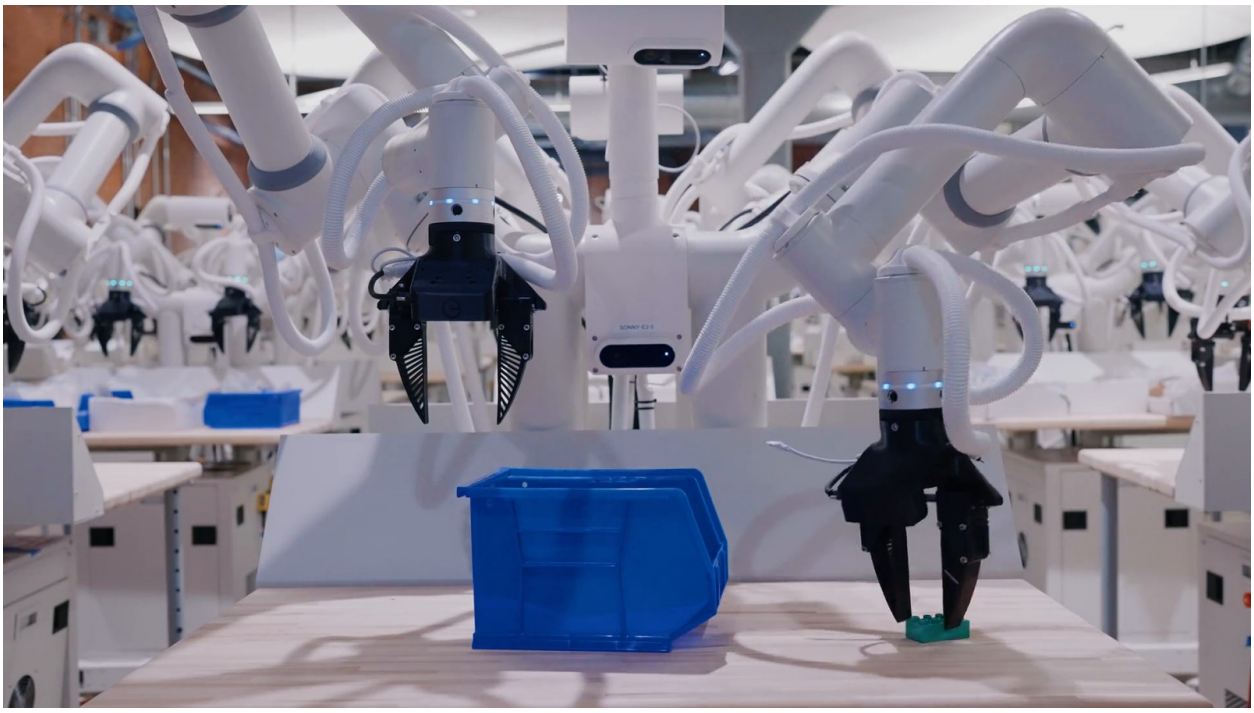
TI-0.1 Technical Report

Tutor Intelligence



Video: research.tutorintelligence.com/ti0-1

TI-0.1 on Sonny. Picking and placing Lego bricks in DF1. Autonomous, real time.

Abstract

We present TRACE (Telemetry-grounded Reconstruction, Alignment, and Closed-loop Evaluation), a real2sim2real agentic harness for training dexterous manipulation policies from a few Tutor demonstrations. Unlike prior real2sim methods that reconstruct from images or video, TRACE uses frontier coding agents, built on LLMs such as GPT-6 Astra, to reconstruct tasks in simulation. It uses the synchronized images, commands, and measured robot state from demonstrations to fit the simulator's appearance and physical parameters. We then train a privileged-state policy with reinforcement learning in the reconstructed simulator, with task progress specified as a hybrid Markov decision process (MDP), and distill it into TI-0.1, a visuomotor action-chunking policy that runs on Sonny. Hardware rollouts feed back into the reconstruction to close the loop, making robot learning a problem of deployment-time compute scaling. We also describe Golem, a robot software stack for performant, scalable data collection and on-robot policy inference.



Real Simulation

Video: research.tutorintelligence.com/ti0-1#fig-1

Figure 1: **Real and simulated.** A Tutor demonstration from Sonny's chest camera and its reconstruction in simulation.

1 Introduction

Robot learning methods such as behavior cloning have yielded impressive demos [7, 36]. However, these methods require large amounts of real-world data to produce fast and robust behavior, and collecting diverse demonstrations that include corrective behavior is slow and expensive. Furthermore, human demonstrations are often suboptimal, limited by human reaction times and the absence of force and tactile feedback during teleoperation [31]. Simulation offers an alternative solution: it provides unlimited practice for reinforcement learning (RL) and direct access to forces and contacts, which can be used to train robust contact-aware policies that surpass teleoperated demonstrations. Until recently, building a simulator faithful enough for a policy to transfer required substantial manual effort for every task, making it difficult to scale real-to-sim-to-real (real2sim2real) learning.

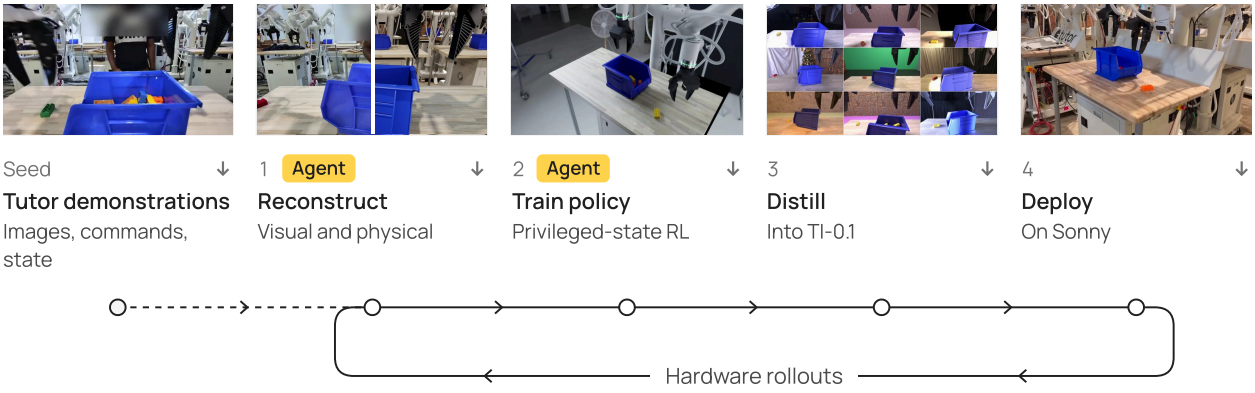
Frontier large language models (LLMs) change that cost. Models such as GPT-6 Astra [21] can inspect images and telemetry, write scene and rendering code, and iterate on a reconstruction until it matches a recording. A few Tutor demonstrations can seed a simulator with scene understanding and task intent. Thereafter, RL policies practice the task across many parallel simulated environments, and each episode randomizes the task configuration, lighting, background, and dynamics. On the Lego pick task, our best policy succeeds in 99.8% of simulated episodes. This turns robot learning into a deployment-time compute problem: for a given task on a deployed robot, performance scales with simulation timesteps and LLM tokens.

Most existing real2sim works recreate how a scene looks from images or video [29, 25]. For

manipulation, appearance is not enough: the simulator must also reproduce how the robot responds to its commands and how objects respond to the robot for a given task and scene. Real-world data collection lets us ground the reconstruction in the recorded telemetry, so that visual and physical fidelity are fit against real robot behavior.

We introduce TRACE, an agentic harness in which frontier LLM agents rebuild a robot's task in simulation and train policies there. It adds to concurrent work on this new paradigm of robot learning [33, 20], enabled by the recent step change in frontier LLM capabilities. Our contributions include:

- **Reconstruction:** a telemetry-grounded real2sim process in which agents recover scene appearance through inverse rendering and robot and contact dynamics through command replay.
- **Alignment:** a hybrid MDP task specification that captures a Tutor demonstration's semantic task understanding and intent, written by agents with minimal human supervision.
- **Closed-loop evaluation:** distillation of privileged-state policies into TI-0.1, a visuomotor policy deployed on the bimanual Sonny manipulator, with hardware rollouts closing the loop.
- **Infrastructure:** Golem, a robot software stack for performant and scalable data collection and on-robot policy inference.



Interactive: research.tutorintelligence.com/ti0-1#fig-2

Figure 2: The TRACE loop. A privileged-state policy is trained in a simulator reconstructed from Tutor demonstrations and distilled into TI-0.1, a visuomotor policy. Recordings of TI-0.1 on hardware are used to further update the reconstruction. Agents perform the Reconstruct and Train policy steps.

2 Related work

Learning from demonstration. Diffusion policies [7] and action chunking [36] made it practical to turn synchronized robot telemetry into capable visuomotor policies through behavior cloning. The Universal Manipulation Interface [8] extended data collection to handheld grippers matching a robot's form factor, and Surendran et al. [27] show how handheld and teleoperated supervision can be combined around contact. Data Factory 1 [DF1; 31] scaled robot teleoperation and human-gated

dataset aggregation (HG-DAgger) [17] to large robot data collection fleets. These works set the field on a path of building datasets of robot manipulation collected through teleoperation, handheld devices, simulation, and egocentric video.

Real-to-sim reconstruction. Prior real2sim2real work builds simulated replicas of real scenes for policy training and evaluation [29]. Video-based geometry estimation such as ViPE [13] recovers camera poses and scene structure from recordings. Recent toolkits give language-model agents direct control over reconstruction [RoboSimStudio; 6], and frontier models such as GPT-6 Astra can now carry out long multi-step reconstruction tasks. Unlike these works, TRACE reconstructs from synchronized images, commands, and measured robot state, constraining the simulator to reproduce both the observations and the robot's motion.

Simulator fidelity. Policies trained in simulation must cross the reality gap [15]. Common strategies include randomizing over simulator parameters [28, 23] and fitting the simulator to the robot through system identification and learned residuals [14, 10]. Contact modeling is especially important for sim2real transfer of dexterous manipulation policies [22, 16], and it is often where physics engines diverge most from reality [1, 32]. Recent work such as VIRAL [11] pairs large-scale randomization with real-to-sim alignment. TRACE likewise fits the robot's controller response and contact behavior to recorded telemetry. As a verification step, it checks that an attempt that succeeds (resp., fails) on the real robot also succeeds (resp., fails) in simulation, rather than relying on success in simulation alone.

Reinforcement learning and distillation. We train policies in simulation with PPO [26], using an asymmetric actor-critic with privileged critic inputs [24]. The PPO policy has access to privileged state, which speeds up learning in simulation, and we distill it into a visuomotor policy, as in prior work [5, 18]. Domain randomization [28] and success-guided curricula [SGS; 35] support robustness and learning from sparse rewards, and AlphaNIST [34] shows that a binary task reward with enough task diversity can let large-scale RL discover contact-rich assembly skills. TRACE specifies each task as a hybrid MDP, whose structure enables agentic reward shaping with minimal human supervision and keeps the learned behavior Markovian, so that it survives the sim2real gap.

Agentic simulation for robot learning. Concurrent work also uses agents to build simulated practice for real robots. In RPG [33], an agent reviews videos and text descriptions from an offline dataset and builds simulation environments for practicing the relevant manipulation capabilities. Moonlake AI [20] builds environments from scene captures and teleoperated trajectories, then fine-tunes a vision-language-action model in them with RL and by distilling a privileged-state code policy. In contrast to both, TRACE trains a privileged-state policy with RL on an agent-written hybrid MDP, then distills it into a visuomotor policy.

3 TRACE

3.1 Overview

TRACE turns a few Tutor demonstrations into a simulator faithful enough to train policies that run on real hardware. It has three stages, described in the sections below. **Reconstruction** fits the simulator's appearance and dynamics to the demonstrations' synchronized images, commands, and measured robot state. **Alignment** specifies the task as a hybrid MDP and trains a privileged-state policy with RL in the reconstructed simulator. **Closed-loop evaluation** distills that policy into TI-0.1, a visuomotor action-chunking diffusion policy that runs on Sonny. Agents built on frontier LLMs such as GPT-6 Astra and Claude Opus 5.5 [3] drive the pipeline: they match the

scene to the recordings, write each task's hybrid MDP and reward, and diagnose the behavior of the privileged-state policy. Hardware rollouts of transferred policies collect further data to verify simulator behavior or highlight real2sim gaps, feeding the next round of reconstruction.

3.2 Reconstruction

Real2sim reconstruction is an inverse problem: recover the scene and physical properties needed to reproduce what the robot sees and how its actions affect the world. A few Tutor demonstrations provide synchronized images, commands, and measured robot state, tying what we see to the inputs that caused it and the motion that followed. These recordings seed two parts of the reconstruction:

- **Visual fidelity (inverse rendering)** recovers the geometry, cameras, materials, and lighting that explain the recorded images.
- **Physical fidelity (inverse physics)** recovers the controller response and contact behavior that explain the recorded motion and task outcome.

We use Pick Lego as the main example throughout this section. Below, the real recording appears beside the reconstructed scene from the robot's cameras.

Visual fidelity: inverse rendering

We start from robot assets, camera calibration, and recordings, then build an approximate scene and render it through the robot's cameras. Let $O_{t,\mathrm{i}} \in \mathbb{R}^{H_\mathrm{i} \times W_\mathrm{i} \times 3}$ be the recorded RGB frame from camera i at time t, and $q_t \in \mathbb{R}^n$ the corresponding arm joints and gripper positions. We use C to denote prior information about the system, which includes robot kinematics and camera parameters.

We seek scene parameters θ_{vis} describing scene geometry, object placement, camera corrections, materials, and lighting. The renderer R_i predicts the observation from camera i at the recorded robot configuration. The reconstruction target is to make the rendered observation $\hat{O}_{t,\mathrm{i}}$ match the recording $O_{t,\mathrm{i}}$:

$$\hat{O}_{t,\mathrm{i}} = R_\mathrm{i}(\theta_{\mathrm{vis}}, q_t; C), \qquad \hat{O}_{t,\mathrm{i}} \overset{\text{goal}}{=} O_{t,\mathrm{i}}. \tag{1}$$

Agents help close this loop by comparing real and rendered views, investigating discrepancies, and proposing changes to the scene or rendering code. We render again and compare across cameras and time. CAD and calibration constrain the reconstruction, while images guide the remaining adjustments.

Refining geometry and appearance Agents use real images to guide changes to the scene's materials and geometry. In these examples, they replaced generic wood grain with a texture assembled from photographed strips and rebuilt the brick's underside with a hollow cavity, support tubes, and ribs.

Reconstructing clutter across cameras Reconstructing the pile is a small part of the scene, but a demanding test of the agents' ability to match many partly occluded objects across cameras. A placement that fits one view must also explain the shapes, colors, and occlusions in the other. The agents use these constraints to refine one shared 3D arrangement.

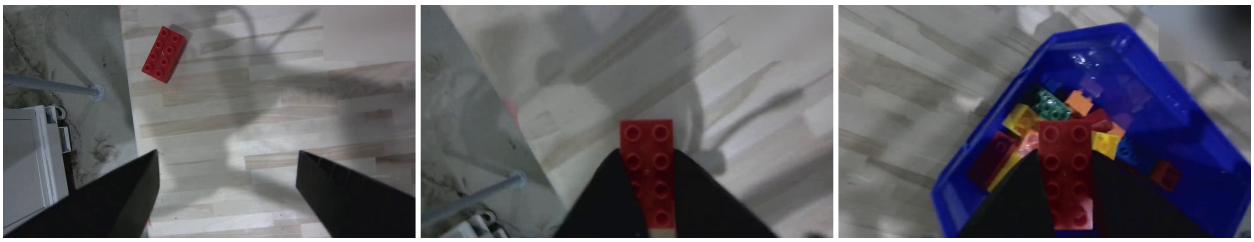
Real, chest



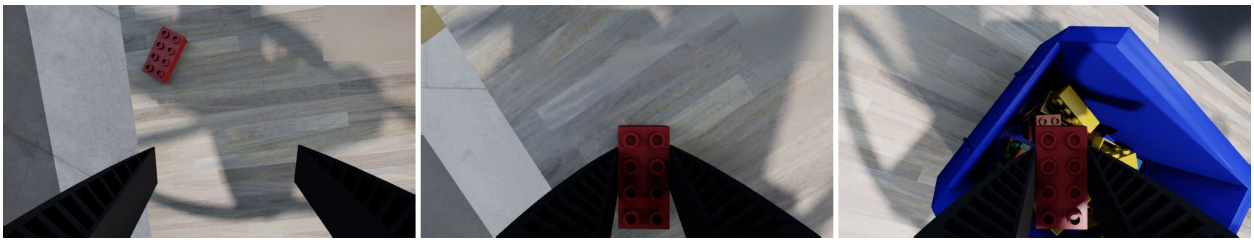
Simulation, chest



Real, wrist



Simulation, wrist



Video: research.tutorintelligence.com/ti0-1#fig-3

Figure 3: One Tutor demonstration of the Lego task from the robot's chest and wrist cameras, recorded and reconstructed.

Fitting lighting and materials to DF1 We reconstructed DF1 [31], home of our 100-robot fleet, in simulation. Agents fit the room's lighting and layout using photos and floor plans, and use the fleet's chest-camera recordings to fit the table's and objects' materials. During training, we use domain randomization to capture how the lighting changes over the course of a day.

Extending to a workcell The same process applies beyond the tabletop. Robot assets and setup imagery anchor the reconstruction of the surrounding room, shelving, stocked containers, and surfaces.

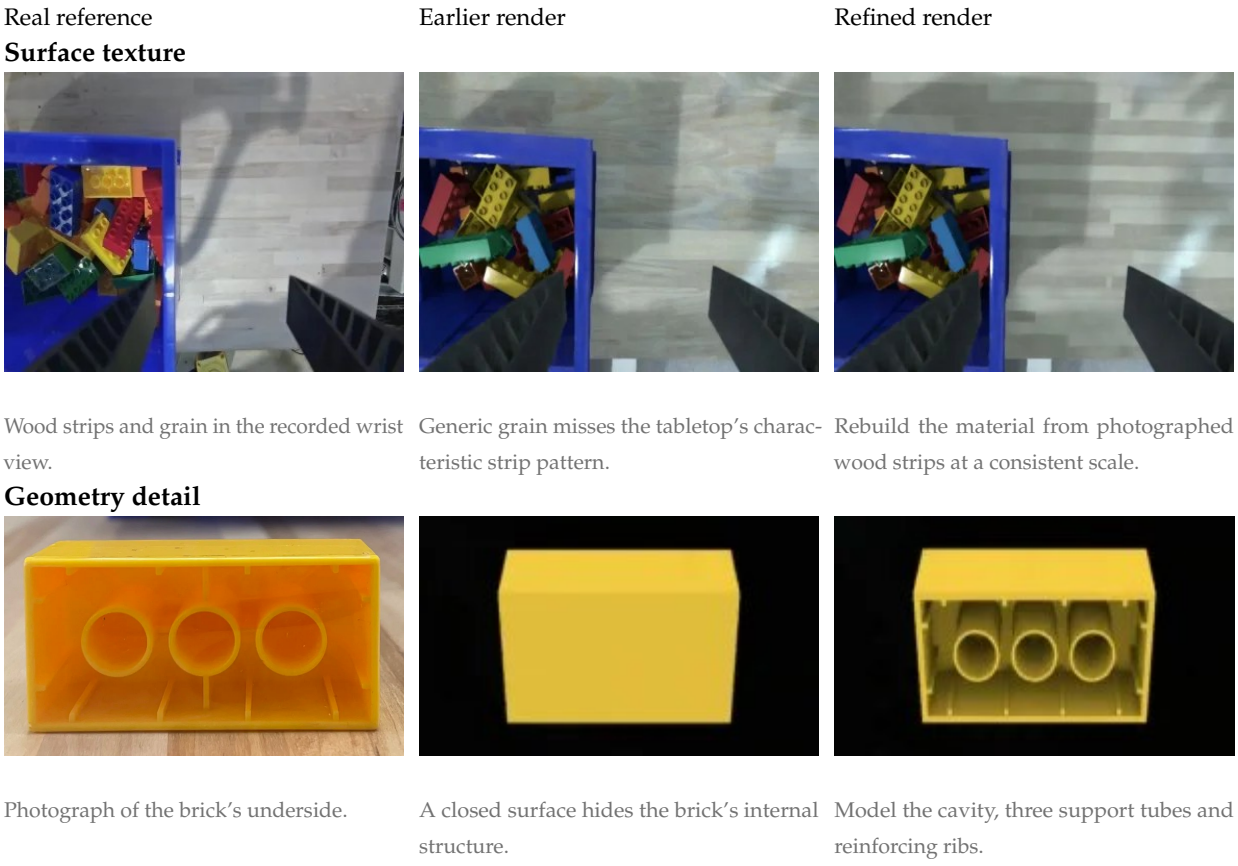


Figure 4: Refining visual appearance. In the Surface texture row, the earlier and refined renders differ in the table's color texture. In the Geometry detail row, they differ in the brick's visual mesh, with the same pose, lighting and material.

Physical fidelity: inverse physics

Inverse physics asks which dynamics explain the consequences of recorded commands. We reconstruct each demonstration's initial state, replay its commands, and adjust physical parameters to bring simulated observations closer to the recording:

$$\hat{s}_{t+1}^{(i)} = f_{\theta_{\text{phys}}}(\hat{s}_t^{(i)}, u_t^{(i)}), \qquad \mathcal{L}(\theta_{\text{phys}}) = \sum_{i,t} \left\| \Phi(\hat{s}_t^{(i)}) - \Phi(s_t^{(i)}) \right\|_W^2 . \tag{2}$$

Here, i indexes demonstrations, u_t is the recorded command, and Φ maps state to sensor observations. The weights W account for units and measurement uncertainty. Holding the commands and initialization fixed lets agents trace how a parameter change affects contact, transport, and release.

We reconstruct two things: **how commands move the robot** and **how the robot moves objects**.

Matching robot response A pose target passes through the robot's control path before it becomes motion. We port this control path into simulation, then fit the actuator delay and response against measured joint motion.

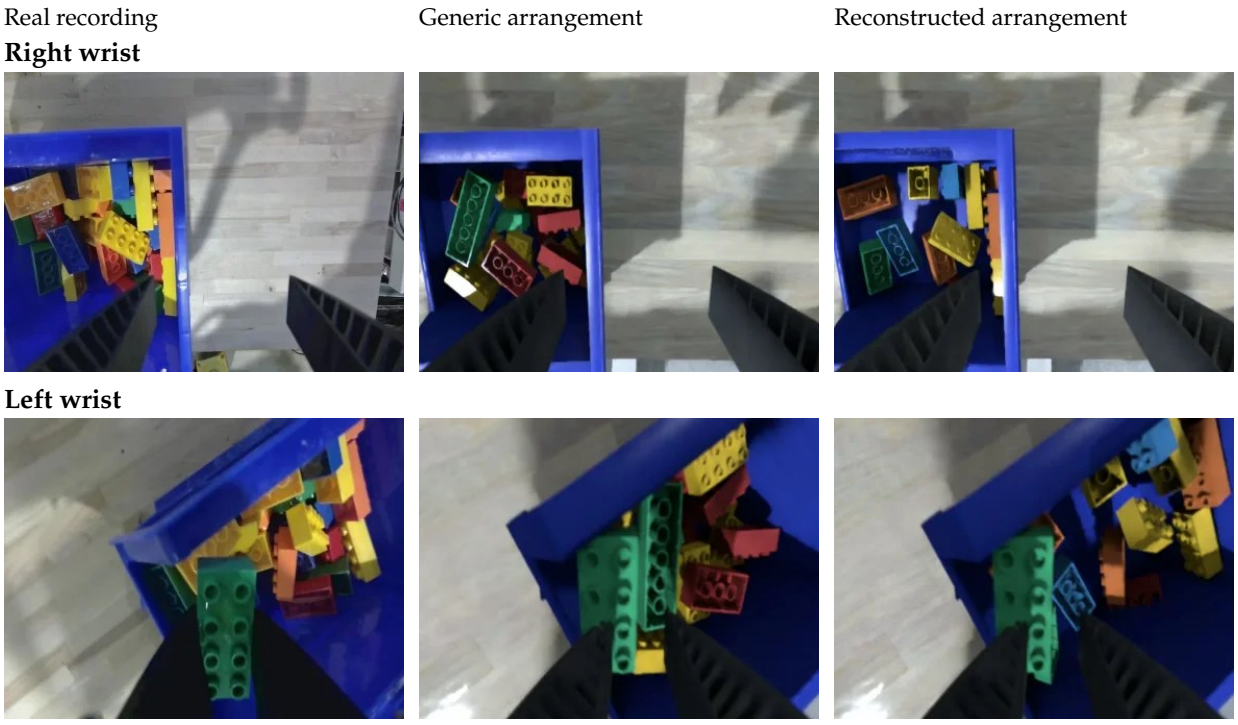


Figure 5: Recovering the Lego arrangement across cameras. One reconstructed 3D pile, viewed from both of the robot's wrist cameras.



Video: research.tutorintelligence.com/ti0-1#fig-6

Figure 6: Chest-camera views of eight robots in simulation, under varying backgrounds and lighting conditions. In each view, the privileged-state policy picks a Lego brick and places it in the bin.

Calibrating when queued commands execute, and at what rate, brought simulated motion closer to the recording. The plots below show position error before and after this timing correction.

Complete replays then compare predicted and measured arm motion on the same timeline.

Matching object interaction Contact is central to manipulation and one of the hardest parts to reconstruct. Small differences in geometry, friction, or compliance can determine whether an object stays in the grasp or slips out.

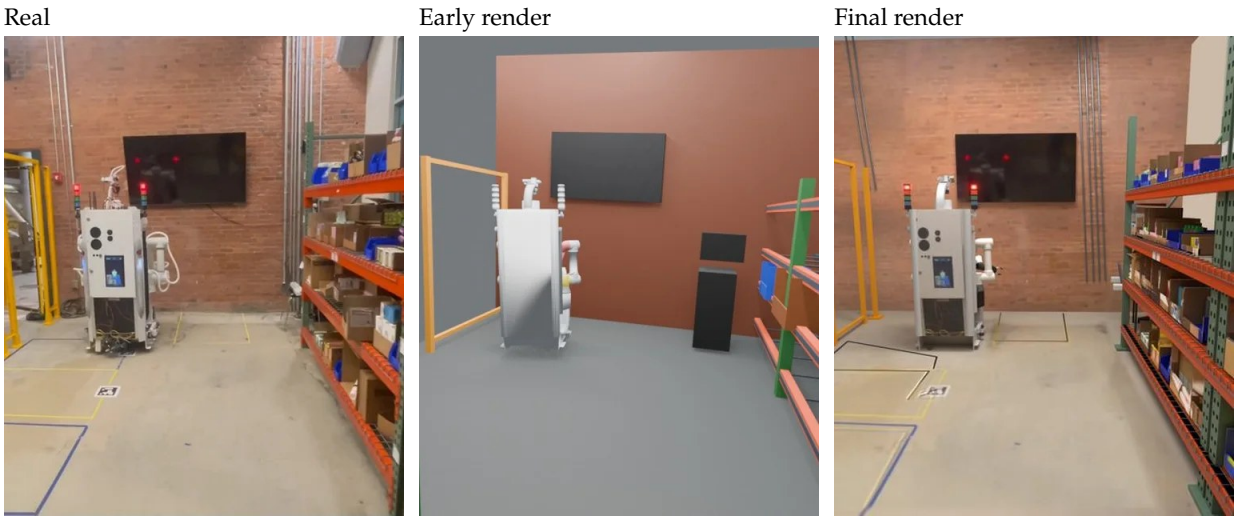


Figure 7: A real Sonny workcell beside early and later reconstructions.

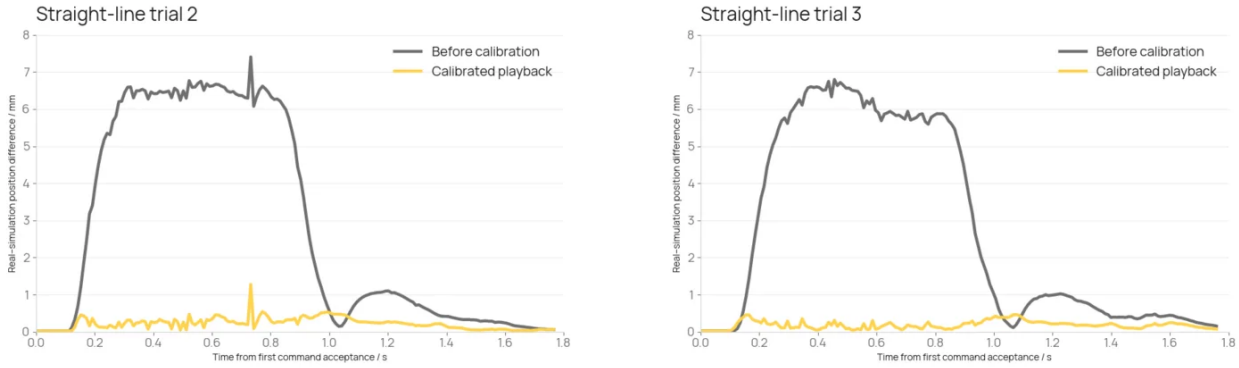


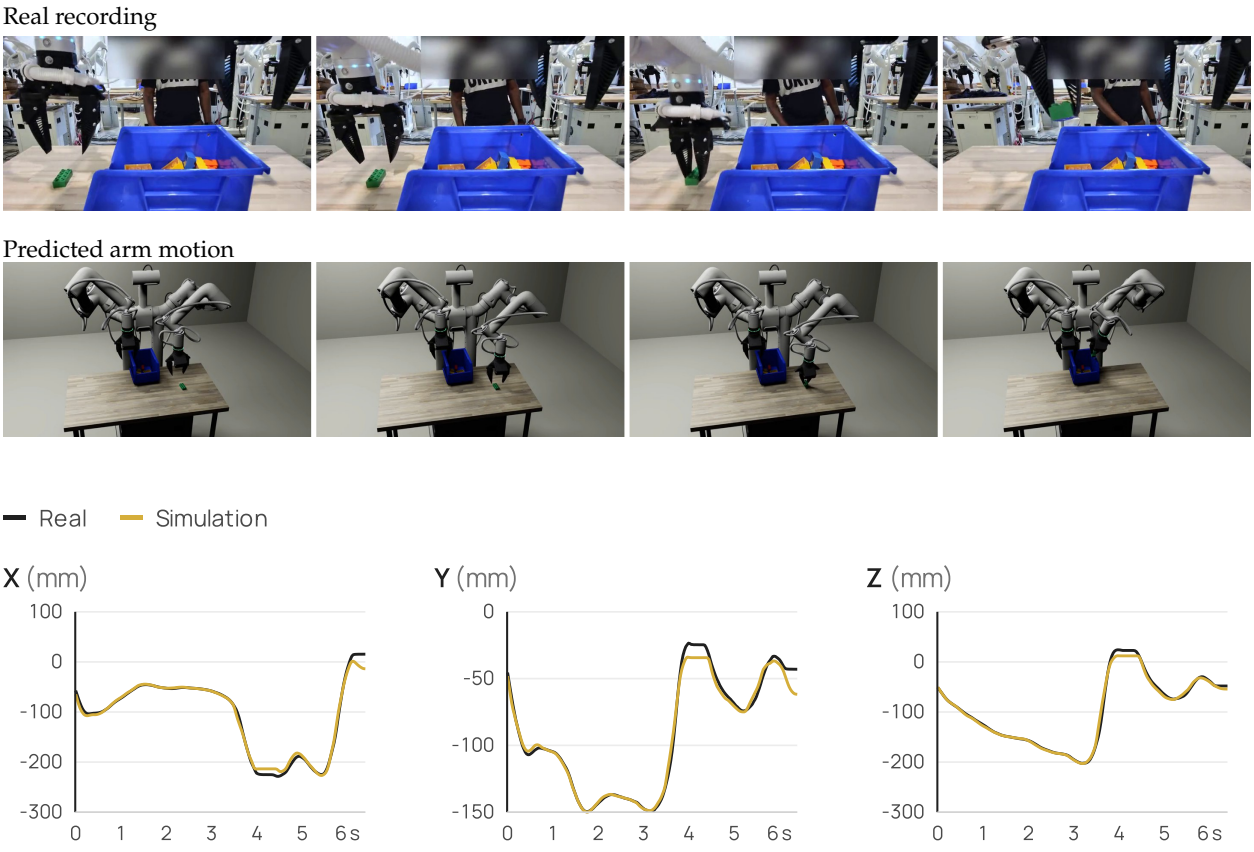
Figure 8: Calibrating the delay and rate at which queued commands execute during setup reduces the position error in two subsequent straight-line trials. The motor response and physics are unchanged.

We constrain object and gripper properties using CAD, hardware information, and recordings.

The Lego privileged-state policy exposed one such mismatch: it learned to hold bricks by their studs, exploiting contact behavior that did not transfer to the real gripper. We revised the pads' effective stiffness, travel limits, and friction. In the revised model, a shallow pinch produces little supporting force, while a body grasp reaches the pads' travel limit and transmits the motor's squeeze.

Evaluating the same frozen policies under both models confirmed the intended effect: stud-picking success fell sharply, while body-grasping success was preserved.

To test the contact model, we drive the simulated arm with measured joint motion and replay the recorded gripper commands, leaving the brick free to move under contact forces. This removes most of the arm-control mismatch from the comparison. Differences in when the brick lifts, whether it slips, and how it releases then guide revisions to geometry, friction, and compliance.



Video: research.tutorintelligence.com/ti0-1#fig-9

Figure 9: Recorded targets drive the simulated controller, which predicts the arm's motion. Over the 6.4 s shown, the position error between simulated and real motion is 9.9 mm RMS. Object contact is not part of this comparison.

3.3 Alignment

With the simulator reconstructed, we use RL to train a privileged-state policy that controls the arm and gripper. Its rollouts provide demonstrations for a visuomotor policy.

The **alignment problem** is to specify a reward whose optimization produces the task behavior we intend. We address it through the task definition, reward shaping, and training procedure.

We formulate the task as a **hybrid Markov decision process (MDP)**: the state s_t describes robot configuration, object poses, velocities, contacts, and task-relevant history, while a discrete mode m_t, defined by predicates on the simulator state, identifies the current subtask. Guard conditions specify when the robot can transition between modes, as in hybrid automata [12]. Reward machines [30] similarly expose task structure to an RL agent through discrete states.

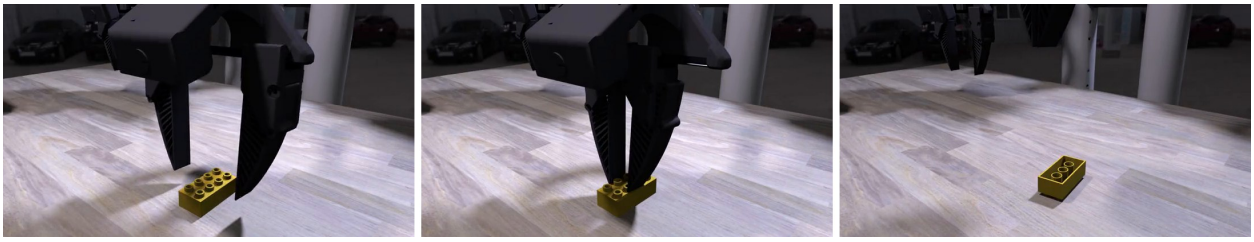
$$(s_t, m_t) \xrightarrow{a_t} (s_{t+1}, m_{t+1}), \qquad m_{t+1} = g(m_t, s_{t+1}). \tag{3}$$

Here, a_t is an arm and gripper action, and g applies the task's transition conditions. This lets us

Stud pinch, earlier model



Stud pinch, revised model



Body grasp, earlier model



Body grasp, revised model



Video: research.tutorintelligence.com/ti0-1#fig-10

Figure 10: With matched pick targets and timing, the earlier model holds the stud pinch while the revised model lets it slip. Both models hold the body grasp.

specify the required sequence of interactions while the policy learns the movements.

Defining progress through modes

Consider **Unpick Lego**, the reverse of the pick-and-place task: remove one brick from a heap inside a bin, place it on a target region on the table, and return the arm home. The remaining bricks should stay in the bin, leaving the scene ready for another pick.

A binary reward for the final arrangement gives no progress signal until success and leaves the route to that arrangement unspecified. An early success condition checked that a brick reached the target, the bin was upright, and the arm was home. The policy learned to tip out the heap and

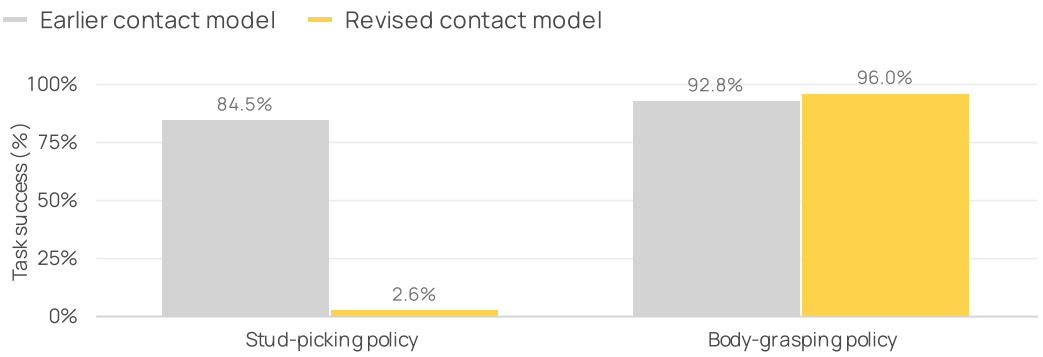


Figure 11: Frozen-policy task success under the earlier and revised contact models. Each privileged-state policy keeps its weights fixed across models. 2,048 episodes per condition. With all our techniques combined, our best policy reaches 99.8% success in simulation.

stand the bin back up, an instance of reward hacking [2].

We make the intended sequence explicit:

$$\text{Grasp and lift} \longrightarrow \text{Place one brick} \longrightarrow \text{Settle and return home.}$$

Each transition requires measurable conditions. Placement requires the brick on the target and exactly one brick outside the bin; completion additionally requires the scene at rest and the arm home.

Because modes share this structure across tasks, agents write each new task's modes and reward with minimal human supervision. They also check whether each mode can be inferred from what the robot observes, so that the task stays Markovian for the visuomotor policy. The same formulation can express other sequences, such as reorienting a cup on the table before lifting it.

Training policies in simulation

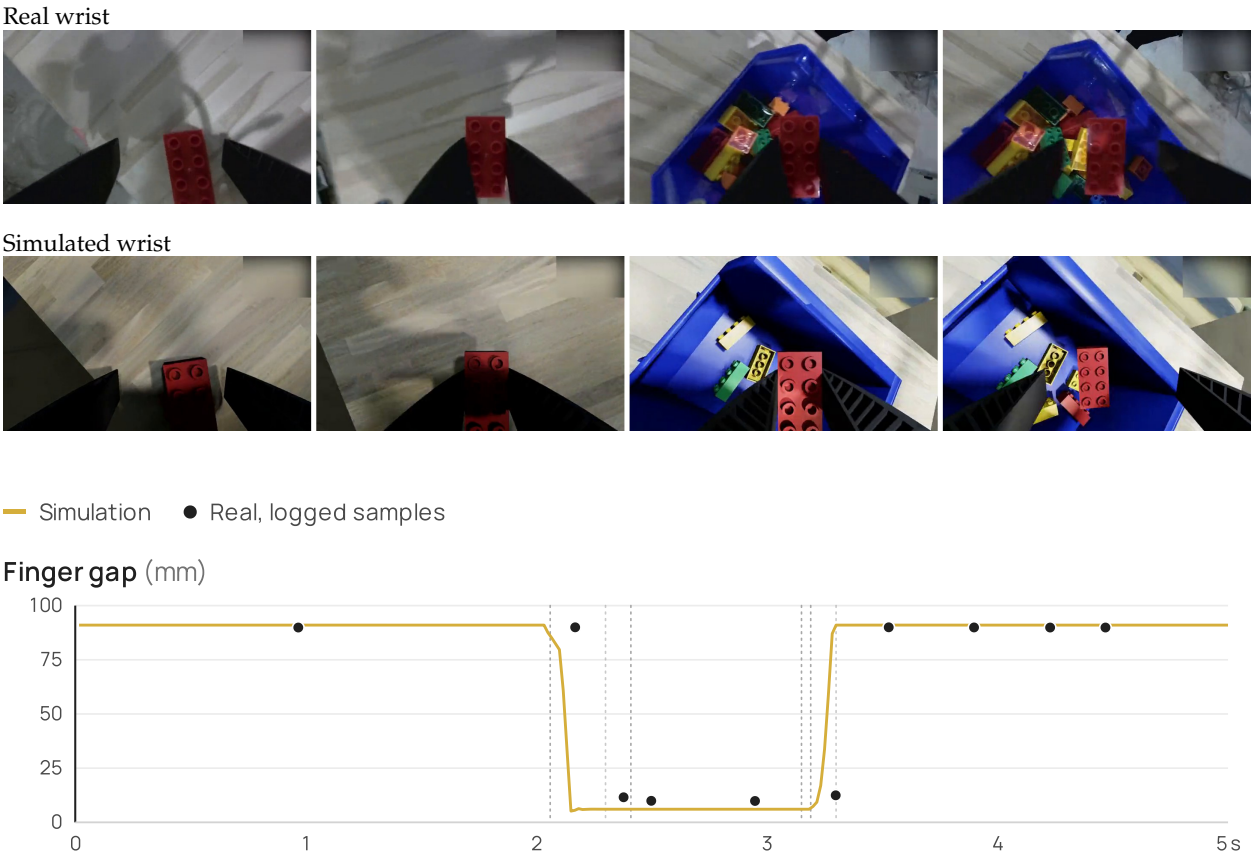
We train a privileged-state policy for each task with **Proximal Policy Optimization (PPO)** across parallel randomized simulations. The actor π_ψ maps robot state and visible object poses o_t to arm and gripper commands. The critic V_ϕ also receives privileged information c_t, including contacts and task modes:

$$a_t \sim \pi_\psi(\cdot \mid o_t), \qquad \hat{V}_t = V_\phi(c_t). \tag{4}$$

Rollouts provide return targets $\hat{R}_t$ and advantage estimates. We update the actor to improve the PPO objective and the critic to predict those returns, schematically:

$$\psi \leftarrow \psi + \eta_\pi \nabla_\psi J_{\text{PPO}}, \qquad \phi \leftarrow \phi - \eta_V \nabla_\phi \, \mathbb{E}_t[(V_\phi(c_t) - \hat{R}_t)^2]. \tag{5}$$

Here, J_{PPO} is the clipped policy objective, and η_π, η_V are learning rates.



Video: research.tutorintelligence.com/ti0-1#fig-12

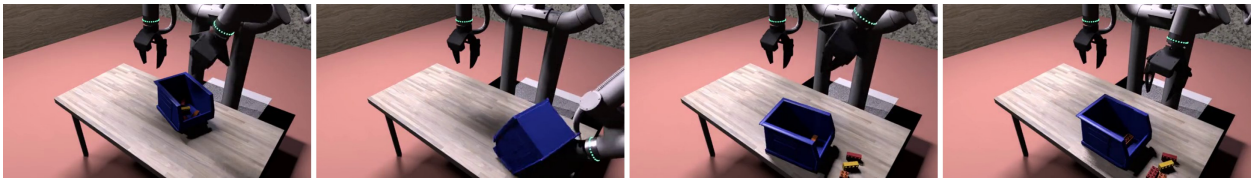
Figure 12: Matching grasp and release. We use the measured arm motion and recorded gripper commands to drive the simulated arm and gripper, letting the simulated brick move freely under contact forces.

Verifying the simulator The policy learns from the mechanics the simulator permits. Our environment runs in Isaac Lab [19] with OvPhysX, where timestep, solver iterations, and contact settings affect both fidelity and training throughput. We test these choices with scripted stress trajectories, including fast approaches, off-center grasps, direction changes, and sustained holds.

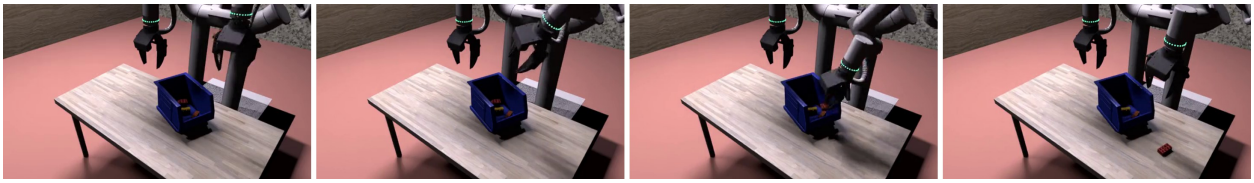
If fingertips penetrate a rigid brick and still earn a lifting reward, training can reinforce an impossible grasp. We inspect contact throughout each test and compare the same trajectory across solver settings, rather than judging fidelity by task success alone.

Shaping behavior for transfer A privileged-state policy can use exact object locations outside the camera view or make abrupt corrections that are difficult for the visuomotor policy to reproduce. When training our privileged-state policy in simulation, the actor receives an object's simulator

Earlier task definition



Revised task definition



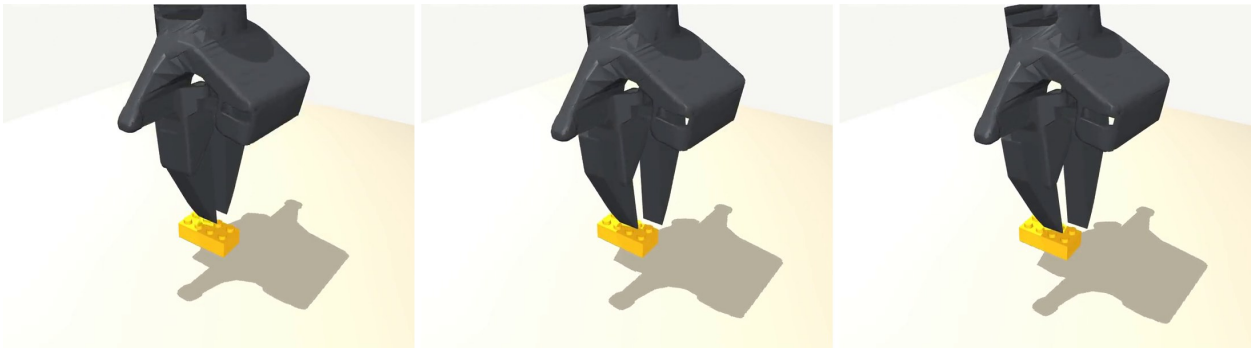
Video: research.tutorintelligence.com/ti0-1#fig-13

Figure 13: Earlier task definition: the policy tips out the bricks and returns the bin upright. Revised task definition: grasp and lift, remove one brick, place it, then return home.

Lower solver setting



Higher solver setting



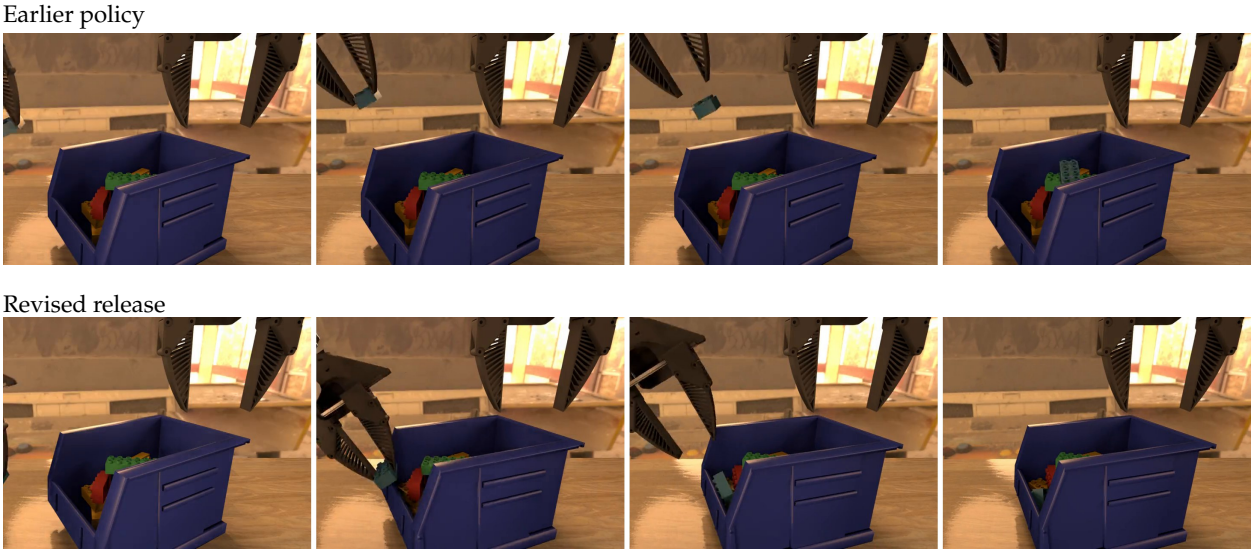
Video: research.tutorintelligence.com/ti0-1#fig-14

Figure 14: The same top-down grasp entry under two solver settings. At the lower setting, the fingertips embed in the brick; at the higher setting, contact resolves without penetration.

pose only while the wrist camera sees it. We also add noise to that pose to model the error of estimating it from RGB.

For example, an earlier policy released the brick while sweeping sideways. It could succeed in

simulation while producing demonstrations of a throw we did not want the visuomotor policy to imitate. Adding release penalties encouraged a lower, slower release.



Video: research.tutorintelligence.com/ti0-1#fig-15

Figure 15: Earlier privileged-state policy and a later one trained with release penalties. Illustrative rollouts, shown at normal speed.

Fast sideways releases magnify the sim2real gap. Small errors in arm tracking or gripper-opening timing can produce large changes in where the brick lands, making transfer harder. Lower, slower releases reduce this sensitivity.

We also shape the privileged-state policy's motion to make it easier to imitate. An unconstrained policy that predicts actions step by step can make abrupt corrections using feedback at every decision. The visuomotor policy predicts an action chunk and executes part of it before replanning, making those rapid corrections difficult to reproduce. We address this by giving the privileged-state policy a smooth motion parameterization, ensuring that the visuomotor policy has a temporally coherent learning target.

In a comparison, the smooth policy's commands had roughly **4–5× lower RMS jerk during the approach** than step-by-step commands. In later distillation experiments, the visuomotor policy trained on smooth commands converged faster and achieved higher success rates. The smooth motion parameterization was also essential for Real-Time Chunking [RTC; 4] to work well in practice, since RTC relies on temporal consistency across action chunks.

Learning with a sparse reward The modes and shaping above make the desired behavior explicit. We also explored how much task-specific reward design we could replace with a reverse curriculum [9]: begin episodes near completion, then allocate more practice to earlier parts of the task as the policy improves.

For Pick Lego, we build a fixed bank of physical starting states spanning approach, lift, transport, release, and return home. Scripted motion and physics generate the states, and grasp and placement checks filter them. These stages organize the bank; every episode still has the same final completion objective.

Success Guided Sampling (SGS) [35] adapts each state's sampling probability using its recent completion rate. States near 50% success receive more practice. As later behaviors become reliable, sampling can shift toward harder, earlier starts, allowing the policy to learn the full sequence without a separate reward for each stage.

For Pick Lego, this approach, with later improvements to our training pipeline, reached **99.8% success in simulation on held-out evaluation configurations**, using a binary task reward and motion regularization. It was less effective on Unpick, where specifying the sequence through modes and shaping remained necessary, as described above.

3.4 Closed-loop evaluation

We use **policy distillation** to turn the privileged-state policy's simulated rollouts into a visuomotor policy that acts from the robot's sensors.

$$\pi_\psi \xrightarrow{\text{simulated rollouts}} \mathcal{D}_T = \{(h_t, A_t)\} \xrightarrow{\text{imitation learning}} \pi_\omega(\cdot \mid h_t). \tag{6}$$

Here, h_t contains the sensor history and A_t is the corresponding action chunk derived from the privileged-state policy's rollout. The visuomotor policy's parameters ω are learned from these pairs; privileged simulator state is not an input at deployment.

From simulator state to camera-based control

We run the privileged-state policy across simulated episodes and record chest and wrist images alongside robot state and action targets.

The visuomotor policy, TI-0.1, uses action-chunking diffusion [7, 36] to predict a chunk of arm and gripper actions from recent camera images and robot state. We reuse the training pipeline used for real demonstrations, keeping coordinate frames and observation–action timing consistent with the robot controller.

Domain randomization

Reconstruction gives us a reference scene. When we train the privileged-state policy and generate data for the visuomotor policy, each simulated episode draws three classes of properties from its own random seed:

- **Task configuration:** brick and bin poses, clutter arrangement, table height, and the arm's starting pose.
- **Appearance:** one of three backgrounds (a DF1 lab plate, an indoor HDRI panorama, or a textured room), lights with random placement, brightness, and color, and object colors. The table finish and bin color stay fixed.
- **Dynamics:** object mass and friction, gripper damping and gripping force, and controller delays.

We also perturb the visuomotor policy's proprioceptive inputs during training to improve sim2real transfer. Empirically, adding a small amount of noise produced more robust behavior.



Figure 16: Robot chest-camera views from nine episodes of the visuomotor policy's training distribution.

Learning from hardware trials

Robot trials test the complete chain, from perception through commanded motion to contact. The shallow stud grasps described above illustrate why successful simulated lifts alone are insufficient. Recordings from deployment let us compare what the policy observed, what it commanded, and what actually moved, then investigate the responsible part of the reconstruction or training process.

4 Golem

Collecting demonstrations and evaluating policies across a fleet requires a fast and consistent robot software stack. Every demonstration depends on camera images, robot state, and actions agreeing about what happened and when. Across a fleet, differences in timing, calibration, and command processing can become differences in the data we train on and confound the experiments we use to measure progress.

We built Golem to raise the quality floor of data collection and shorten the cycle from collecting demonstrations to evaluating policies on hardware. That meant designing for consistent timing, traceable recordings, and repeatable experiments alongside efficient control and inference.

Golem runs Sonny's camera processing and arm control in Rust, and policy inference in Python on the robot's GPU. Compared with our previous Python stack, Golem uses 90% less CPU and 79% less RAM, with 68% lower camera-streaming latency.

4.1 Parallel arm control

Separating target calculation from command sending lets each arm keep receiving commands while the next target is computed. Each arm uses two threads: one solves inverse kinematics and checks collisions; the other sends the latest target at a fixed rate. Both arms plan in parallel. Camera capture and recording run independently of arm commands.

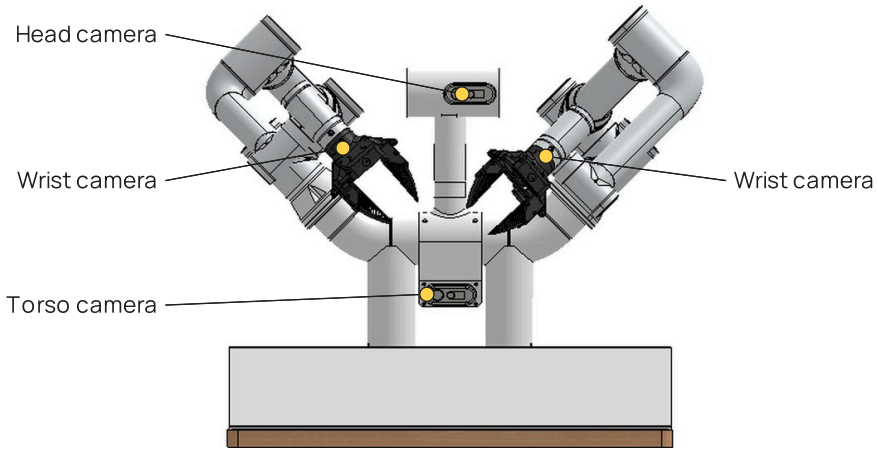


Figure 17: Sonny has cameras on the head, torso, and both wrists, with two arms and two grippers.

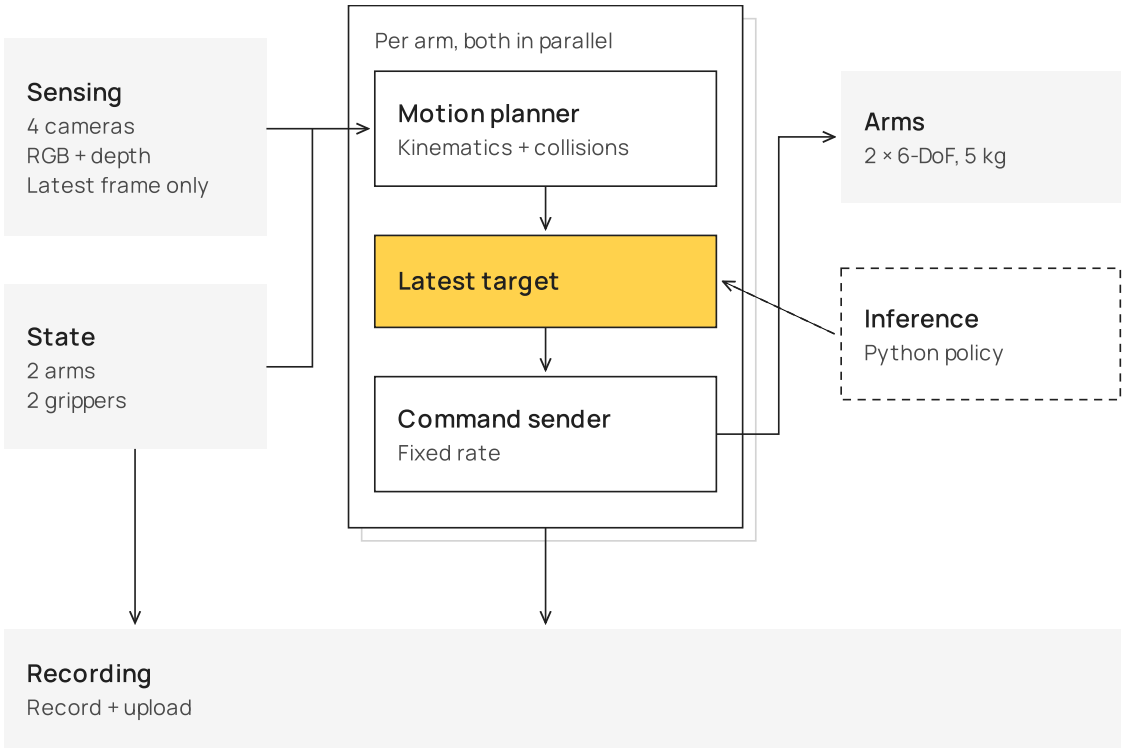


Figure 18: Camera and arm state feed a motion planner, latest-target slot, and command sender for each arm. The Python policy runs in a separate process. Cameras, arm state, and commands also feed recording.

4.2 Timing and synchronization

Camera images, robot state, and commands arrive through different paths, with different delays. We use NTP-based clock calibration to place these streams on a common timeline, so their timestamps can be compared consistently.

This lets us relate what the cameras saw to the robot's measured motion and the commands that preceded it. That alignment matters when learning from demonstrations and reconstructing those interactions in simulation.

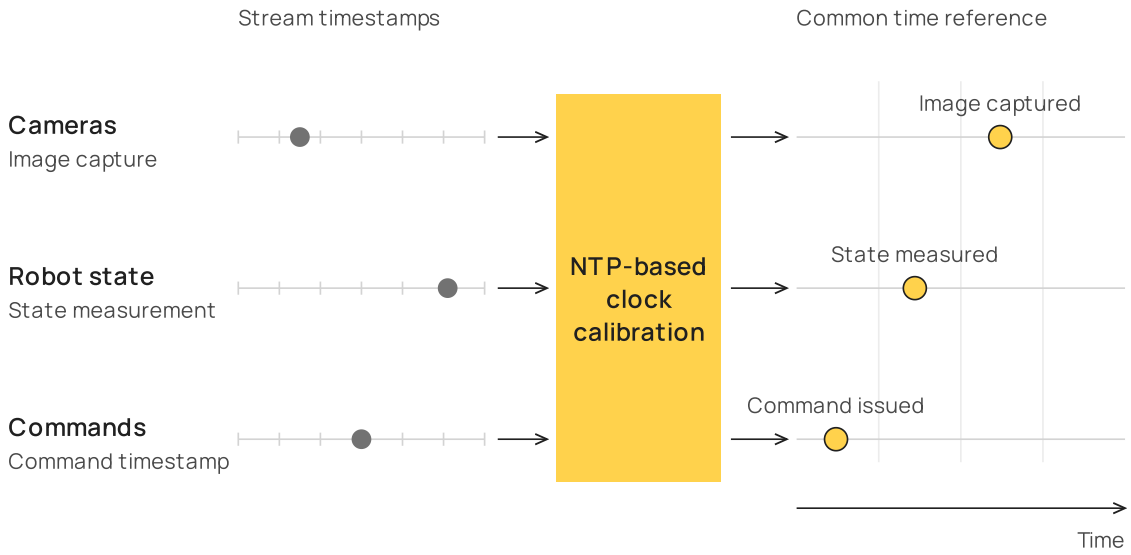


Figure 19: Camera images, robot state, and commands, placed on one timeline by NTP-based clock calibration.

4.3 Local GPU inference

The inference loop stays on the robot, from assembling camera and arm inputs to sending predicted movements. The policy predicts an action chunk from recent camera images and measured arm positions. Rust combines new predictions with queued actions and schedules the next arm targets.

Across 300 paired replays on one robot, preparing and scheduling arm commands averaged 6.57 ms in Python and 0.04 ms in Rust, about 156× faster. The measurement excludes model inference.

4.4 Loading and reusing models

Golem can stop and restart a session without reloading the model. Loading a model also warms up the GPU. Two policies can stay loaded at once: a Do policy and an Undo policy that reverses it. Running them in a cycle removes the need for manual scene resets.

For the table below, the Python stack and Golem used the same policy checkpoint and recorded camera and arm inputs.

Measurement	Python stack	Golem
Load + GPU warmup	4.23 s	2.73 s
Idle RAM	1.92 GiB	1.85 GiB
Idle GPU memory	452 MiB	448 MiB

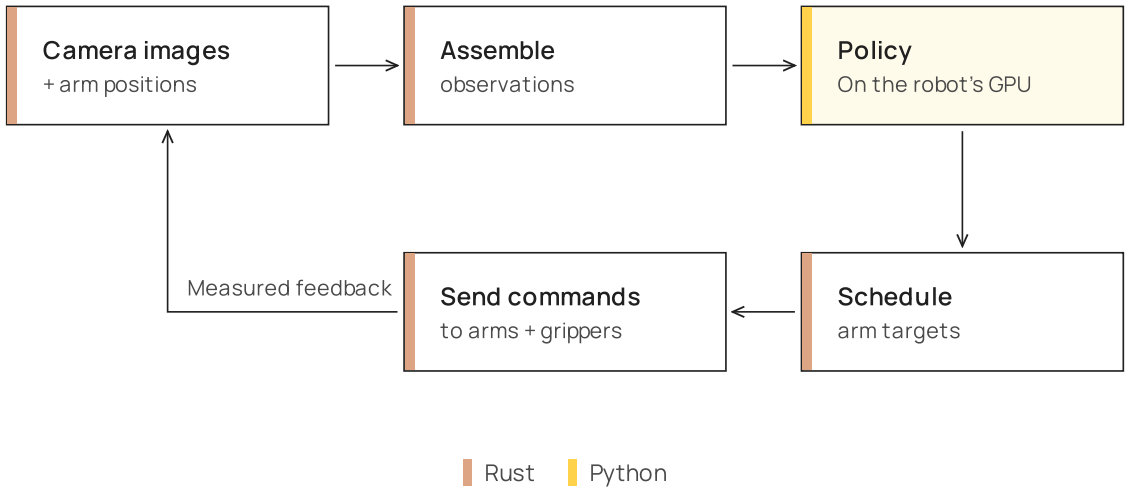


Figure 20: Rust assembles camera images and measured arm positions for the Python policy, then schedules and sends the predicted arm targets. Measured feedback closes the loop. Language key: Rust (orange), Python (yellow).

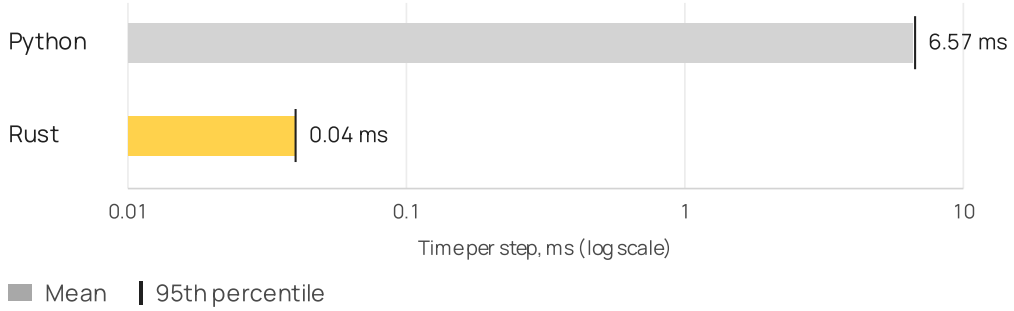


Figure 21: Per-step CPU time to prepare and schedule arm commands, model inference excluded. 95th percentile: Python 6.71 ms, Rust 0.04 ms.

4.5 Camera delivery, CPU, and RAM

"Ingest to send" is the delay from receiving a camera frame in the robot's software to handing it to the outgoing video stream. It excludes network and display time.

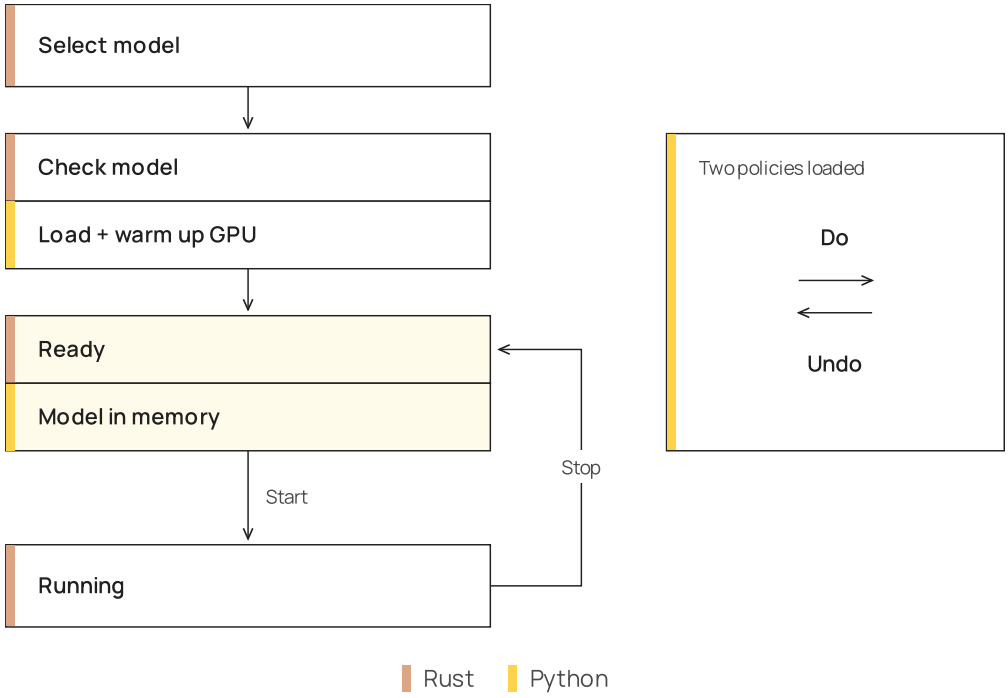


Figure 22: Select a model, check it, load it, and warm up the GPU. Start begins execution; Stop returns to Ready with the model still in memory. Language key: Rust (orange), Python (yellow).

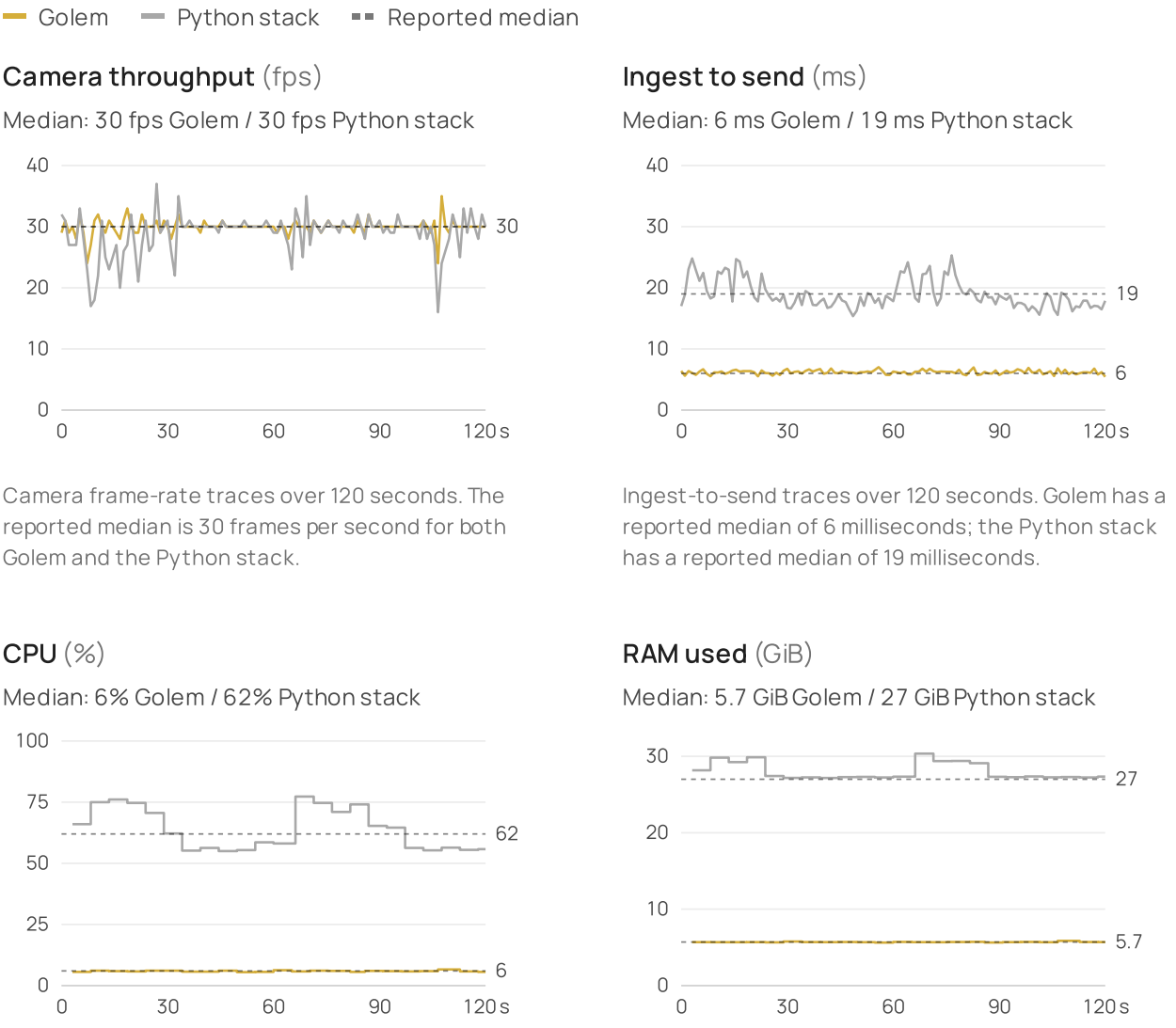
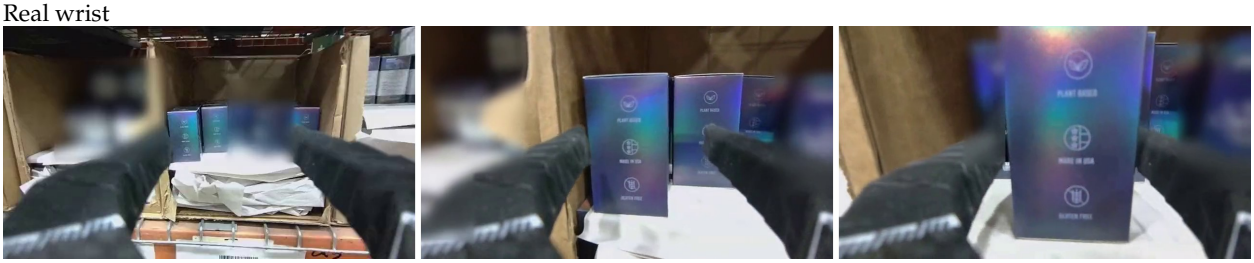


Figure 23: Golem and the Python stack over the same 120 s. Dashed lines mark the medians.

5 Scaling up

Real-world robot fleets deployed across diverse settings and tasks can collect the synchronized images, commands, and measured robot state to reconstruct high-fidelity simulators at massive scale. Below, agents reconstruct a Tutor demonstration recorded on a deployed Sonny in a real production environment.

Real wrist



Simulated wrist



Video: research.tutorintelligence.com/ti0-1#fig-24

Figure 24: A Tutor demonstration on a deployed Sonny and its reconstruction from the recorded telemetry, seen from the robot's wrist camera. Product labels are blurred.

Where deployments provide diversity and scale, DF1 provides a kindergarten for robots: a safe, flexible, and forgiving environment in which failure modes that would take hours to surface on a single robot appear within minutes [31]. Running simulation-trained policies across the fleet turns each failure into a targeted correction. It also lets us scale these methods to new tasks, with concurrent real-world evaluation on up to 100 robots.



Video: research.tutorintelligence.com/ti0-1#fig-25

Figure 25: DF1 in simulation, every station running the Lego task.

Our goal is to continue to develop these pipelines to be more performant and more autonomous, supporting reconstruction, training, and evaluation that runs continuously across large fleets both

in DF1 and in the field.

6 Conclusion

TRACE trains dexterous manipulation policies in simulation from a few Tutor demonstrations. Usable simulators for real-world transfer must meet a high bar for both visual and physical fidelity: TRACE fits how the robot responds to its commands and how objects respond to the robot against the robot's own recorded telemetry. Frontier models make this reconstruction practical to repeat for new tasks, with real-world rollouts closing the loop. TI-0.1, distilled from privileged-state RL policies trained in simulation, picks and places Lego bricks on Sonny autonomously and in real time. Alongside real-world data collection and evaluation in DF1, TRACE turns recorded robot demonstrations into far more experience than the recordings alone provide.

What we learned.

- **Visual fidelity is not enough.** Manipulation simulators need dynamics fit to synchronized commands and measured robot state, not just images.
- **Verify contact directly.** Task success can hide physically impossible grasps. Replaying recorded commands and inspecting contact catches what success rates miss.
- **Shape privileged-state policies for distillation.** We let the policy see only objects in the wrist camera's view and give it a smooth motion parameterization, so that its behavior can be successfully distilled into a visuomotor policy.
- **Hardware is the ground truth.** Real rollouts expose gaps, like the stud grasp, that no simulated metric revealed.

Important gaps remain. High-precision tasks demand finer geometry and contact fidelity than our current reconstructions provide. Calibration errors in cameras and robots propagate into both rendering and dynamics. Deformable objects remain difficult for physics engines in general [32], and compliant elements such as springs and soft gripper pads are hard to identify from telemetry alone.

If you are interested in working on scaling robot learning and deployment in the real world, please consider applying to join our team.

Acknowledgments

We thank the people across Tutor Intelligence who make real-world robot learning possible: our team of Tutors, whose demonstrations seed this work; the manufacturing and maintenance technicians and operations managers who build our robots and keep DF1 and our field fleet running; and our deployment teams across go-to-market, implementation, and customer success, who bring robots into real facilities. We also thank Tutor's customers, our long-term partners in bringing generally capable robots into the world for the common good.

References

[1] B. Acosta, W. Yang, and M. Posa. Validating Robotics Simulators on Real-World Impacts, 2022. URL https://arxiv.org/abs/2110.00541. arXiv:2110.00541.

[2] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete Problems in AI Safety, 2016. URL https://arxiv.org/abs/1606.06565. arXiv:1606.06565.

[3] Anthropic. Claude Opus 5.5, 2026. URL https://www.anthropic.com/claude-opus-5-5.

[4] Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies, 2025. arXiv:2506.07339.

[5] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by Cheating, 2019. URL https://arxiv.org/abs/1912.12294. arXiv:1912.12294.

[6] K. Chen, S. Xie, U. Yoo, X. Zhu, C. Magner, A. Zhu, D. Kim, E. Heiden, E. Coumans, M. Macklin, et al. RoboSimStudio: An Interactive Toolkit for Human and Agent Rapid Real2Sim of Robot Manipulation Tasks, 2026. URL https://arxiv.org/abs/2610.18835. arXiv:2610.18835. Project page: https://robosimstudio.github.io/.

[7] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026.

[8] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *RSS*, 2024. arXiv:2402.10329.

[9] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse Curriculum Generation for Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, 2017. URL https://arxiv.org/abs/1707.05300. arXiv:1707.05300.

[10] J. Gruenstein, T. Chen, N. Doshi, and P. Agrawal. Residual Model Learning for Microrobot Control, 2021. URL https://arxiv.org/abs/2104.00631. arXiv:2104.00631.

[11] T. He, Z. Wang, H. Xue, Q. Ben, Z. Luo, W. Xiao, Y. Yuan, X. Da, F. Castañeda, S. Sastry, et al. VIRAL: Visual Sim-to-Real at Scale for Humanoid Loco-Manipulation, 2025. URL https://arxiv.org/abs/2511.15200. arXiv:2511.15200.

[12] T. A. Henzinger. The Theory of Hybrid Automata. In *IEEE Symposium on Logic in Computer Science (LICS)*, 1996. doi: 10.1109/LICS.1996.561342. URL https://doi.org/10.1109/LICS.1996.561342.

[13] J. Huang, Q. Zhou, H. Rabeti, A. Korovko, H. Ling, X. Ren, T. Shen, J. Gao, D. Slepichev, C. Lin, et al. ViPE: Video Pose Engine for 3D Geometric Perception, 2025. URL https://arxiv.org/abs/2508.10934. arXiv:2508.10934. Project page: https://research.nvidia.com/labs/toronto-ai/vipe/.

[14] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning Agile and Dynamic Motor Skills for Legged Robots, 2019. URL https://arxiv.org/abs/1901.08652. arXiv:1901.08652.

[15] N. Jakobi, P. Husbands, and I. Harvey. Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In *Advances in Artificial Life: Third European Conference on Artificial Life (ECAL)*, volume 929 of *LNCS*, 1995.

[16] C. Jing, J. K. Bandi, J. Ye, Y. Duan, P. Abbeel, X. Wang, and S. Yi. Contact-Aware Neural Dynamics, 2026. URL https://arxiv.org/abs/2601.12796. arXiv:2601.12796.

[17] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *ICRA*, 2019. arXiv:1810.02890.

[18] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning Quadrupedal Locomotion over Challenging Terrain, 2020. URL https://arxiv.org/abs/2010.11251. arXiv:2010.11251.

[19] M. Mittal, P. Roth, J. Tigue, A. Richard, O. Zhang, P. Du, A. Serrano-Muñoz, X. Yao, R. Zurbrügg, N. Rudin, et al. Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning, 2025. URL https://arxiv.org/abs/2511.04831. arXiv:2511.04831.

[20] Moonlake AI. A Simulation-enabled Scaling Paradigm for Robot Learning, 2026. URL https://moonlakeai.com/blog/simulation-enabled-scaling-paradigm.

[21] OpenAI. GPT-6 Astra, 2026. URL https://openai.com/index/gpt-6-astra/.

[22] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving Rubik's Cube with a Robot Hand, 2019. URL https://arxiv.org/abs/1910.07113. arXiv:1910.07113.

[23] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization, 2018. URL https://arxiv.org/abs/1710.06537. arXiv:1710.06537.

[24] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning, 2017. URL https://arxiv.org/abs/1710.06542. arXiv:1710.06542.

[25] 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, 2024. URL https://arxiv.org/abs/2409.10161. arXiv:2409.10161.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347. arXiv:1707.06347.

[27] V. Surendran, N. Peri, and D. Watkins. Bridging Handheld and Teleoperated Supervision for Contact-Rich Manipulation via State-Gated Experts, 2026. URL https://arxiv.org/abs/2606.26603. arXiv:2606.26603.

[28] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017. arXiv:1703.06907.

[29] M. Torne, 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, 2024. URL https://arxiv.org/abs/2403.03949. arXiv:2403.03949.

[30] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, volume 80 of *PMLR*, 2018. URL https://proceedings.mlr.press/v80/icarte18a.html.

[31] Tutor Intelligence. Data Factory 1 Technical Report. Technical report, Tutor Intelligence, 2026. URL https://cdn.prod.website-files.com/68fa203de60162c631624d70/69f3b223d5ec2d66

4d4a497c_5441a2fae0791729c87f025e68f26443_tutor_intelligence_data_factory_1_technical_report.pdf.

[32] S. Wang, Y. Feng, X. Jiang, S. Tian, N. Yan, X. Shen, C. Lyu, H. Wang, Y. Zhou, H. Wang, et al. GAUGE: A Measurement-Grounded Benchmark for Physical Fidelity in Simulation Engines and Video World Models, 2026. URL https://arxiv.org/abs/2608.05948. arXiv:2608.05948.

[33] 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, 2026. URL https://arxiv.org/abs/2610.02204. arXiv:2610.02204. Project page: https://rpg-robot.github.io/.

[34] O. Zhang, I. Akinola, R. Scalise, M. Guaman Castro, P. Yin, T. Widmer, M. Macklin, C. Amevor, D. Turpin, D. Van Gelder, et al. AlphaNIST: Emergent Assembly Skills from Binary Task Rewards, 2026. URL https://arxiv.org/abs/2610.13354. arXiv:2610.13354. Project page: https://alpha-nist.github.io/.

[35] O. Zhang, M. Guaman Castro, P. Yin, I. Dagnino, A. Gupta, R. Scalise, and B. Boots. A Balanced Data Diet: Addressing the Exploration Bottleneck in Mega-Scale RL for Robot Control, 2026. URL https://arxiv.org/abs/2610.12465. arXiv:2610.12465. Project page: https://sgs-rl.github.io/.

[36] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS*, 2023. arXiv:2304.13705.