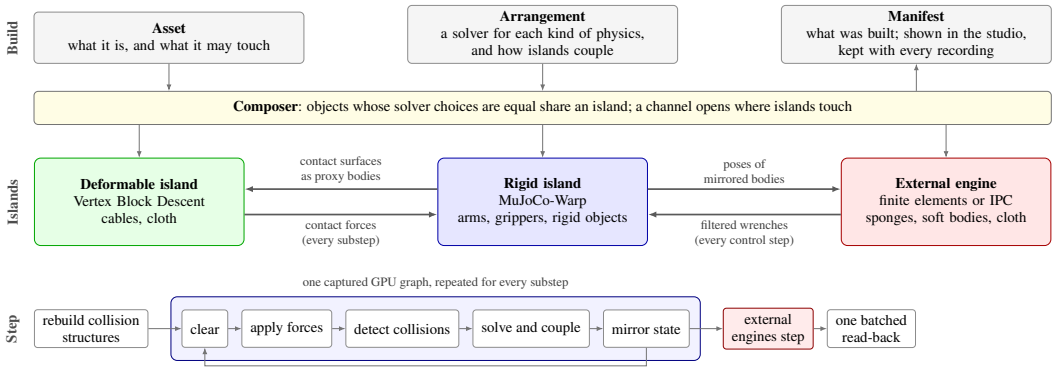


Figure 11: **Physics composition.** Top: an asset states what it is, a scene's arrangement states which solver serves each kind of physics, and the composer records what it built. Middle: objects are grouped into islands, one per solver, and contact is exchanged where islands touch: proxy bodies and contact forces between Newton's own solvers, poses and wrenches between Newton and an external engine. Bottom: every substep runs as one captured GPU graph; external engines step after it, and state is read back once per control step.

to push. It never names a solver. A scene's *arrangement* states which solver serves each kind of physics and how groups of objects couple. It never names an object. The *composer* reads both when the scene is built, decides which solver receives which object's state, and records the result as a *manifest*. The manifest is what the studio displays and what every recording carries, so a run can be rebuilt from its record alone.

Islands. The composer groups objects into *islands*. Each object resolves to a solver choice: its own if it states one, otherwise the arrangement's choice for its kind of physics. Objects whose choices are equal share an island and are solved together. Nothing more is needed for a cable and a towel to interact inside one implicit solve; the arrangement names the same solver for both. In the default arrangement the robot is an island in MuJoCo-Warp, soft bodies form an island in an external finite-element engine, and everything else shares a Vertex Block Descent island. A task can restate any of these, and the scene in Fig. 7 shows the regrouping that follows.

Table 3: **Coupling methods in RoboSimStudio.** The first three couple Newton's own solvers inside the captured step. Co-simulation couples Newton to an engine outside it.

Method	What crosses the boundary	Exchanged	Used for
Proxy	Rigid surfaces enter the deformable island as proxy bodies; contact forces return	Every substep, once or iterated	A hand that must feel what it grasps: cables
One-way	Rigid poses lead the deformable island as a moving boundary	Every substep	Contact over a deformable's full surface: cloth folding, film and tape peeling
Constraint-based	Contacts and attachments shared as constraints across islands	Iterated within a substep	More than two islands; attachments that span solvers
Co-simulation	Body poses go out as kinematic mirrors; contact wrenches return	Once per control step	Soft bodies and cloth in an external engine

Coupling inside Newton. Where two islands touch, the composer opens a *channel* between them, and a coupler exchanges contact across it within every substep (Table 3). In *proxy* coupling, the surfaces of a rigid island that may touch a deformable—a gripper's fingers, or a whole object—enter the deformable island as proxy bodies that follow the rigid solver's motion. The deformable solver collides against the proxies, and the contact forces it finds return to the rigid solver as forces on the original bodies. One exchange per substep is an explicit coupling with one substep of lag; repeating the exchange within a substep tightens it at proportional cost. Proxy coupling is what lets a hand

feel the cable it pulls. *One-way* coupling drops the return path, and the rigid island leads the deformable one as a moving boundary. RoboSimStudio uses it where a deformable needs contact over its whole surface and the robot is far stiffer than what it handles, as in cloth folding and film peeling. *Constraint-based* coupling poses contacts and attachments across islands as shared constraints and solves them iteratively. It serves scenes with more than two islands or with attachments that span solvers; it is available, but no task in the current gallery uses it.

Co-simulating external engines. Some physics is better served by engines outside Newton. RoboSimStudio currently couples two: a finite-element engine for soft bodies and rods (SuperDex [3]) and an incremental-potential-contact engine for cloth and soft bodies (Genesis [2]). The contract is the same for both: *pose in, wrench out*. The external engine receives the Newton bodies its objects may touch as kinematic mirrors, driven by the poses Newton computed in this control step. It advances its own objects against them, and returns the contact wrench on each mirror, which RoboSimStudio applies to the Newton body. Three details make this robust. Only shapes that Newton itself would collide are mirrored, so a hand's visual meshes and reference frames never press on anything. Mirror motion is interpolated across the external engine's own substeps, so a fast hand does not pass through a thin object between two poses. And the returned wrench is smoothed as it rises and bounded in the velocity change it may cause, because an unfiltered wrench applied one step late can launch a light body. The external objects' particles still live in Newton's state, marked as not integrated there. Rendering, dragging, recording, and task evaluation therefore work on them without knowing which engine moved them.

Adding the second engine changed no code in the composer, which is our evidence that the seam is general. What did not transfer was an idiom. One engine treats a mirror as a static actor that can be placed anywhere before its first step; the other holds a mirror with a soft constraint, so a mirror created away from its body drags the scene with it. Each engine therefore keeps a small bridge of its own, and the rest—mirror motion, wrench feedback, scheduling—is written once.

One step program. All of this runs under one schedule (Fig. 11, bottom). A substep is a fixed sequence of stages: clear and apply forces, detect collisions, solve and couple, and mirror state for the solvers that need it. RoboSimStudio captures the whole substep loop, couplers included, as a single GPU graph, so the physics loop moves no data between device and host. External engines step once per control step after the graph has run. Everything the environment, the renderer, and the recorder need is then read back in one batched copy. A coupled scene therefore pays for its solvers, not for the composition (Appendix D).

Holding deformables. Contact alone does not yet carry every grasp, and RoboSimStudio makes the alternatives explicit rather than hiding them. A sponge can be lifted by friction between two fingers. A cable is held by a latch: a bounded spring between a cable segment and the gripper, engaged when the fingers close on it. Cloth is held by pinning the particles under the fingertips to the hand, and soft bodies can be pinned the same way. Each attachment is a tunable part of the scene, and Appendix C.1 lists what this leaves unmodeled.

Extending the composition. Every extension point is a named registry rather than a class the toolkit must know. A solver that Newton releases tomorrow is one entry in a scene's arrangement. A new kind of physics, a new coupler, or a new external engine registers from outside the toolkit, together with the units, ranges, and update cost of its parameters, from which the studio generates its controls.

C.1 Physics Composition Details

From declarations to a schedule. The composer runs once, when a scene is built, in a fixed order. It first builds an interaction graph whose nodes are the objects, with the state each owns and the roles it plays, and whose edges are the contact rules the assets declared. It then forms islands, opens a channel for every contact between a rigid island and a deformable one, and routes every candidate

contact pair to exactly one collision pipeline: a channel's own if the pair crosses that channel, the main one otherwise. Finally it builds the solvers, lays out the step program, and writes the manifest. A declaration the arrangement cannot honor is an error at build time that names the asset, the rule, and the arrangement, rather than a scene that silently simulates something else.

Why external engines step last. External engines step after the captured substeps rather than before them. Newton's deformable solver writes velocities for every particle in the state, including those an external engine owns, so particles written before the substeps would be overwritten by them. Stepping last also lets the engine read the body poses of the control step that has just been computed, at the cost of one small read from the device. The alternative, reading the previous step's poses, costs nothing and leaves the mirrors one control step stale; RoboSimStudio defaults to the fresh poses.

Documented limits of the coupling. The coupling is explicit, and its limits are stated in the toolkit's documentation and tests:

- A mirror that moves farther in one external-engine step than the thickness of the object it meets can pass through it. Fast contacts need more engine substeps or more contact padding.
- Proxy coupling pushes a deformable but does not pinch one. A cable is therefore held by a latch and cloth by pinned particles (Appendix C).
- A light, free rigid body resting on a co-simulated soft body is out of scope: the wrench it receives is one control step late, and it creeps or is launched. Arms and driven tools are the supported case.
- The incremental-potential-contact engine must run in double precision. In single precision the force it returns vanishes as the coupling stiffens, which reads as a loose coupling rather than as an error.
- Contact between a deformable in Newton and a deformable in an external engine is not coupled.
- Batched worlds are supported for Newton's islands and for one of the two external engines.

D Throughput

Table 4: **Throughput in detail.** Islands and channels are what the composer built. The control rate is the full environment step; the physics rate is the solvers alone; the overhead is the difference per control step. One world, one RTX PRO 6000 GPU, 120 steps after 60 warm-up steps, each scene in its own process.

Scene	Islands	Channels	Control (steps/s)	Physics (steps/s)	Overhead (ms)
Tool sorting	1	0	113.9	123.1	0.7
Cloth folding	2	0	34.8	37.4	2.0
Cable routing	2	1	10.4	10.5	0.9
Film peeling	2	0	5.8	5.8	1.6
Sponge pressing	2	0	138.2	158.2	0.9

Table 4 reports the rate of representative scenes with a single world. The rigid scene and the co-simulated sponge run at over a hundred control steps per second. Deformable scenes solved inside Newton run an order of magnitude slower, set by the solver iterations their contacts need rather than by the composition layer: on top of the solvers, the environment adds about two milliseconds or less per control step in every scene. Every scene keeps the GPU busy for the whole physics period, so the control rate is set by the solvers. What the environment adds on top—gathering observations, evaluating the goal, and, for the sponge, stepping the external engine—is the overhead column. The sponge scene shows where an external engine's cost lands: its step runs outside the captured graph, so it appears in the control rate and not in the physics rate. Batched worlds amortize the cost of rigid scenes: on the same GPU, the tool-sorting scene runs 256 worlds at about real time and 512 worlds at 0.78 times real time, while a single cloth world runs at 0.58 times real time.

E Task Gallery

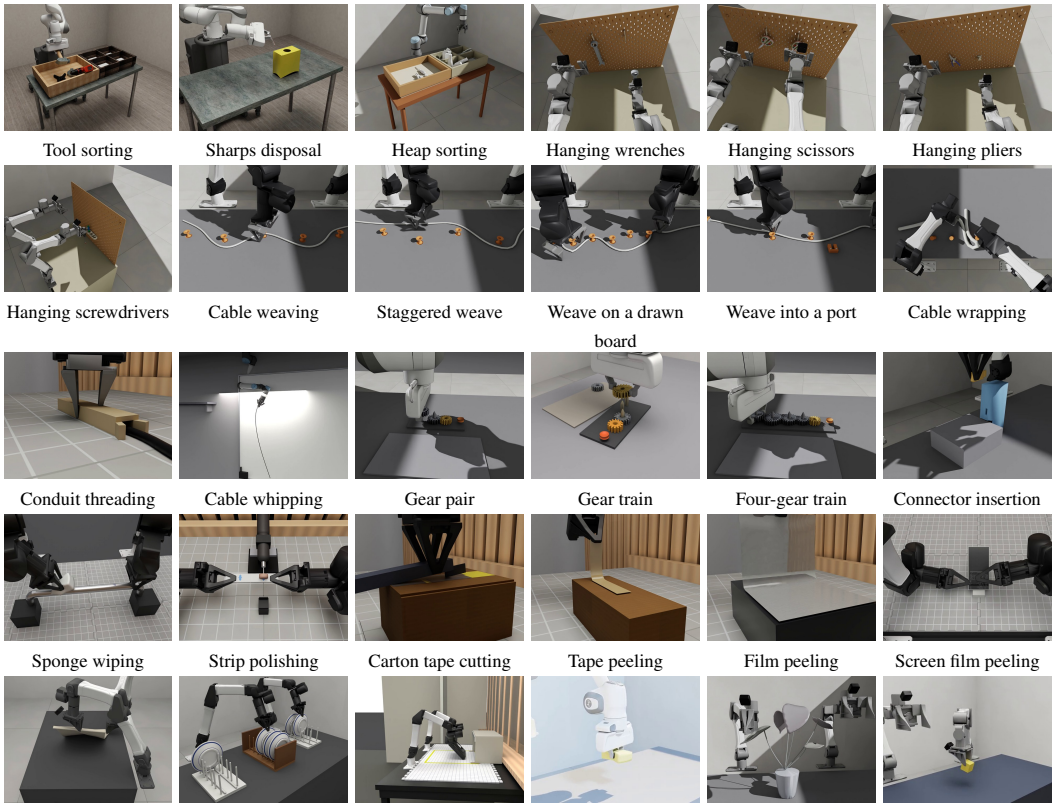


Figure 12: **Task gallery.** Thirty scenes in RoboSimStudio, each a frame of a recorded rollout re-rendered with the path tracer. The scenes cover three robots (Franka Panda, UR5e, bimanual YAM) and rigid, articulated, cable, cloth, film, and soft-body physics. Most are tasks with a goal and a shipped policy; cable whipping, screen film peeling, foam block lifting, leaf pressing, and cube lifting are example scenes and simulations without a goal.

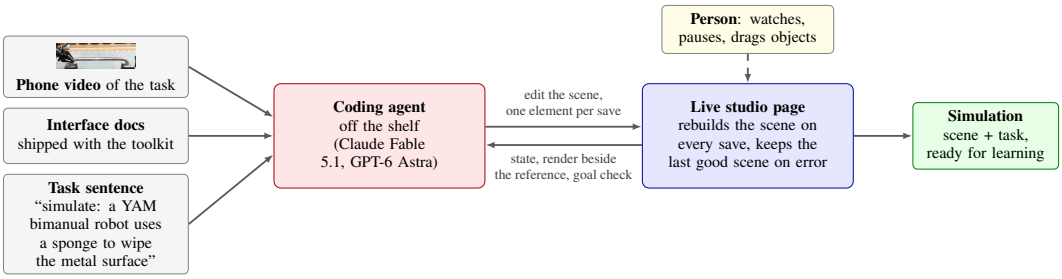
Table 5: **Task families in RoboSimStudio.** A family is one task document with declared variation; a variant fixes object sets, counts, layouts, and difficulty.

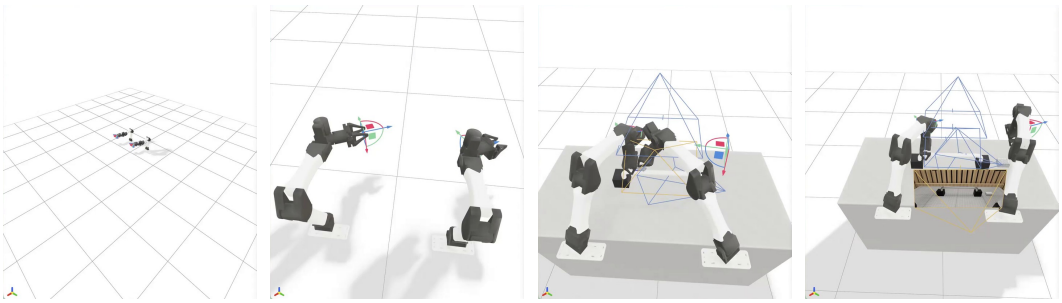
Task family	Variants	Robot	Physics	Coupling
Tool sorting	62	Franka Panda	rigid, articulated	none
Sharps disposal	64	Franka Panda	rigid, articulated	none
Gear meshing	200	Franka Panda	rigid, articulated	none
Tool hanging	208	bimanual YAM	rigid, articulated	none
Cable routing	202	bimanual YAM	cable, articulated	proxy
Connector insertion	202	bimanual YAM	rigid, articulated	none
Bench tasks	8	bimanual YAM	film, cable, soft, rigid	one-way, co-simulation
Surface wiping	2	bimanual YAM	soft body, articulated	co-simulation
Cloth folding	2	bimanual YAM	cloth, articulated	one-way
Crate flipping	4	bimanual YAM	rigid, articulated	none
Heat-transfer printing	1	bimanual YAM	cloth, rigid	one-way
Plate racking	1	bimanual YAM	rigid, articulated	none
Total	956			

Figure 12 shows the scenes in RoboSimStudio, and Table 5 lists the task families behind them. A task is a document: it names its objects, the distribution their initial conditions are drawn from, and a goal. A family declares its variation in the same document, so its variants differ in which

objects appear, how many, where, and how hard the layout is. The bench tasks are seven jobs on one calibrated bench: tape peeling, film peeling, carton tape cutting, decanting, conduit threading, strip polishing, and sponge wiping. Most rollouts in Fig. 12 run a shipped policy: a scripted routine, or a graph policy [14] given either simulator state or camera images.

F Agentic Real2Sim





1 min: empty page | 13 min: two arms | 46 min: bench, camera | 56 min: full scene

Figure 13: **Agentic real-to-sim.** Top: an off-the-shelf coding agent receives a phone video, the toolkit's interface documentation, and one sentence naming the task. It edits a scene while a live studio page rebuilds it, and checks each edit against the reference. A person can watch and intervene at any time. Bottom: the studio page during one build of the wiping simulation by Claude Fable 5.1, with elapsed time (Fig. 14 shows more of the build).

Starting from a video and a sentence. The agent receives three inputs. The first is a video of the task, recorded on a phone. The second is a description of RoboSimStudio's interface. The third is one sentence naming the task, such as "You need to simulate: a YAM bimanual robot uses a sponge to wipe the metal surface." The interface description is not a prompt written for one task. It is documentation that ships with the toolkit: how to start the studio, how scenes are described, which robots and assets exist, and how to check each change. Any coding agent that reads a repository's agent instructions picks it up on its own, so the per-task prompt only has to say which video to rebuild.

Rebuilding the scene on every save. The agent works on a single scene description while a studio page runs beside it. Every time the description is saved, the studio rebuilds the scene and swaps it into the open page without a reload. The camera stays where it was, and a description that fails to build leaves the previous scene running with the error shown. A rebuild takes seconds. Building a simulation becomes a tight edit-and-observe loop, the same loop a person uses in the studio. The agent's edits arrive one at a time on a page that a person can watch, pause, and interact with.

Looking before acting. The agent closes the loop visually. After each edit, it reads the scene's state—what was built, and where each object came to rest—and renders the scene from a camera placed where the reference was recorded, beside the reference frame. It compares the pair: what is present, how large it is, and where it sits. It builds one element at a time: the work surface, then the

robot, then each object, then the camera. Only when the pair agrees does it make the robot act, step the simulation, and check the task's goal.

What makes this tractable. Three properties of RoboSimStudio carry the weight. First, scenes are short declarative descriptions, so an agent can write one after reading the documentation and a few examples. Second, every state and parameter is explicit, so the agent can test a hypothesis—is the sponge resting on the bar?—by reading the state instead of guessing from pixels. Third, the side-by-side render gives a simple, general fidelity signal that needs no task-specific metric. What remains hard is physical rather than geometric: accurate robot motion, accurate robot–object contact, and accurate object–object contact. These are what the agent tunes in the second half of a build, and where simulations still fail (Section 4, Fig. 5e).

F.1 Build Details

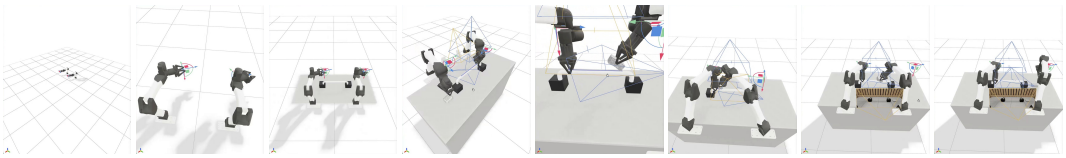


Figure 14: **One build, start to finish.** The studio page during the wiping build of Fig. 13, with elapsed time: an empty page, two arms, the bar on its supports, the bench with a camera at the reference viewpoint, a close look at the sponge grasp, and the finished scene with the camera's own rendering shown on the page.

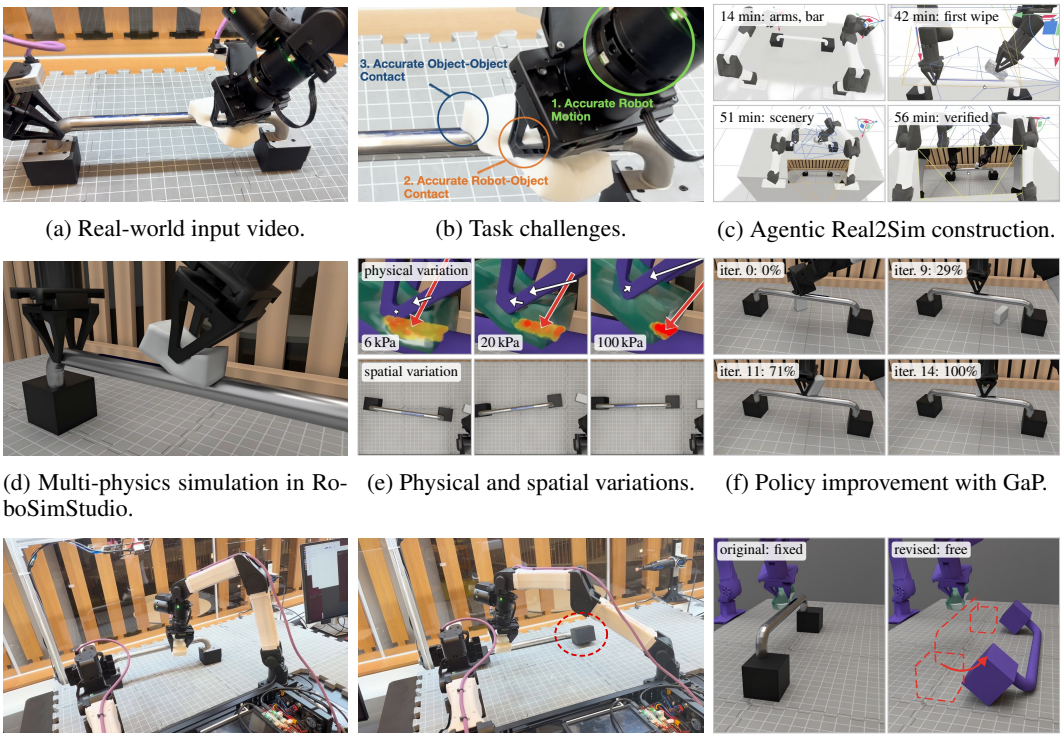
What the agent is given. The per-task prompt names the video and states the task in a sentence. Everything else is documentation that ships with RoboSimStudio and that a coding agent reads on its own: how to start the live page, how a scene is described, which robots and assets exist, and a procedure of one change per save, each followed by a look at the result. The prompt contains none of that procedure.

The live page. One process serves a studio page over a single scene description. Each save rebuilds the scene and swaps it into the open page without a reload, keeping the viewer's camera. A rebuild takes a few seconds, about half of which is collision geometry for the fingertips; the swap itself is not noticeable. A description that fails to build leaves the previous scene running and reports the error on the page. The agent queries the same page a person sees: the names and resting places of the objects, a rendering from any camera placed beside the reference frame, and the task's goal.

What the build reused. The build in Fig. 14 is honest about its starting point. The agent spent its first minutes reading an existing wiping arrangement in the toolkit and built on it: the foam material, the coupling of the arm to the sponge's finite-element engine, the camera rig, and the surface finishes were already there. What the session added was the scene re-measured from the video, a camera fitted to the reference view, the pressing and wiping motion, and a path-traced camera view on the page. The sponge is held by pinning the foam under each finger pad, and friction-only grasps were not attempted in that session. Building a simulation in about an hour is therefore the cost of a new scene in a toolkit that already models its materials, not the cost of modeling a new material.

G Agentic Real2Sim2Real: The Complete Pipeline

Figure 15 shows the complete agentic Real2Sim2Real pipeline for the sponge-wiping task of Section 3.3. From a smartphone video (a) of a task whose simulation requires modeling robot motion, object-object contact, and robot-object contact (b), a coding agent constructed and verified the simulation in 56 minutes (c). The resulting multi-physics simulation couples MuJoCo-Warp within



(a) Real-world input video. (b) Task challenges. (c) Agentic Real2Sim construction.

(d) Multi-physics simulation in RoboSimStudio. (e) Physical and spatial variations. (f) Policy improvement with GaP.

(g) Sim2Real policy deployment. (h) Real-world failure from unmodeled physics. (i) Agentic simulation refinement.

Figure 15: **Agentic Real2Sim2Real for sponge wiping.** (a) A real-world bimanual manipulation task, where one arm holds a bar while the other wipes a marker stripe using a sponge. (b) Simulation challenges involving robot motion, object-object contact, and robot-object contact. (c) A coding agent constructs and iteratively refines the simulation through the interactive studio, completing a verified simulated wipe in 56 minutes. (d) A multi-physics simulation in RoboSimStudio. (e) Physical variation of sponge stiffness and spatial variation of the bar and sponge across 30 generated layouts (bottom). (f) GaP [14] improves the simulated wiped area from 0% to 100% over 14 iterations using spatial-temporal simulation feedback from RoboSimStudio. (g) The improved policy is deployed on the real robot to perform sponge wiping. (h) A real-world failure caused by displacement of a bar support that was fixed in the original simulation. (i) Given the observed failure, the coding agent revises the physical simulation to allow the bar and its supports to move under contact. Under an off-center wiping motion, the revised simulation captures bar displacement and tipping.

Newton for the arms, bar, and supports with SuperDex for the deformable sponge (d). RoboSimStudio then varied the sponge's stiffness and generated 30 spatial layouts of the bar and sponge (e), over which GaP [14] improved the simulated wiped area from 0% to 100% in 14 iterations using spatial-temporal feedback from the simulator (f), and the improved policy was deployed on the real robot (g). One real rollout failed when the wiping motion displaced a bar support that was fixed in the simulation (h). Given the failure video and a description of the cause, the coding agent replaced the fixed constraint with a free rigid body for the bar and its supports (i). Neither simulation reproduces the failure under the nominal stroke; under an off-center wipe, found by searching over press depth and lateral offset rather than measured on the real robot, the revised simulation captures bar displacement and tipping while the original keeps the bar in place.

G.1 Real Tasks, Their Simulations, and Event Timing

The sponge-wiping (Section 3.3) and crate-flipping (Appendix H) simulations are agent builds from a single phone video. RoboSimStudio also holds simulations of a full bench of real tasks, built over

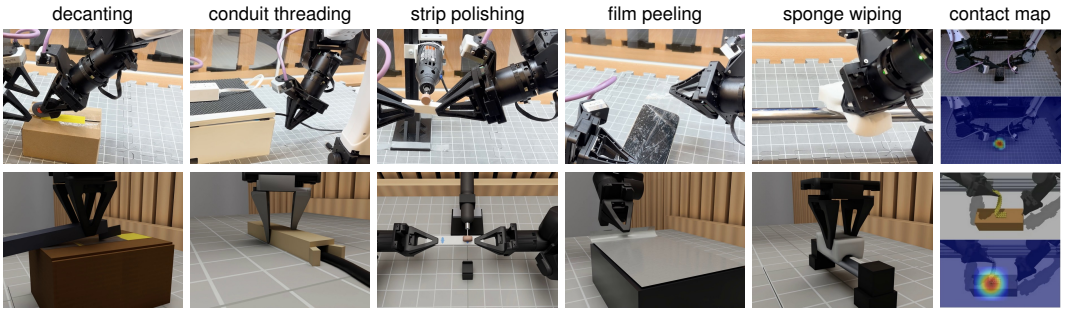


Figure 16: **Real bench tasks and their simulations.** Top: frames of five real tasks on a bimanual YAM workcell. Bottom: the same tasks in RoboSimStudio on a simulation of the bench. Right: for film peeling, the real cell's sensing overlay locates the contact event (top pair), and the simulation draws its own contact readout in the same place (bottom pair). These simulations were built with RoboSimStudio over several sessions; they are not the one-hour agent builds of Section 3.3.

several sessions against a calibrated camera rig (Fig. 16). They share one bench model and one set of cameras, so a simulation is rendered through the camera the real clip was recorded from.

Table 6: **Event timing on sponge wiping**, in seconds from the start of each take. The real clip is recorded at twice real time; its times are doubled here. Simulations A and B are those of Fig. 5b and Fig. 5c.

Event	Real	Sim. A	Sim. B
Sponge gripped	7.4	10.9	15.0
Sponge on the bar	9.6	18.9	28.2
Last stain cleared	24.0	28.9	35.4
Sponge released	26.4	33.8	44.5
Done	30.7	35.9	49.7

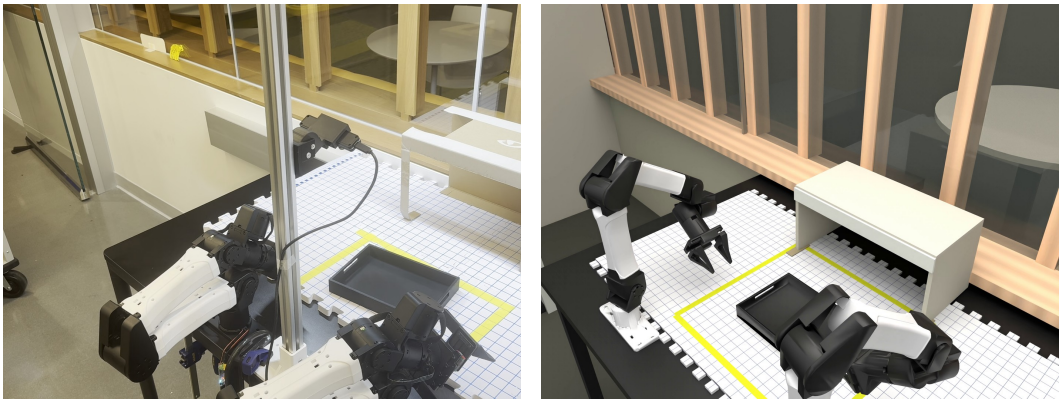
Timing as a first fidelity measure. Side-by-side stills say that a simulation looks right. They do not say that it moves right. Table 6 lines up five events that occur in the real wiping clip and in both agent-built simulations of Fig. 5. Both simulations are slower than the real robot, and not uniformly: they take several times longer to carry the sponge to the bar, and less time than the real robot to wipe it. Aligning a simulation to its reference on events like these, rather than on appearance alone, is a step toward the fidelity metric that Section 4 lists as missing.

H Crate Flipping

Task. Two YAM arms take a shallow plastic crate by its two short end walls, turn it over, and push it into an open box at the back of the bench. Success requires both: a crate that is flipped but left outside the box is only a partial outcome. The crate is a thin shell with a cavity and hand-holes in its end walls. A convex collision approximation would fill the cavity and remove the very rim the grippers pinch, so the simulation must keep the shell geometry.

Setup. The input is a phone recording of a real bimanual workcell (Fig. 17a), with the same interface documentation and a one-sentence description. GPT-6 Astra, running in Codex, built the simulation in about an hour (Fig. 17b).

Learning in simulation. GaP then refined a graph policy for the task inside the simulation. Table 7 reports successes at each iteration on six training configurations and, every other iteration, on twelve held-out configurations. The first three iterations failed on all training configurations. The first successes appeared at iteration 3, and by iteration 6 the graph succeeded on 5 of 6 training and



(a) Real input. (b) Agent-built simulation.

Figure 17: **Crate flipping.** (a) A phone recording of a real bimanual workcell. (b) The simulation GPT-6 Astra built from it, path traced.

10 of 12 held-out configurations, up from 1 of 12 at the start. Early graphs often flipped the crate but stopped short of the box: at iteration 2, three of six rollouts ended this way. Progress was not monotonic: iteration 4 regressed to 0 of 6, and iteration 7 to 4 of 6. Self-learning in simulation therefore needs held-out configurations, and it needs to keep the best graph rather than the last.

Table 7: **Self-learning on crate flipping.** Successes per GaP iteration inside the agent-built simulation, on six training configurations and twelve held-out configurations. A success is a crate flipped and pushed into the box; a flip that stops short of the box counts as a failure.

Iteration	0	1	2	3	4	5	6	7
Training (of 6)	0	0	0	2	0	2	**5**	4
Held-out (of 12)	1	–	0	–	1	–	**10**	–

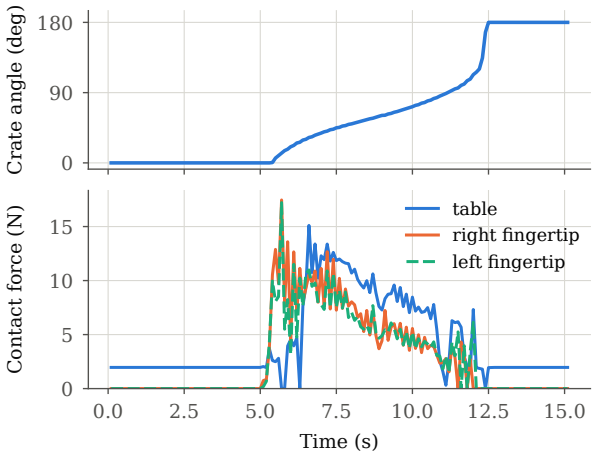


Figure 18: **What the simulation exposes.** One recorded flip in the crate simulation: the crate's angle (top) and the contact force on each fingertip and on the table (bottom). The fingertips load as the arms squeeze and raise the near edge, the table carries the pivot, and all three unload once the crate passes vertical and falls onto its back. None of this can be read from the video the simulation was built from.

What the simulation exposes. A simulation is more than a replay of the motion it was built from: its state can be read and its assumptions can be edited. Figure 18 plots, for one recorded flip, the crate's angle together with the contact force on each fingertip and on the table. The same recording can be replayed under different assumptions. With fingertip friction lowered, the crate never leaves the table. With table friction lowered, it rises to about thirty degrees, slides, and falls back. In the simulation, then, the flip depends on both frictions. The friction values themselves are tuned estimates, not measurements of the real materials; the replay tests whether the motion depends on

them, not what they are. This is the kind of question a learned video model cannot be asked, and it tells a person which contact to measure first on the real cell.