GR00T 2: Scaling Robot Foundation Models for Generalization, Dexterity, and Adaptation

NVIDIA
https://research.nvidia.com/labs/gear/gr00t-2

Contents

1 Introduction **3**

2 GR00T 2 Overview **5**

3 GR00T 2 Capabilities **6**
3.1 Fast Adaptation: Zero-Shot Generalization and Few-Shot Mastery . . . 6
3.2 X-Adaptability: Cross-Embodiment Skill Transfer . . . 8
3.3 Robust Deployment through Reinforcement Learning . . . 9
3.4 Agentic GR00T 2 . . . 10
3.5 Humanoid Loco-Manipulation . . . 13

4 How We Made GR00T 2 **14**
4.1 Training: Pre-Training and RL Post-Training . . . 14
4.2 Data . . . 15
4.3 Real-World Benchmarking: Mapping the Hill We Climb . . . 16
4.4 Learning without Teleoperation: Devices and Pipelines . . . 17
4.5 Simulation and Evaluation . . . 20
4.6 Deployment and Real-Time Inference . . . 22
4.7 Infrastructure . . . 24

5 Conclusion and Open Problems **26**

A Fast-Adapt with GR00T 2 **38**
A.1 Benchmark Tasks . . . 38
A.2 Zero-Shot Generalization with GR00T 2 . . . 40
A.3 Dexter-50: Emergent Cross-Embodiment Zero-Shot and Few-Shot Transfer. . . . 41

B X-Adapt with GR00T 2 **42**
B.1 Human → Tianji + Sharpa . . . 42
B.2 UMI → YAM . . . 48
B.3 DexUMI for Complex, Long-Horizon, and Dynamic Post-Training . . . 50
B.4 DexUMI → Tianji + Sharpa . . . 52
B.5 Human → Simulation → Tianji + Sharpa . . . 53

C Data for GR00T 2 **57**
C.1 Human Data for GR00T 2 . . . 57
C.2 UMI-10K for GR00T 2 . . . 61
C.3 Teleoperation Data for GR00T 2 . . . 62
C.4 Data Annotations . . . 63

D Real-World Benchmarking with GearArena **65**

© 2026 NVIDIA. All rights reserved.

D.1 Why Aggregate Real-Robot Evaluations, and Why A/B . . . 65
D.2 Designing Evaluation Tasks from First Principles . . . 66
D.3 The Arena: Stations, Operators, Campaigns, and Suites . . . 67
D.4 Leaderboard Mathematics . . . 67
D.5 A/B Evaluation Infrastructure . . . 69
D.6 Results: What the Arena Shows about GR00T 2 . . . 69
D.7 Limitations . . . 71

E UMIverse for GR00T 2 **73**
E.1 Gripper UMI Hardware . . . 73
E.2 UMI Direct . . . 74
E.3 UMI → YAM Transfer: Embodiment Gaps and Ablations . . . 75
E.4 DexUMI Exoskeleton Hardware . . . 81
E.5 DexUMI Post-Training Recipe . . . 82
E.6 DexUMI WAM-Native Inpainting . . . 82

F RL Post-Training **84**
F.1 RL-Based Adaptation of World-Action Models . . . 84
F.2 Automated Discovery of RL Post-Training Algorithms . . . 87
F.3 Online RL Post-Training Infrastructure . . . 89
F.4 Precision Manipulation through RL Post-Training . . . 91
F.5 Reinforcement Learning with Neural Simulation . . . 92

G Agentic GR00T 2 **98**
G.1 Agentic Skill Composition and Recovery with Code-as-Policy . . . 98
G.2 Agentic Steering with Visual Guidance . . . 102
G.3 Toward Robotic Self-Improvement . . . 107

H Humanoid Loco-Manipulation **109**
H.1 GR00T 2 Turbo on Whole-Body Latent Actions . . . 109
H.2 Real-World Whole-Body Loco-Manipulation . . . 109
H.3 WAMs Consume Simulation Data Better than VLAs . . . 111

I Accelerating GR00T 2 **113**
I.1 Accelerating Model Inference . . . 113
I.2 Policy Smoothness . . . 115
I.3 Accelerating Task Throughput Beyond Demonstrations . . . 116

J Simulations **118**
J.1 Real-to-Sim Generation of High-Fidelity Tasks . . . 118
J.2 Accelerating GR00T 2 Development with Simulation . . . 119
J.3 Simulation Benchmarking . . . 123

K Infrastructure **127**
K.1 Overall Software and Hardware Stack . . . 127
K.2 Hardware . . . 127
K.3 Cluster and Job Management . . . 128
K.4 Distributed Training . . . 129
K.5 Inference Infrastructure . . . 132
K.6 Scalable Multi-Modal Data Processing . . . 133

1. Introduction

Building general-purpose robots requires models that can acquire diverse physical skills, generalize across tasks and embodiments, and adapt efficiently to new environments and objectives. Recent advances in vision-language-action (VLA) models have demonstrated the potential of large-scale pretraining for robotic manipulation. However, scaling robot intelligence involves challenges beyond learning from increasingly large demonstration datasets. A general-purpose robot foundation model must learn from heterogeneous sources of physical experience, acquire new capabilities with limited task-specific supervision, improve through interaction, and support reliable, responsive execution in complex robotic systems. Furthermore, as robots become increasingly autonomous, their learned physical capabilities must be accessible to higher-level reasoning and planning systems. Meeting these requirements calls for models that capture not only how robots act, but also how their actions change the physical world.

We introduce **GR00T 2**, a new generation of foundation world-action models (WAMs). GR00T 2 jointly predicts future visual observations and embodied actions, learning how physical interactions unfold alongside the actions that produce them. Through large-scale pretraining on diverse human and robot experience, GR00T 2 develops general-purpose manipulation capabilities across both parallel-gripper robots and dexterous hands. We demonstrate four major advances. First, GR00T 2 delivers strong out-of-the-box performance across diverse manipulation tasks and embodiments. Second, its pretrained capabilities enable rapid adaptation to challenging new tasks from remarkably few demonstrations, while reinforcement learning (RL) further improves manipulation reliability through real-world interaction. Third, GR00T 2 enables *x-adaptation* (cross adaptation), allowing physical knowledge acquired from one embodiment to be transferred to another through joint prediction of future observations and actions. Finally, GR00T 2 supports *agentic control* through two complementary capabilities: skill composition and recovery with code-as-policy (CaP), enabling higher-level agents to orchestrate manipulation and recover from execution failures, and *steerable world-action generation* through a *WAM API*, allowing agents to guide and compose robot behaviors using visual targets.

Emergence of X-Adaptability. A central innovation of GR00T 2 is its cross-adaptation (**x-adapt**) capability, which allows the model to generalize to task-scene-embodiment combinations unseen during training. Consider a manipulation task observed only in human demonstrations and a robot embodiment represented by data from an entirely different set of tasks. Through joint world-action pretraining, GR00T 2 learns to imagine how the task would unfold when performed by the target robot and to generate actions corresponding to that imagined interaction. This creates a mechanism for transferring physical knowledge across heterogeneous sources of experience, including UMI demonstrations, dexterous UMI data, human hand interactions, and robot embodiments with substantially different kinematics. We explore this capability through cross-embodiment transfer experiments and a zero-teleoperation dexterous learning system, demonstrating the potential to acquire robot skills directly from human experience without collecting task-specific robot demonstrations. More broadly, cross-embodiment imagination points toward a new scaling property of robot foundation models: physical behaviors learned during pretraining can become executable capabilities across embodiments.

Fast Adaptation and Agentic Control. GR00T 2 advances the capabilities of pretrained robot policies across generalization, dexterity, and adaptation. A diverse pretraining mixture equips the model with a broad repertoire of physical behaviors, enabling strong out-of-the-box performance and rapid acquisition of challenging new skills from few demonstrations. We further extend these capabilities through RL to improve manipulation reliability and efficient inference for responsive, closed-loop control. Building on these learned skills, we integrate GR00T 2 with higher-level reasoning agents that use CaP to compose manipulation skills, monitor execution, and recover from unexpected failures, bringing GR00T 2 closer to deployment-level reliability on challenging long-horizon manipulation tasks. Beyond skill composition and recovery, GR00T 2's joint world-action representation provides a natural interface for steering robot behavior through desired visual outcomes and intermediate future states. We develop a *WAM API* that exposes this capability to higher-level agents, allowing them to guide and compose robot behaviors through visual targets. Together, these capabilities

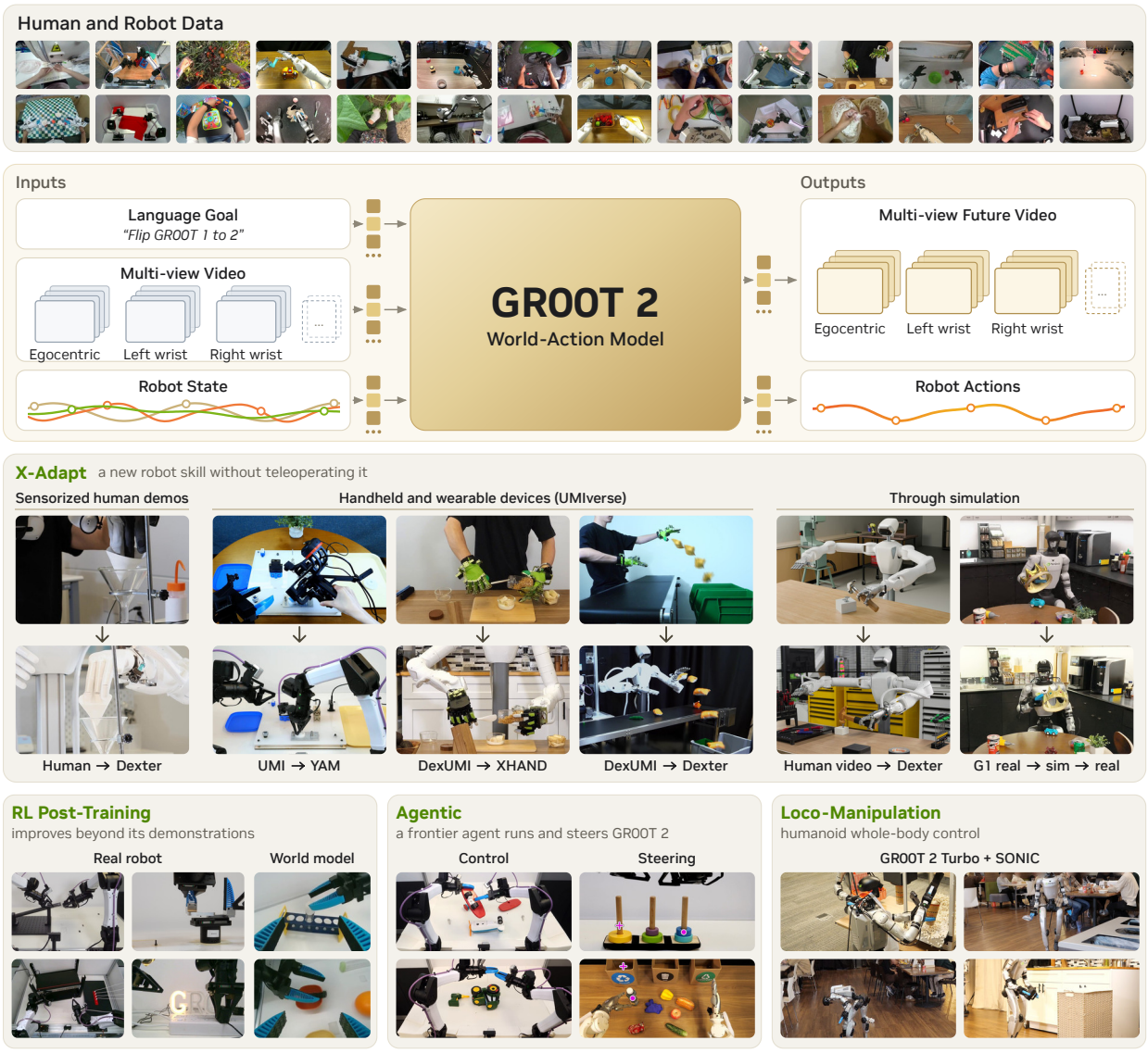


Figure 1: We introduce GR00T 2, a world-action model for general-purpose robots. *Top:* GR00T 2 pretrains on human and robot data. *Second row:* given a language goal, multi-view video from the egocentric and both wrist cameras, and the robot state, GR00T 2 jointly predicts multi-view future video and robot actions. *Third row:* with X-Adapt, GR00T 2 learns a new robot skill from sensorized human demonstrations (Human → Dexter, a Tianji robot with Sharpa hands), from handheld and wearable capture devices (UMI → YAM, DexUMI → XHAND, and DexUMI → Dexter), and through simulation, either from a single human video (Human video → Dexter) or by replaying teleoperated G1 motions across varied simulated scenes (G1 real → sim → real). *Bottom:* RL post-training on real robots and in a world model improves GR00T 2 beyond its demonstrations (*left*), a frontier agent runs and steers GR00T 2 (*center*), and GR00T 2 Turbo directs the SONIC whole-body controller for humanoid loco-manipulation (*right*).

create a path toward agentic robotic systems in which high-level reasoning and planning can directly leverage the physical intelligence acquired through world-action pretraining.

GR00T 2 Turbo: Token-Efficient World Modeling. We also introduce GR00T 2 Turbo, a lightweight variant of GR00T 2 designed for efficient world-action modeling. Joint video-action prediction enables actions to benefit from learned visual dynamics, but processing video tokens can be costly, particularly for multi-view, low-latency control. GR00T 2 Turbo addresses this cost with a more compact video representation while retaining the joint world-action modeling framework. This design enables faster training and inference, making GR00T 2 Turbo well suited for whole-body control while remaining well above the baselines on bimanual and dexterous

manipulation tasks.

Realizing these advances requires innovations throughout the robot learning stack. We first describe the GR00T 2 architecture and pretraining framework, including Human-1M, our large-scale corpus of embodied human experience (Sec. 4.2.1), and the data processing and training infrastructure that support learning from heterogeneous sources. We then investigate cross-embodiment transfer and introduce a dexterous human-to-robot learning system that combines motion retargeting, visual alignment, and world-action modeling. Beyond pretraining, we develop reinforcement learning methods for improving manipulation through interaction and a steerable WAM API that enables agentic planning and guided execution.

We evaluate GR00T 2 across a broad suite of simulated and real-world manipulation tasks, spanning general-purpose policy performance, few-shot adaptation, cross-embodiment transfer, dexterous manipulation, and whole-body humanoid control. Our results demonstrate the versatility of world-action pretraining as a foundation for acquiring and transferring physical skills, and highlight the complementary roles of large-scale pretraining, interactive post-training, and system-level integration. Together, GR00T 2 represents a step toward general-purpose robot foundation models whose physical knowledge can be transferred across embodiments, efficiently adapted to new tasks, and directly accessed and directed by intelligent agents.

2. GR00T 2 Overview

GR00T 2 is a world-action model (WAM): a single model that, given the robot's visual and proprioceptive input and a user instruction, predicts both how the scene will unfold and the robot's actions. One pretrained checkpoint covers parallel-gripper arms, dexterous hands, and a humanoid, and learns diverse source data sources including human egocentric videos, handheld UMI gripper data, and robot teleopration data. (Sec. 4.2).

Inputs and outputs. At every control step, GR00T 2 receives (i) a 0.8s history of 224×320 images from up to four cameras at 5 fps, typically a head camera and one camera on each wrist, (ii) the robot's proprioceptive state, and (iii) a natural-language instruction. It returns a chunk of future actions together with the predicted future frames of every camera. The robot executes the chunk through its own controller while the next chunk is computed (Sec. K.5).

Architecture. GR00T 2 is built on Cosmos-Predict2.5-2B, a video diffusion transformer pretrained for world simulation (NVIDIA et al., 2025a). Video, state, and action inputs become tokens in one sequence that a single 2B-parameter transformer processes (Fig. 1):

- *Video.* A frozen Wan 2.1 video VAE (Wan Team et al., 2025) compresses each camera stream $4\times$ in time and $8\times$ in each spatial dimension. The latents are split into patches, giving 280 tokens per latent frame and camera at 224×320. A learned view embedding tells the model which camera each token comes from. Past frames enter as context, and future frames are generated.
- *Language.* A frozen Cosmos-Reason1 vision-language model (NVIDIA et al., 2025b) encodes the instruction, and the transformer reads it through cross-attention.
- *State and actions.* Small embodiment-specific MLPs map the robot state and the noisy action chunk into the transformer's width, and an embodiment-specific decoder maps the output tokens back to actions.

Inside the transformer, video tokens attend to one another across time and cameras, and action tokens attend to all video tokens, including the future frames being generated. Actions can therefore follow the future the model imagines. GR00T 2 is trained with flow matching (Lipman et al., 2023) on future video latents and actions together, with one loss for each. At inference, a few denoising steps suffice: one or two for video and two to four for actions in our deployments (Sec. I.1).

GR00T 2 Turbo: Efficient World-Action Modeling. While GR00T 2 already runs in real time when served from a workstation GPU (a median of 153 ms to predict an action chunk for Dexter from three camera views on a GeForce RTX 5090, Tab. 27), humanoid whole-body loco-manipulation sets a much stricter budget. The policy must react quickly to keep the whole body coordinated as the robot moves, and it must run onboard on

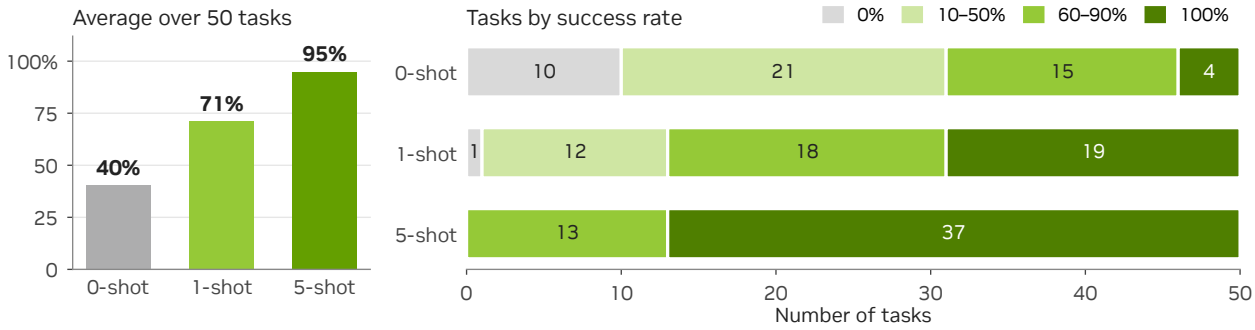


Figure 2: Cross-embodiment transfer to Dexter with 0, 1, and 5 Dexter episodes. Results on the Dexter-50 tasks, which have no Dexter training data but appear in YAM teleoperation data, human video, or both. Zero-shot runs the pretrained model directly, while 1-shot and 5-shot fine-tune it with LoRA on 1 or 5 Dexter episodes. Left: average success rate over the 50 tasks. Right: number of tasks in each success-rate range. The tasks are those the model learns with five episodes (at least 80% success). Per-task results are in Fig. 24.

an edge GPU such as the NVIDIA Jetson AGX Thor, which has far less compute. We therefore design GR00T 2 Turbo to make the world-modeling objective itself even cheaper. Most of the tokens GR00T 2 predicts are future video, and video is highly redundant. Turbo keeps the 2B transformer that models video and actions jointly but encodes video with a higher-compression VAE, so it still predicts future observations and actions together, with far fewer visual tokens. This makes both training and inference faster. Turbo predicts in 124 ms on the onboard Jetson Thor of the Unitree G1, where it plans whole-body motion for the SONIC controller (Sec. 3.5 and 4.6.1). Turbo trails GR00T 2 on dexterous tasks but remains well above the baselines on bimanual and dexterous manipulation, and offers a favorable tradeoff between task performance and inference latency (Sec. 3.1 and Fig. 23).

Specifically, Turbo replaces the video VAE with the Wan 2.2 VAE (Wan Team et al., 2025), increasing spatial compression from $8\times$ to $16\times$ per dimension and producing a $4\times$ smaller spatial latent grid. The latent input and output projections are adapted to the new VAE, while the action horizon remains unchanged.

Using GR00T 2 on a new robot. To bring up a robot, a practitioner maps its cameras to the head and wrist views, expresses its actions in a supported action space (or adds a new one), and post-trains GR00T 2 on demonstrations of the target tasks. Sec. 3 shows how few demonstrations this takes, and how to replace teleoperation with human or UMI demonstrations altogether.

3. GR00T 2 Capabilities

We now present the capabilities of GR00T 2. Sec. 3.1 shows zero-shot deployment and few-shot adaptation on new tasks and post-training on long-horizon tasks that demand precision and dexterity, and Sec. 3.2 studies cross-embodiment transfer, in which GR00T 2 learns a skill on a robot without teleoperated demonstrations of that skill. Sec. 3.3 shows that reinforcement learning improves GR00T 2 beyond its demonstrations, Sec. 3.4 pairs GR00T 2 with a frontier agent for long-horizon tasks, and Sec. 3.5 extends GR00T 2 to whole-body humanoid loco-manipulation. We report the main results here and defer the full evaluations to the appendix.

3.1. Fast Adaptation: Zero-Shot Generalization and Few-Shot Mastery

Zero-shot deployment on new tasks. Pretraining on teleoperation data from many robots and on Human-1M human video (Sec. 4.2) gives GR00T 2 an emergent ability to perform tasks it has never seen on the target robot. We test this on Dexter (Tianji robot + Sharpa hands), a bimanual robot with two 22-DoF dexterous hands, with 50 tasks that are absent from Dexter's own data and appear only in YAM gripper data, human video, or both. Deployed out of the box and conditioned only on the language instruction, GR00T 2 succeeds in 40% of trials and completes 40 of the 50 tasks at least once (Fig. 2). Most failures come from a single step

Table 1: Post-training benchmarks. For each model, Prog. is the average task progress score across all trials and SR is the success rate, both on a scale from 0 to 100. Bold marks the best Prog. and SR in each row, and the last row of each table averages its tasks. Each YAM result is based on up to 20 trials. On Truck Assembly, GR00T 2 and GR00T 2 Turbo were evaluated on a revised, easier set of start scenes, and on the original scenes GR00T 2 succeeds in 7 of 20 trials with a progress score of 94.0. Baselines: $\pi_{0.5}$ (Physical Intelligence et al., 2025), GR00T 1.7 (NVIDIA, 2026a), FastWAM (Yuan et al., 2026), MolmoAct2 (Fang et al., 2026), and FLUX 3 Action (Black Forest Labs, 2026).

(a) YAM long-horizon post-training benchmark

	GR00T 2		**GR00T 2 Turbo**		$\pi_{0.5}$		GR00T 1.7		FastWAM		MolmoAct2		FLUX 3 Action	
Task	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR
Hot Wheels Assembly	**95.4**	**85.0**	68.6	50.0	13.1	0.0	19.5	0.0	0.8	0.0	1.5	0.0	0.5	0.0
Toy Tractor Assembly	**63.8**	**45.0**	54.5	40.0	8.4	0.0	3.6	0.0	1.9	0.0	1.1	0.0	2.4	0.0
Toy Plane Assembly	**71.0**	**70.0**	63.5	55.0	24.0	10.0	11.0	0.0	10.0	0.0	13.5	0.0	10.0	0.0
iPhone into Box	89.0	**65.0**	**91.0**	55.0	67.5	5.0	49.5	15.0	17.0	0.0	38.0	0.0	19.5	0.0
Fixed-Pipe Fittings	70.0	70.0	**90.0**	**90.0**	30.0	30.0	35.0	35.0	45.0	45.0	35.0	35.0	0.0	0.0
Wire into Breaker	98.5	95.0	**100.0**	**100.0**	30.0	30.0	17.0	0.0	18.5	5.0	10.0	10.0	8.0	0.0
Average	**81.3**	**71.7**	77.9	65.0	28.8	12.5	22.6	8.3	15.5	8.3	16.5	7.5	6.7	0.0

(b) Dexterous-hand long-horizon post-training benchmark

	GR00T 2		**GR00T 2 Turbo**		GR00T 1.7		$\pi_{0.5}$	
Task	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR
Make Shipping Label	**78.0**	**60.0**	73.0	**60.0**	41.0	5.0	16.0	0.0
Flashlight Battery Replacement	**86.0**	**55.0**	49.0	15.0	34.0	0.0	41.0	0.0
GB300 Memory	**96.0**	**85.0**	81.0	50.0	43.0	0.0	55.0	0.0
Truck Assembly	**98.0**	**65.0**	43.0	5.0	0.0	0.0	0.0	0.0
Crude Salt Filtering	**98.0**	**85.0**	88.0	65.0	17.0	0.0	14.0	0.0
Vitamin C Test	**86.0**	**75.0**	67.0	25.0	9.0	0.0	14.0	0.0
Microscope	**64.0**	**55.0**	33.0	20.0	10.0	0.0	9.0	0.0
Average	**86.6**	**68.6**	62.0	34.3	22.0	0.7	21.3	0.0

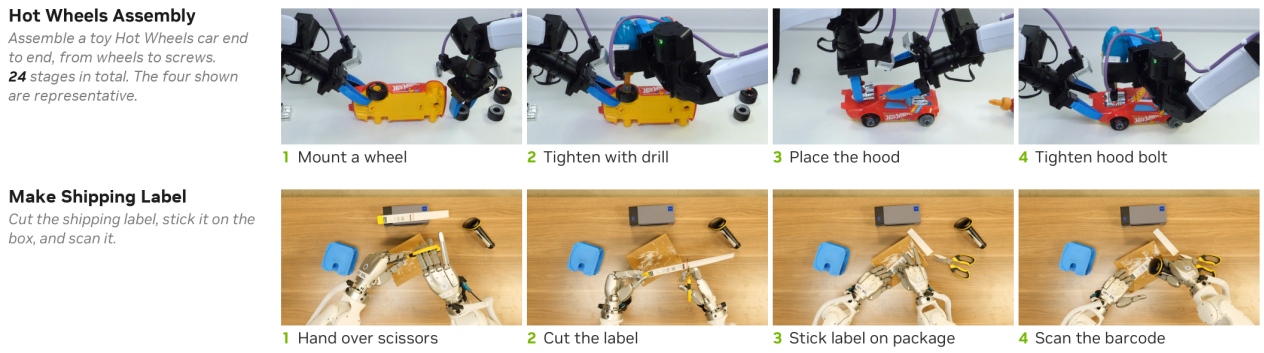


Figure 3: Representative tasks from the two post-training benchmarks. Hot Wheels Assembly on YAM (top) and Make Shipping Label on Dexter (bottom).

that falls short on precision within an otherwise correct execution. The ability grows with data, and zero-shot success on new Dexter tasks more than doubles as the amount of human video in pretraining increases (Fig. 10). On broader zero-shot suites for YAM and Dexter, GR00T 2 also leads the VLA baselines in success rate (Tabs. 2 and 3).

Few-shot mastery with minutes of robot data. Because the pretrained model already knows how to approach

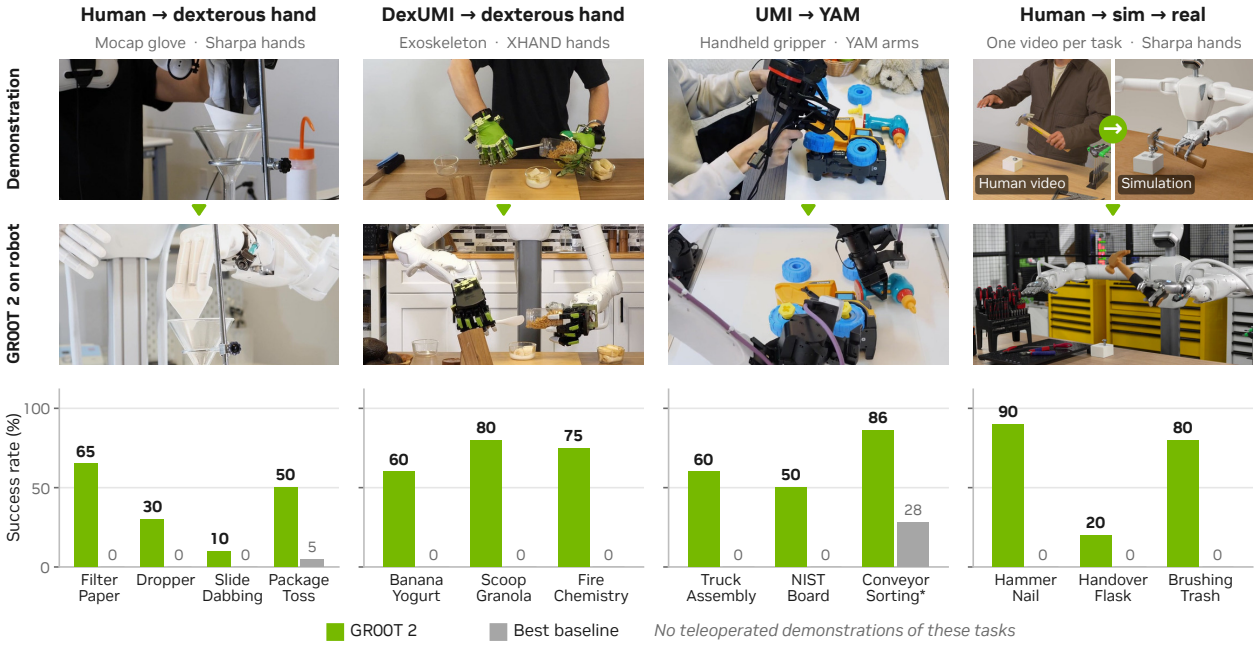


Figure 4: Learning new tasks without teleoperation. Each column is one way to demonstrate a task without the target robot. It shows a demonstration, a GR00T 2 rollout on the robot, and the real-robot success rate per task. Best baseline is the highest success among the other models evaluated on that task (GR00T 1.7 (NVIDIA, 2026a), $\pi_{0.5}$ (Physical Intelligence et al., 2025), LingBot-VA (Li et al., 2026)). *For Conveyor Sorting, the bars show the share of the 50 conveyor items sorted, not the share of successful trials. Full results are in Fig. 27 and Tabs. 4, 5 and 7.

these tasks, very little robot data closes the gap. One teleoperated episode per task lifts success to 71%, and five episodes, under three minutes of robot data, lift it to 95%, with 37 of the 50 tasks succeeding in every trial. The 50 tasks were chosen among those the model learns with five episodes (at least 80% success), so the five-episode rate is high partly by construction. Each adaptation is a LoRA fine-tune of about 30 minutes on a single RTX 5090, and merging the adapter into the weights means the adapted model is served exactly like the pretrained one (Sec. A.3).

Long-horizon tasks that demand precision and dexterity. The same pretraining pays off when GR00T 2 is post-trained on long-horizon tasks that chain many precise steps. On six YAM tasks, from assembling a toy car end to end in 24 stages to inserting a wire into a breaker, GR00T 2 reaches 71.7% average success, compared with at most 12.5% for the VLA and WAM baselines (Tab. 1a and Fig. 3). On seven dexterous-hand tasks, such as filtering crude salt, testing for vitamin C, and swapping a GB300 memory module, GR00T 2 reaches 68.6% average success, compared with at most 0.7% for GR00T 1.7 and $\pi_{0.5}$ (Tab. 1b). Completing such tasks end to end, with dexterous hands and tight tolerances at every step, is a new capability for a generalist robot model. All tasks of both benchmarks are shown in Sec. A.1.

GR00T 2 Turbo. Although Turbo is designed primarily for onboard deployment on edge devices (Sec. 2), it still retains much of GR00T 2 capability. In zero-shot evaluation, it stays close to GR00T 2 on YAM and ranks second among the evaluated models on Dexter (Tabs. 2 and 3). After post-training it gives up some success, reaching 65.0% on the YAM tasks and 34.3% on the dexterous-hand tasks, still well above the baselines. Fig. 23 shows the resulting trade-off between task progress and inference latency.

3.2. X-Adaptability: Cross-Embodiment Skill Transfer

GR00T 2 can learn a new task on a robot without any task-specific teleoperated demonstration. Fig. 4 summarizes four ways to teach it such a task, from demonstrations recorded with a motion-capture glove, the DexUMI exoskeleton, or the handheld Gripper UMI, and from a single human video expanded in simulation.

For the glove and Gripper UMI demonstrations, which come from a different embodiment than the robot, one recipe bridges the gap. Demonstrations recorded without the robot specify the *task*, task-agnostic play data from the target robot specify the *embodiment*, and cross-embodiment pretraining connects the two. Without this pretraining, the same post-training data yield substantially lower performance (Sec. B.1.5). For the glove, DexUMI, and Gripper UMI demonstrations, the baselines are post-trained on the same data as GR00T 2 (Fig. 27 and Tabs. 4 and 5). When teleoperated demonstrations are available, they complement rather than replace the human data (App. B).

Transfer to dexterous hands. Two devices record dexterous demonstrations without the robot: a motion-capture glove worn on the bare hand, and the DexUMI exoskeleton, which replicates the robot hand's joints and contact surfaces and records fingertip forces in the hand's tactile format. With the glove, human demonstrations are recorded with head and wrist cameras, retargeted offline to the robot hand, and co-trained with play data of the target robot (Sec. B.1.3). On a bimanual robot with Sharpa hands, this recipe teaches precise chemistry-lab tasks and a fast conveyor toss that cannot be demonstrated through teleoperation. On one task, GR00T 2 trained without any teleoperated demonstration outperforms a baseline without cross-embodiment pretraining that receives ten (Fig. 27). DexUMI extends robot-free collection to tasks that depend on force and compliance: from its force recordings, GR00T 2 learns to predict the force to apply along with the motion, and the hand's stiffness adapts to that force. Trained on DexUMI demonstrations alone, GR00T 2 slices a banana and unscrews a jar in a long-horizon breakfast task and lights a lamp with a struck match, with 60 to 80% success per task, whereas GR00T 1.7 and $\pi_{0.5}$ complete none (Fig. 15 and Tab. 5). DexUMI demonstrations also transfer to a hand that the exoskeleton does not replicate. After retargeting to the 22-DoF Sharpa hand, GR00T 2 succeeds in 60% of trials on microscope slide dabbing and 65% on conveyor package selection and tossing, and the same two baselines succeed in none (Tab. 6).

Transfer to parallel-gripper robots. Demonstrations are recorded with the handheld Gripper UMI, in robot format through robot-style wrist cameras and a width encoder, and deployed on YAM, a robot that shares neither its cameras, nor its gripper, nor its workspace. GR00T 2 co-trains them with task-agnostic YAM play data that supply the embodiment and aligns them to YAM's camera geometry and action space (Sec. 4.4.2). With 41 to 75 demonstrations per task and no teleoperation, GR00T 2 completes a long-horizon toy assembly and a board disassembly in 60% and 50% of trials and sorts 86% of the items off a moving conveyor, whereas the baselines complete neither long-horizon task (Tab. 4). As few as 10 to 20 demonstrations yield end-to-end success. Collection is thus decoupled from the robot, and human-speed motion makes dynamic skills learnable.

Transfer from a single human video through simulation. From one human video, we reconstruct the objects and the hand-object motion, train an RL expert in simulation, and fine-tune GR00T 2 on visually randomized rollouts of that expert. The resulting policy transfers zero-shot to the real robot and drives a nail in 9 of 10 trials, whereas GR00T 1.7 and $\pi_{0.5}$ complete none of the real-world tasks (Sec. B.5).

3.3. Robust Deployment through Reinforcement Learning

Supervised fine-tuning (SFT) equips GR00T 2 with manipulation skills, but contact-rich tasks remain sensitive to small errors in alignment and physical interaction. We apply real-world RL on top of SFT policy to further improve the reliability of GR00T 2 through real-world interaction and task-level success feedback.

We evaluate four precision manipulation tasks: fixed-pipe fittings (the task of Tab. 1a, started from harder fitting poses), finger placement, cable insertion, and charger insertion (Fig. 5). We directly optimize GR00T 2 using Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and further train on successful GRPO rollouts through GRPO-induced self-imitation. We also compare against residual RL, which learns corrections on top of a frozen GR00T 2 policy (Sec. 4.1).

GRPO-induced self-imitation achieves **88%**, **98%**, **94%**, and **96%** success on the four tasks, respectively, compared with 40 to 76% after SFT. Residual RL also improves performance, but without updating the underlying GR00T 2 model (Fig. 5). These results show that real-world RL improves the reliability of GR00T 2

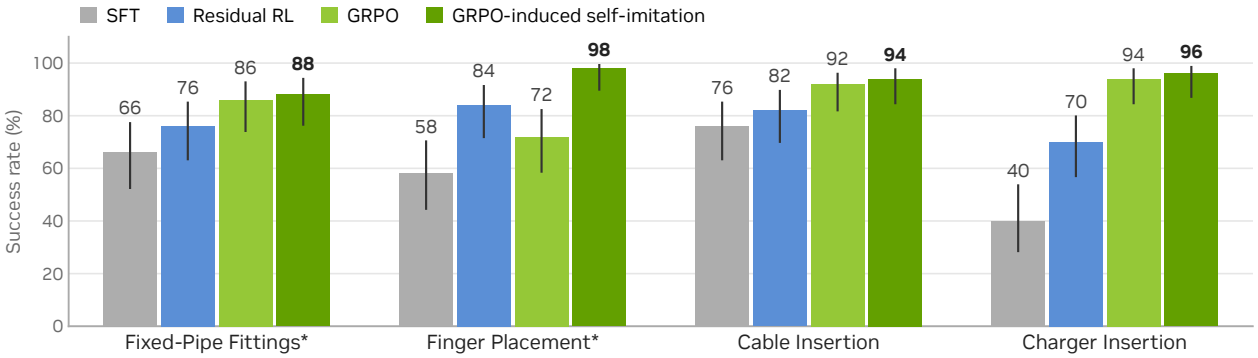


Figure 5: Real-robot post-training on four contact-rich manipulation tasks. Labels report success rates, and whiskers are 95% Wilson intervals over 50 trials per bar. Bold marks each task's best method. GRPO and GRPO-induced self-imitation update GR00T 2 directly, whereas residual RL learns a correction policy on top of a frozen GR00T 2. GRPO-induced self-imitation trains on successful GRPO rollouts. *Evaluated with CPS sampling noise.

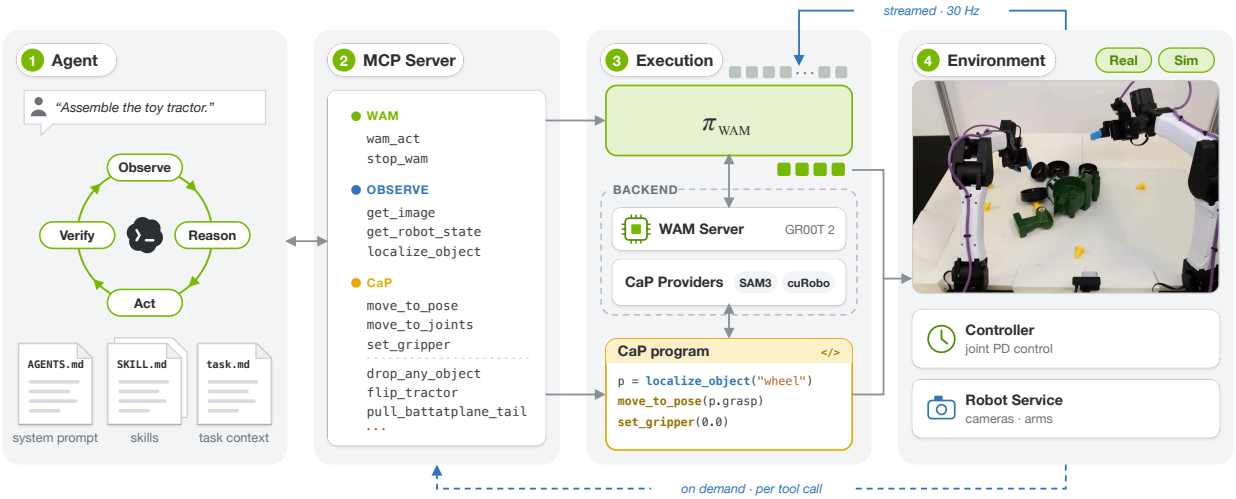


Figure 6: Agentic runtime architecture. System 2 monitors execution and invokes GR00T 2 and CaP tools through an observe-reason-act-verify loop. GR00T 2 or ASPIRE-generated program serves as execution backends with shared execution environment interface.

on contact-rich manipulation. The resulting skills can be integrated into agentic execution for more complex, long-horizon tasks (Sec. 3.4), as we will introduce next.

3.4. Agentic GR00T 2

A frontier multimodal agent and GR00T 2 address complementary parts of a task. The agent determines *what* to do: it decomposes long tasks, monitors progress, and writes code. GR00T 2 determines *how* to do it, performing the contact-rich manipulation that is difficult to script. The agent serves as System 2 and calls GR00T 2 as a tool alongside camera, state, and CaP tools (Fig. 6). It supports GR00T 2 in four ways. **Augment:** for tasks that need memory or explicit reasoning, the agent runs a task-level program that calls GR00T 2 for each manipulation step. Together with RL post-training, this puts GR00T 2 first on both the RoboCasa365 (Nasiriany et al., 2026) and RoboDojo (Chen et al., 2026a) leaderboards (Tab. 19). **Assist:** on real robots, the agent watches GR00T 2 and, when it sees a dropped part or stalled progress, repairs the state with a CaP skill and hands control back, lifting a post-trained GR00T 2 to near-perfect success on three cluttered assembly tasks (Fig. 7). **Guide:** beyond language, the agent can steer GR00T 2 with visual targets through the WAM API, described below. **Improve:** an autonomous research agent discovered the basis of the RL recipe we use to post-train GR00T 2 (Sec. 4.1

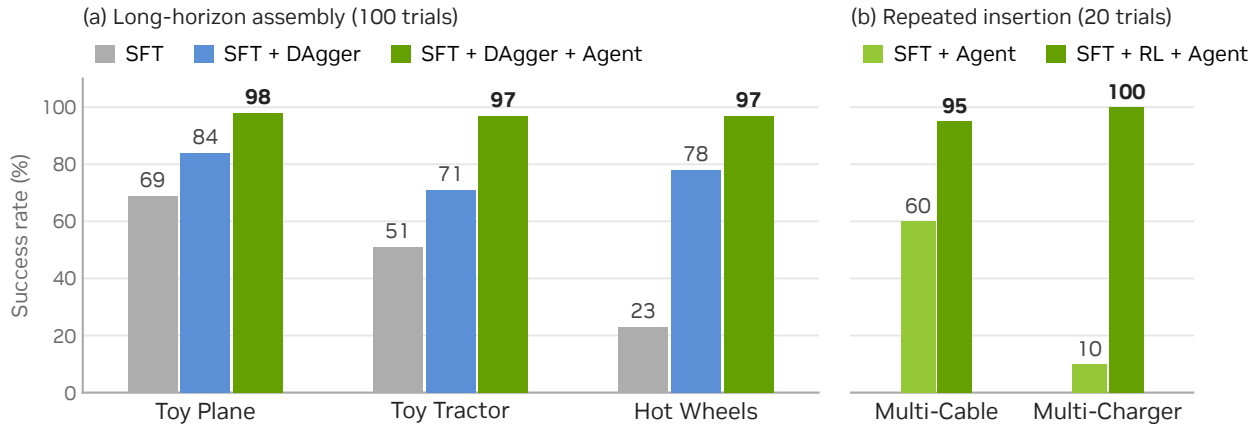


Figure 7: Agentic assistance on real robots. (a) Successful trials out of 100 per method on three long-horizon assembly tasks on YAM. Before every trial a person randomly dumps all parts onto the table, a cluttered starting configuration never seen in post-training and much harder than the arranged layouts of Tab. 1a. A trial that needed human help counts as a failure. The agent lifts GR00T 2 after SFT + DAgger to at least 97% on all three tasks. **(b)** Full-task success over 20 repeated-insertion episodes with SFT and RL-post-trained GR00T 2 in the same agentic framework. RL post-training raises success from 60% to 95% on multi-cable and from 10% to 100% on multi-charger insertion.

and F.2). We therefore recommend assigning manipulation to GR00T 2 and providing the agent with the task structure, success checks, and a small library of recovery skills (Sec. G.1).

Agentic Skill Composition and Recovery with Code-as-Policy. In simulation, we use ASPIRE (Lu et al., 2026) to discover task-level programs that compose GR00T 2 calls with perception, reasoning, and CaP primitives. These programs use GR00T 2 as a manipulation tool within more complex task sequences, while System 2 monitors execution and invokes online CaP recovery when unexpected failures occur. Combined with RL post-training, this approach achieves **88.0%** success on RoboCasa365 and **75.8%** on RoboDojo, evaluated on one of its three official layout banks, the highest reported success rates on both leaderboards as of November 6, 2026 (Tab. 19).

On physical robots, GR00T 2 remains the primary executor, while the agent monitors progress and invokes CaP skills for targeted recovery. We stress-test this approach on three long-horizon assembly tasks on YAM, a Toy Plane, a Toy Tractor, and a Hot Wheels car (Fig. 55), under conditions much harder than the post-training benchmark in Tab. 1a. There, every trial starts from an arranged layout of the parts. Here, a person **randomly dumps all parts onto the table** before each trial, so every trial starts from a new cluttered configuration, with overlapping parts and arbitrary poses never seen in post-training, and each method runs 100 trials. The robot must find the parts, assemble them in order, fasten them with a toy drill, and recover from its own errors, all within a single trial that typically lasts 5 to 14 minutes. These numbers therefore measure robustness and are not directly comparable with Tab. 1a. For Hot Wheels Assembly, the benchmark reports 85% success from its fixed layout, while the SFT policy completes 23 of 100 trials once the parts are dumped. DAgger (Ross et al., 2011) raises the three tasks to 84, 71, and 78 successes out of 100, and with the agent repairing dropped parts and stalls before handing control back to GR00T 2, hybrid execution reaches **98, 97, and 97 successes out of 100 trials** (Fig. 7). These results bring GR00T 2 closer to **deployment-level reliability**, with robust long-horizon manipulation from highly variable, unseen initial configurations.

Building on the RL-trained precision skills from Sec. 3.3, agentic execution composes these skills into longer tasks. In repeated cable and charger insertion, the robot must consecutively insert five Ethernet cables into a network switch or five chargers into selected outlets (Fig. 55). CaP handles object localization, grasping, and positioning, while RL-trained GR00T 2 performs the precision-critical insertions. The agent coordinates the sequence, verifies each insertion, and invokes CaP recovery when needed. Within the same agentic framework, replacing SFT policies with RL-post-trained GR00T 2 improves full-task success from 12/20 to **19/20** for cable

insertion and from 2/20 to **20/20** for charger insertion (Fig. 7).

Agentic Steering. With the agent acting as System 2 and GR00T 2 as System 1, a central design question is what the interface between them should be, that is, what the agent sends to GR00T 2 for each step; we refer to this interface as the WAM API. Language is the natural choice, but grounding it in the scene is hard: GR00T 2 must itself find the objects and locations that an instruction refers to. We therefore explore guiding GR00T 2 with image-space grasp and release points in addition to language: the points specify directly which object to manipulate, where to place it, and which arm acts. We mid-train GR00T 2 to condition on these points, rendered as arm-colored markers on its camera image alongside the language instruction. Point supervision is generated automatically from the gripper and hand states in existing demonstrations, requiring neither additional manual annotation nor new robot demonstrations. Once mid-trained, GR00T 2 can be steered by agent-generated points without task-specific policy fine-tuning. During execution, GR00T 2 generates the robot's actions, while the agent monitors progress and revises its guidance to advance, retry, or finish. In these experiments, the agent is used zero-shot, with one conversation per episode (Sec. G.2).

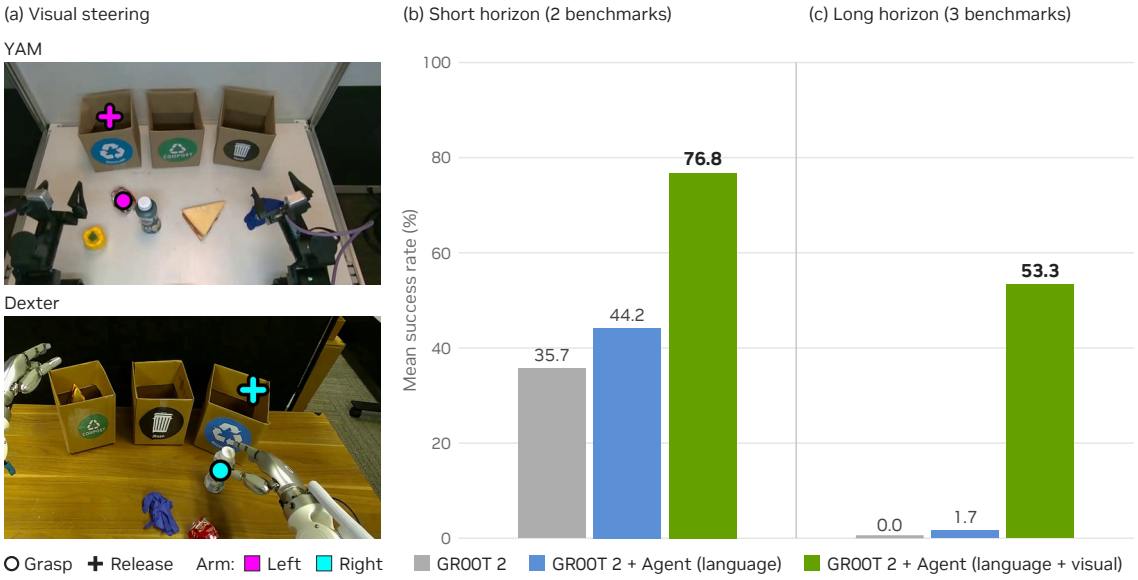


Figure 8: Higher average success with visual steering on both evaluation tiers. (a) Left- and right-hand steering examples. Circles mark grasp targets and crosses mark release targets, and colors identify the acting arm. **(b, c)** Equal-weight means of benchmark success rates, not pooled trial rates. Short horizon averages two benchmarks (144 and 64 trials per configuration; the Dexter benchmark was obtained with an earlier implementation of the model), and long horizon averages three robot-task combinations: trash sorting on both robots (20 episodes each) and a YAM-only board-disassembly stress test (10 episodes). Robot-specific protocols and results are given in Sec. G.2.

We evaluate this approach on two embodiments, YAM and Dexter, in two tiers: short-horizon benchmarks test only a few steps instruction following, while long-horizon benchmarks require completing an entire trash-sorting task on both robots, plus board disassembly on YAM as a stress test. Short-horizon instructions are organized by reference type to assess distinct grounding requirements (nominal, spatial, ordinal, cardinal, arm, and object-exclusion references) and execution requirements (action and grasp references). Sec. G.2 details this taxonomy together with the evaluation protocols, individual benchmarks, and instruction-category breakdowns. We compare three settings: GR00T 2 given the original task instruction, GR00T 2 given the agent's language guidance, and the point-conditioned GR00T 2 given the agent's language guidance and visual markers. Visual steering achieves the highest average success in both tiers: benchmark-mean success of **76.8%** on short-horizon tasks and mean full-task success of **53.3%** on long-horizon tasks, compared with 44.2% and 1.7% using the agent's language guidance alone (Fig. 8).

Despite these gains, grasp and release points cannot convey every task requirement. The markers specify spatial

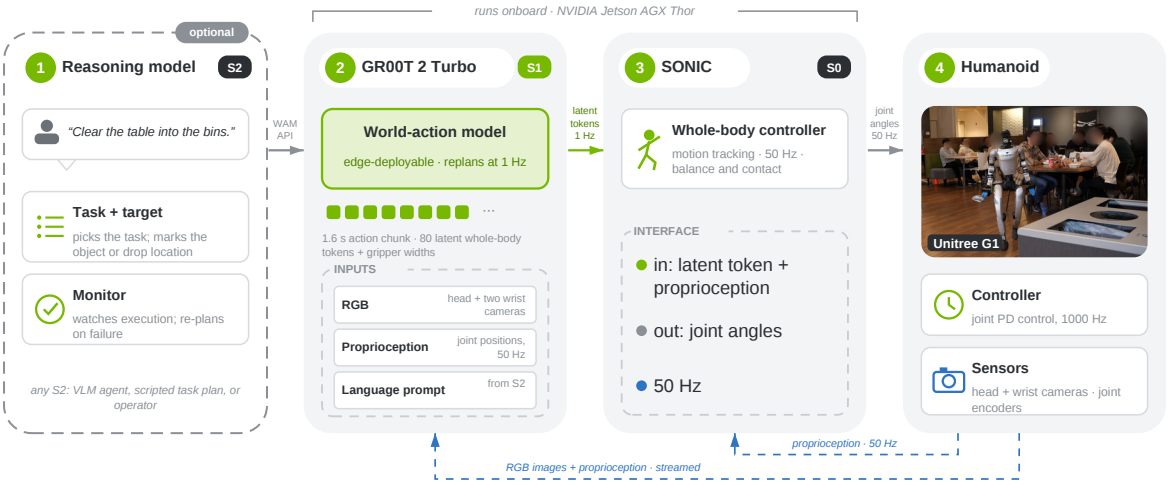


Figure 9: Three-level stack for whole-body loco-manipulation. An optional reasoning model (S2) picks the task and the prompt. GR00T 2 Turbo (S1) reads RGB images and proprioception and predicts 1.6 s chunks of SONIC whole-body motion tokens at about 1 Hz. SONIC (S0) decodes the tokens into joint position targets at 50 Hz, which the robot's PD controller tracks at 1000 Hz. S1 runs onboard on an NVIDIA Jetson AGX Thor, and S0 runs onboard on an NVIDIA Jetson Orin NX.

targets, not a complete manipulation strategy: requirements such as placing an object on its side, pouring, or scooping still depend on language and GR00T 2's learned skills. Richer guidance, including target 6D object poses, goal images, or affordance-based marking, could extend this interface.

3.5. Humanoid Loco-Manipulation

Bipedal humanoid loco-manipulation poses additional challenges to robotic foundation models: the robot must stay balanced through every motion, which requires high-frequency whole-body control at 50 Hz. We bring GR00T 2 to this setting by pairing GR00T 2 Turbo with SONIC (Luo et al., 2026), a learned whole-body controller (Fig. 9). Turbo uses camera observations and proprioception to choose what motion to take next, and SONIC turns that desired motion into joint commands for execution. Both models run onboard the humanoid using a multi-layer stack (Sec. H.1).

Whole-body Loco-manipulation. Using SONIC's whole-body motion latent interface as its action space, GR00T 2 Turbo coordinates the humanoid's entire body for locomotion and manipulation. We evaluate the resulting policies on four real-world long horizon whole-body tasks. In *popcorn serving*, the robot holds a cup while scooping and pouring with the other hand. In *table clean-up*, it repeatedly grasps diverse objects, walks to the trash bins, and places them inside. In *ground pick-up*, it bends down to pick up diverse objects. In *whole-body cleaning*, it picks up a cup from the floor, wipes a spill with a towel under its foot, then kneels to retrieve and discard the towel. On popcorn serving, Turbo completes all 48 trials, compared with 83.3% for GR00T 1.7 using the same demonstrations and controller. Each trial requires two filled cups. Detailed results for all four real-world tasks are reported in Sec. H.2 and Tab. 25.

Sim-and-real for Loco-manipulation. Whole-body teleoperation data is costly to collect, so we design a motion-centric real-to-sim pipeline that turns each recorded motion into demonstrations across varied scenes. We replay the motion through SONIC in simulation, keeping it fixed while fitting table and object placements so that the robot completes the pick, then render successful rollouts in diverse rooms. These demonstrations expand visual diversity while preserving the real robot's controller and action representation (Sec. H.3). The WAM makes better use of this simulated data than the VLA baseline on out-of-distribution table pick-and-place: starting from the same 6.7% success with real data alone, co-training raises GR00T 2 Turbo to 63.3%, compared with 40.0% for GR00T 1.7, over 30 trials each (Tab. 26). Reusing real motions in simulation therefore provides a practical way to improve the WAM's generalization without collecting additional teleoperated demonstrations.

4. How We Made GR00T 2

This section describes the methods behind GR00T 2, focusing on the key design decisions, while the appendix provides the full details. Sec. 4.1 presents the training recipe, from pretraining to RL post-training, and Sec. 4.2 the training data. Sec. 4.3 describes how we benchmark checkpoints against each other on real robots. Sec. 4.4 describes the capture devices and pipelines for learning without teleoperation, Sec. 4.5 the simulation and evaluation framework, Sec. 4.6 real-time deployment, and Sec. 4.7 the training and data infrastructure.

4.1. Training: Pre-Training and RL Post-Training

Pre-training. We initialize GR00T 2 from the pretrained Cosmos-Predict2.5-2B video diffusion transformer. To turn this video model into a general-purpose robot policy, we need it both to understand how physical interactions unfold and to connect those dynamics to actions across embodiments. We therefore train on a diverse mixture of human and robot experience in three stages: we first adapt the video model to embodied interactions, then learn video and actions jointly, and finally place greater emphasis on robot data. This progression lets us draw on the breadth of human experience while focusing the resulting model on robot execution.

Stage 1: Single-view video pre-training. We first adapt the pretrained video model to embodied visual experience from egocentric human video, supplemented by robot data. Human demonstrations provide diverse examples of hands interacting with objects across tasks and environments, while robot observations introduce the visual conditions of robot manipulation. Each training example uses a single camera view, and the model is trained to predict future video from the observation history. The goal is to establish a visual dynamics prior for embodied interaction before training the model to generate actions.

Stage 2: Multi-view video-action training. We next connect the learned visual dynamics to actions by extending the model with embodiment-specific state and action modules. We jointly train video and action prediction on human and robot data, using multi-view observations where available to capture interactions from complementary viewpoints. Robot trajectories provide embodiment-specific actions, while human demonstrations provide tracked hand-pose trajectories as action supervision. Video and action are trained with flow-matching objectives, and the action loss is applied when valid action supervision is available. The mixture increases the contribution of YAM and dexterous-hand sources while retaining substantial egocentric human data. Joint training is intended to bring experience from these different embodiments into a shared model that can predict both the evolution of a scene and actions for the target robot.

Stage 3: Robot-upweighted video-action training. A broad mixture of human and robot data provides diverse physical experience, but the final policy must generate actions in the robot's own action space. We therefore continue joint video-action training with a stronger emphasis on robot demonstrations, particularly YAM, while retaining dexterous-hand, egocentric human, and other data sources. This reweighting focuses training on the observations and control required for robot execution while continuing to draw on the broader experience acquired in the earlier stages.

The resulting model serves as the starting point for the task-specific adaptation and RL post-training described below. Details of the data sources are provided in Sec. 4.2.

RL post-training. To explore, we replace GR00T 2's deterministic action denoising with Coefficients-Preserving Sampling (CPS) (Wang and Yu, 2025) during rollouts, injecting noise only into the action denoising process while video denoising stays deterministic. Policy updates use a group-relative objective in the style of GRPO whose baseline also remembers each context's past returns, so that groups in which every attempt succeeds, or every attempt fails, still carry a learning signal. The basis of this recipe was found by AutoRL, a research agent that iterated on the algorithm in simulation for about 35 days (Sec. F.2). Two complementary approaches are GRPO-induced self-imitation, which fine-tunes GR00T 2 on successful GRPO rollouts, and residual RL, which learns a small corrective policy on top of a frozen GR00T 2 (Sec. F.1.2). For RL without the real robot, we adapt the DreamDojo world model to the target tasks from rollouts of the initial policy and use a learned

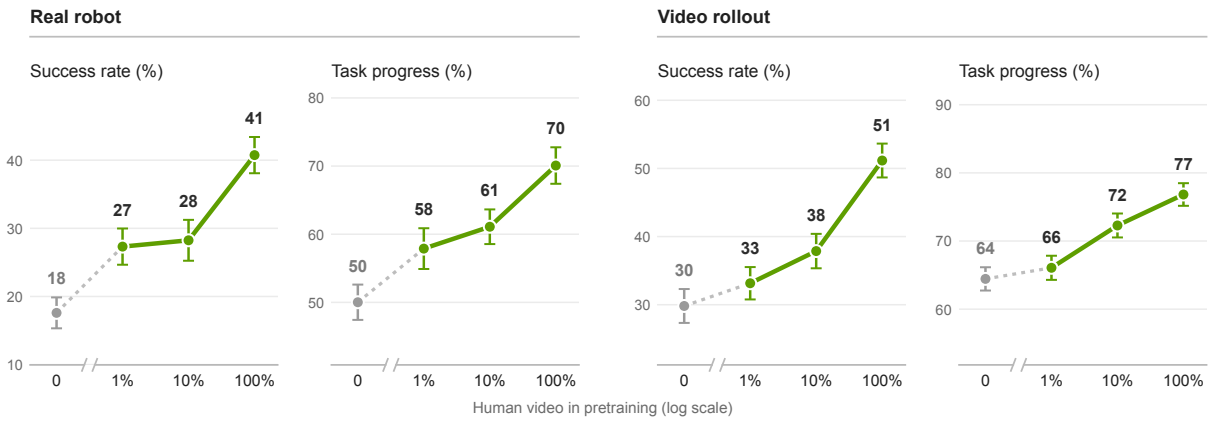


Figure 10: Scaling human video in pretraining. Human-1M data pre-training (0%, 1%, 10%, or 100%) post-trained with Dexter teleoperation data and evaluated on 72 tasks. Real Robot: 3 rollout episodes per task. An operator grades binary success and a vision-language model scores task progress. Video rollout: autoregressive video generation from 3 recorded initial configurations per task and 10 generation seeds per configuration. A vision-language model judges success and scores task progress. We report % progress and success rate with ± 1 SEM bars.

completion detector as the reward (Sec. F.5). An online infrastructure coordinates robot interaction, training, and evaluation, and retains experience across interruptions (Sec. F.3).

4.2. Data

GR00T 2 pretrains on human demonstrations, UMI demonstrations, and robot teleoperation, which play complementary roles. Egocentric human video from Human-1M provides broad coverage of physical interactions and environments, and robot performance grows with the amount of it (Sec. 4.2.1). UMI provides scalable, task-focused demonstrations with robot-compatible action representations (Sec. 4.2.2), and teleoperation grounds the model in embodiment-specific behaviors and executable actions (Sec. 4.2.3). Finally, Flora, our data engine, labels the subtask annotation of human and robot episodes at this scale (Sec. 4.2.4). App. C details the collection, processing, and curation of each source, and Sec. 4.7.3 describes the pipeline that converts raw data into training-ready format.

4.2.1. Human Data

For a world action model, egocentric human experience captures the motion and visual diversity needed to achieve strong coverage across objects, affordances, tasks, scenes and environments. GR00T 2 leverages 402,730 hours across approximately 13.75 million episodes, spanning over 51,500 unique collectors and 123,000 scenes of high quality egocentric data filtered from Human-1M. The dataset is a *1 million hour* egocentric human experience dataset of proficient human labor sensorized with egocentric cameras, dense language annotations, SLAM and 3D hand pose keypoints curated by NVIDIA, spanning demonstrations in both natural "in-the-wild" and controlled studio environments. Additional details on filters and dataset composition in Sec. C.1.

In-lab human data. To enable direct transfer to the target robot embodiment, GR00T 2 contains 3,000 hours of robot-aligned studio data, collected under a stricter protocol. The visual gap is mitigated by matching the egocentric and wrist camera viewpoints, while the motion gap is mitigated by matching demonstration speed with the robot. This data is demonstration-style where each task is demonstrated hundreds of times with a repeatable strategy (Sec. C.1.3).

Human Data Scaling. Under the same GR00T 2 architecture, training recipe and post-training teleoperation data, we ablate four Human-1M data pre-training scales (0%, 1%, 10%, or 100%). We control training length in epochs. The 1%, 10%, and 100% models start from a shared human-video pretraining checkpoint and are each pretrained for about 0.5 epoch of their share, while the 0% model skips human-video pretraining

entirely. The step from 0% to 1% therefore also includes this shared stage, and the 1%, 10%, and 100% models differ only in their share of human data and the matching training length. Here one epoch means the model passes once over every step of every 30 Hz trajectory. Every model is then trained on the same target Dexter teleoperation data, co-trained with its human share, for the same 3 epochs of that robot data. We measure real robot and video rollout success rate (%) and task progress (%) on 72 Dexter tasks, 22 out-of-distribution tasks with new kinds of instructions and 50 tasks absent from Dexter's training data. Across all four settings, we see positive scaling in performance as human data is scaled (Fig. 10). Real robot success rates increase from 18% to 41% from 0% human data pre-training to 100%. Video rollout success rates similarly increase from 30% to 51%. Together, these results show that what GR00T 2 learns from human video transfers to the robot. With the robot data held fixed, pretraining on all of the multi-view human video more than doubles real-robot success on new tasks, and none of the metrics levels off at 100%, which suggests that more human video will keep improving robot performance (Sec. C.1.5).

4.2.2. UMI Data

UMI-10K, recorded with the handheld UMI device (App. E), contributes 942k quality-controlled episodes and 10,184 hours of bimanual parallel-gripper demonstrations across 2,897 tasks. The dataset is extremely diverse with no single task contributing more than 1.2% of hours. The instructions span 356 object categories and 49 dextrous skills. The corpus covers 2,798 distinct skill-object combinations, including dynamic, articulated, and deformable manipulation that teleoperation rarely captures. Many tasks contain counterfactual pairs, which is aimed to improve policy instruction following (Sec. C.2).

4.2.3. Teleoperation Data

Cross-embodiment teleoperated demonstrations provide robot action supervision for GR00T 2. The pre-training mixture contains a total of 23.5k hours and 1.16M episodes of teleoperation data including:

- Bimanual YAM arms with parallel grippers (18.7k hours, including ABC-130k (XDOF, 2026)),
- Bimanual arms with Sharpa hands (2.9k hours),
- Bimanual arms with Wuji hands (1.3k hours),
- The Unitree G1 humanoid (275 hours),
- The single-arm Franka Panda from DROID (Khazatsky et al., 2024).

Additional details are provided in Sec. C.3.

4.2.4. Data Annotation with Flora

GR00T 2 pretraining uses dense timestamped language annotations for language conditioning, filtering and rebalancing. GR00T 2 RL post-training additionally requires process rewards, while policy evaluation requires outcome labelling. At the scale of Human-1M and our robot corpora, human labeling is too slow and frontier API models are too expensive. To provide these dense labels, we leverage Flora (Leng et al., 2026), the data engine that annotates training and evaluation data for GR00T 2. Flora consists of three models that identify action boundaries, annotate each action with a self-contained imperative instruction, and generate outcome, dense, and rubric-based rewards. On WGO-Bench (Macrodata Labs, 2026), Flora achieves a temporal segmentation F1 of 50.3, compared to 46.0 for the next-best model, GPT-6 Astra (OpenAI, 2026). Additional details are provided in Sec. C.4.

4.3. Real-World Benchmarking: Mapping the Hill We Climb

Every GR00T 2 candidate earned its place in blinded head-to-head matches on real robots, accumulated into a single leaderboard that tells us which checkpoint is better, by how much, and how sure we can be.

Why head-to-head. Real-robot evaluation is expensive, but its results accumulate, and months of evaluations are only worth keeping if they stay comparable. Absolute success rates do not: they drift with the station, the operator, the hardware, and the day, so a score from one session cannot be compared with a score from another.

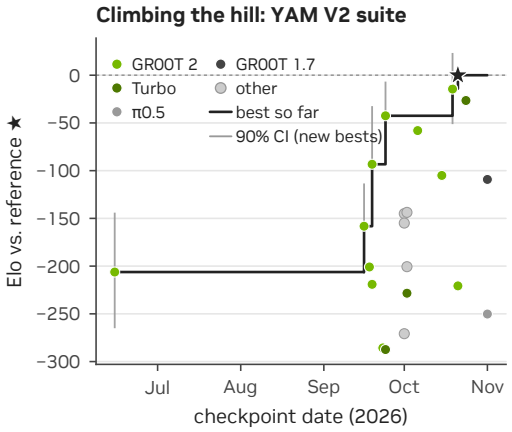


Figure 11: Mapping the hill we climb. Every checkpoint on the current YAM suite, rated relative to the A/B reference (⋆, the board's top-rated checkpoint, here a GR00T 2 Turbo checkpoint) and placed at the date it was saved. The line tracks the best checkpoint so far, and bars are paired 90% bootstrap intervals for the checkpoints that set a new best. Evidence: the 6,035 blinded A/B comparisons of this suite, from 122 campaigns.

A blinded A/B trial runs two to four checkpoints in randomized order on the same scene, by the same operator, within minutes, so the drift is shared and cancels in the comparison. GearArena, our real-robot leaderboard, pools every such trial. By the snapshot used here it held 195 A/B campaigns with 16,044 graded robot episodes and 11,767 pairwise comparisons, collected on 10 YAM stations by more than 20 operators between July and November 2026, rating 71 checkpoints.

Tasks from first principles. We design evaluation tasks by asking what the robot should be able to do, then choosing tasks that expose those capabilities, and finally gauging them on real checkpoints: a task every checkpoint completes is archived, and one no checkpoint completes is shelved until models catch up, since neither separates anything. Two suites resulted: a first suite of general and dexterous manipulation tasks and the current suite of 38 language-following and motion-primitive tasks, each run with partial-credit rubrics.

From matches to a map. Every pair of checkpoints in a trial forms one comparison, decided by the operator's explicit ranking (equal grades count as ties). A Bradley-Terry model (Bradley and Terry, 1952), fit jointly to all comparisons, gives each checkpoint an Elo rating. Within each head-to-head, every task carries equal weight, and confidence intervals come from a bootstrap that resamples tasks rather than trials. Every new checkpoint is first matched against the A/B reference, the best-rated checkpoint on the board, which keeps all ratings on one connected scale (App. D).

What it bought us. Arena turned close calls into decisions. GR00T 2 outrates $\pi_{0.5}$ by 236 Elo (95% CI 171 to 325), an expected 80% head-to-head win rate, and GR00T 1.7 by 95 Elo (CI 50 to 148), both in all 500 bootstrap resamples (Tab. 13). On our YAM V2 task suite, which focuses on foundational skills such as language following and motion primitives (Tab. 2), GR00T 2 Turbo is statistically indistinguishable from GR00T 2 in skill while running each inference about three times faster (52 vs. 171 ms per served call on a GeForce RTX 5090 workstation, Fig. 23). Over four months, the best checkpoint on the current suite climbed 206 Elo, from a June model that today's top checkpoint would beat 77% of the time (Fig. 11). Simulated evaluations reproduce Elo ratings fit to blinded A/B trials of this kind with Pearson $r = 0.93$ (Sec. 4.5.2). App. D details the task design, the rating model, the A/B infrastructure, and the full results.

4.4. Learning without Teleoperation: Devices and Pipelines

GR00T 2 learns a skill on a robot from demonstrations recorded without that robot (Sec. 3.2). In all four sources below, the demonstrations specify the task. For human motion capture and the Gripper UMI, whose devices differ from the robot, task-agnostic data from the target robot specify the embodiment, and cross-embodiment pretraining connects the two. The DexUMI exoskeleton replicates the robot hand it is built for, so on that hand its demonstrations need no robot play data (Sec. 4.4.3). Capture and conversion remove what can be removed before training, namely differences in camera geometry (Sec. C.1.3), kinematics, and action semantics

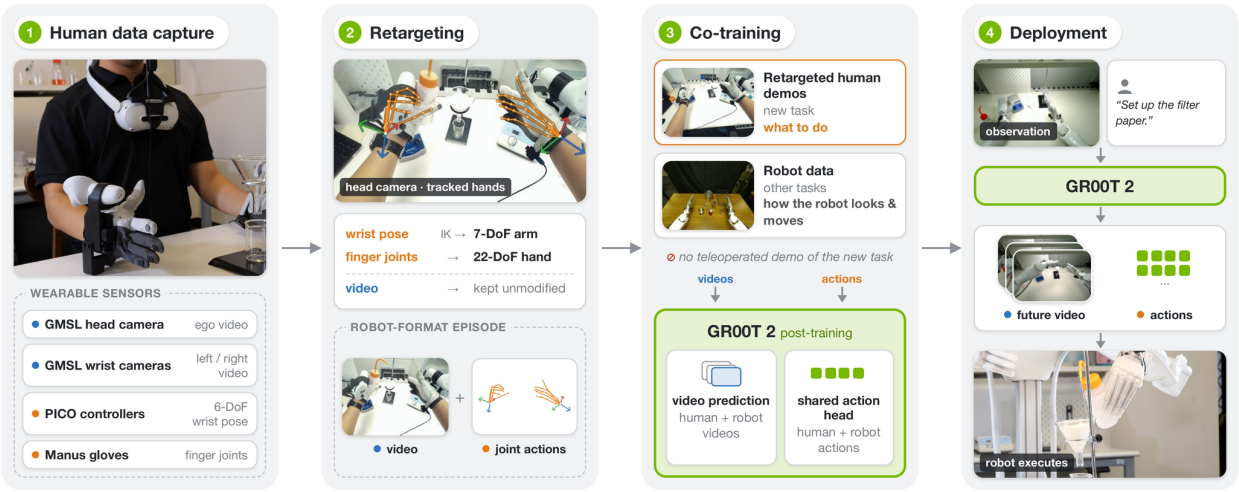


Figure 12: GR00T 2 learns new robot tasks from human demonstrations. (1) A person performs the task wearing GMSL head and wrist cameras at robot-matched poses, PICO controllers that track 6-DoF wrist poses, and Manus gloves that track finger joints. (2) Retargeting maps each wrist pose to the 7-DoF arm and each hand pose to the 22-DoF hand, and the video stays unmodified. (3) GR00T 2 co-trains on the retargeted human demonstrations and on robot data of other tasks: videos supervise video prediction, and human and robot actions train one shared action head. (4) On the robot, GR00T 2 takes the observation and the instruction, predicts a sequence of future frames, and outputs the actions the robot executes. No teleoperated demonstration of the new task enters training.

(Sec. B.1.2), and GR00T 2 bridges the rest. Because it predicts future video jointly with actions, GR00T 2 trained on such demonstrations together with robot data predicts the robot's own hands and arms performing the demonstrated task (Figs. 13 and 14).

4.4.1. Human Demonstrations with Motion Capture

A person performs the task wearing head and wrist cameras at robot-matched poses, a motion-capture glove, and a 6-DoF tracker on each wrist (Fig. 12). With the head camera at the robot's head-camera pose, the wrists are anchored in the robot base frame, and a closed-form inverse-kinematics solver (Chen et al., 2026b; Kong et al., 2026) retargets them offline to the 7-DoF arms and 22-DoF Sharpa hands, placing the robot fingertips at the human fingertips to preserve the pinch geometry. The video stays unmodified. At about 118,000 bimanual solves per second, the same solver also drives teleoperation, which collects task-agnostic play data on the target robot. One play pool, sampled 1:1 with the human demonstrations, serves every task (Sec. B.1.1 to B.1.3 and C.3.1). For tasks that only humans demonstrated, GR00T 2 then predicts the robot's own hands completing each step (Fig. 13).

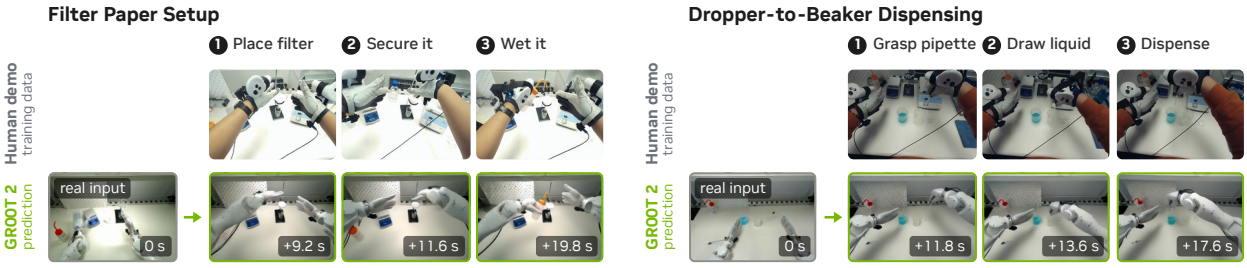


Figure 13: Human demonstrations and GR00T 2 predictions. Two tasks that only humans demonstrated. Given one robot start frame and the instruction, GR00T 2 predicts the robot's ego view. Times are measured from the start frame. Human demonstrations (top rows) appear only in training. All four tasks are shown in Fig. 28.

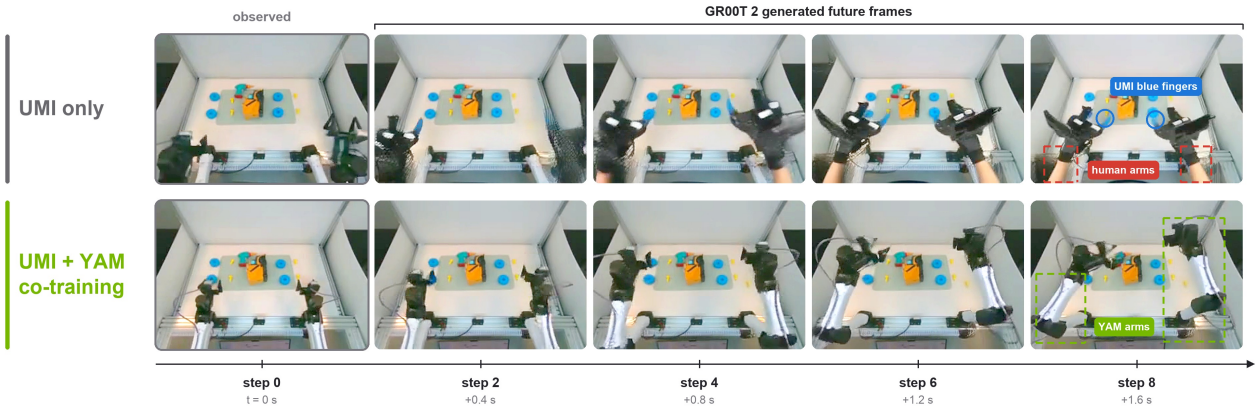


Figure 14: GR00T 2-generated future frames on YAM, without and with YAM co-training. Both models observe the same YAM top-camera frame (left) and predict eight future steps of Toy Dumptruck Assembly, a task demonstrated only with Gripper UMI. Trained on UMI data alone (top), GR00T 2 renders human arms and the blue UMI fingers entering the scene. Co-trained with task-agnostic YAM data (bottom), it renders YAM's own arms. The co-training data supply the robot's appearance as well as its action statistics.

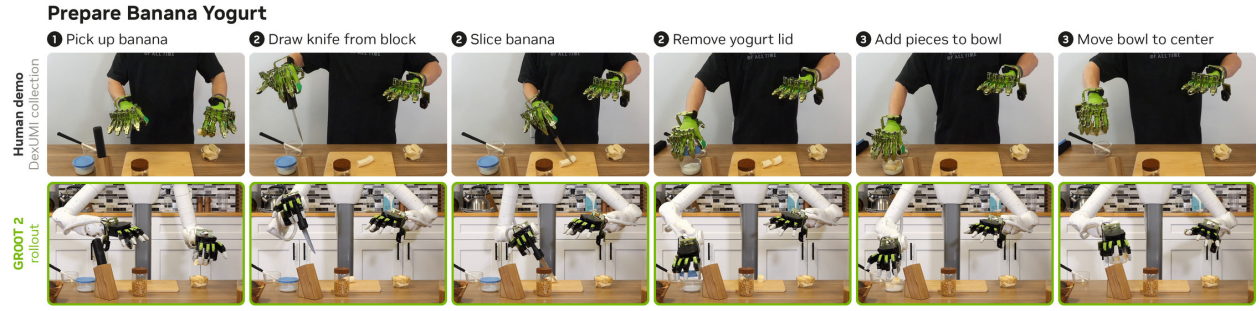


Figure 15: DexUMI demonstration and GR00T 2 rollout. Prepare Banana Yogurt, with keyframes numbered by subtask: a person demonstrating the task with the DexUMI exoskeletons (top) and a GR00T 2 rollout on a Tianji robot with XHAND1 Pro-U hands (bottom). The task is demonstrated only with DexUMI. All three DexUMI tasks are shown in Fig. 30.

4.4.2. Gripper UMI

Gripper UMI is a handheld parallel gripper in the lineage of UMI (Chi et al., 2024). We use our open-source GEAR UMI (NVIDIA GEAR Team, 2026): its robot-style wrist cameras and width encoder record human demonstrations in robot coordinates without a robot (Sec. E.1), and it also yields the UMI-10K pretraining corpus (Sec. 4.2.2). A robot end-effector that reproduces the handheld's cameras and fingers removes the sensing and gripper gaps by construction (Sec. E.2). For an off-the-shelf robot such as YAM, which shares none of them, three findings make the transfer work. (1) Co-training with task-agnostic YAM data closes the appearance and scene gap: GR00T 2 then predicts YAM's arms performing the handheld-demonstrated task (Fig. 14). (2) The model absorbs appearance but not geometry, so the fisheye wrist view is re-rendered to imitate YAM's wrist camera, and a rough match of field of view and viewing direction suffices. (3) Handheld actions are re-expressed in YAM's action space and decoded by its own action head. With cross-embodiment pretraining, 41 to 75 handheld episodes per task reach high success rates, and 10 to 20 already yield end-to-end success. The rarest subskill, not the task length, sets this number (Sec. B.2 and E.3).

4.4.3. DexUMI

DexUMI (Xu et al., 2025) extends robot-free collection from parallel grippers to five-fingered hands. We use our open-source GEAR DexUMI (NVIDIA GEAR Team, 2026): the wearable exoskeleton replicates the kinematics of the target XHAND1 Pro-U hand, with its 16 degrees of freedom mapped one-to-one onto the

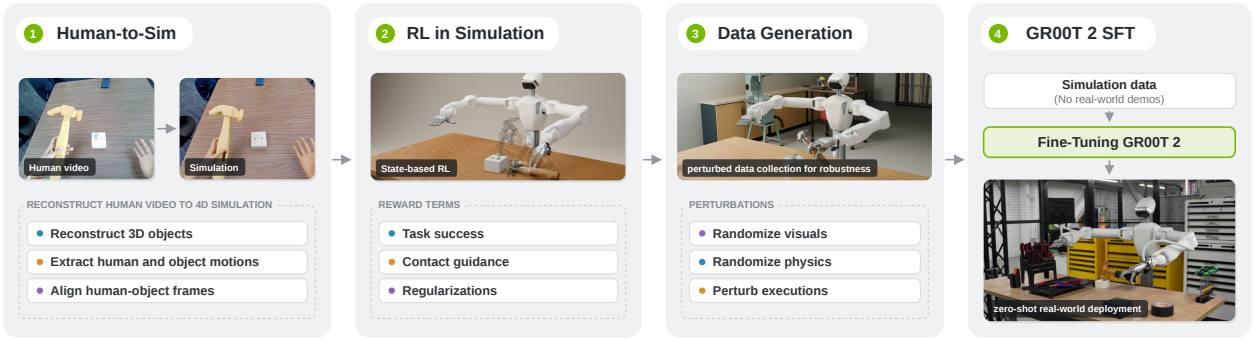


Figure 16: Human-to-sim-to-real transfer for GR00T 2. (1) Human-to-sim reconstructs assets and motions from one human demonstration. (2) RL trains a state-based manipulation policy in simulation. (3) Data collection under perturbations expands state coverage and captures recovery behavior. (4) GR00T 2 is fine-tuned exclusively on simulation data and deployed zero-shot in the real world.

robot joints, and matches the hand's contact surfaces, wrist-camera mount, and tactile format (Sec. E.4). Two choices adapt GR00T 2 to these data. First, the exoskeleton is handled in training rather than by editing video: its green parts are segmented by color, and their appearance is randomized consistently in the conditioning and predicted frames, which removes the earlier step of replaying each demonstration on a robot hand for inpainting (Sec. E.6). Second, following the Adaptive Compliance Policy (Hou et al., 2025), fingertip forces are projected onto the hand joints as torques and added to the action. At execution, the predicted torque lowers each joint's stiffness and offsets its position target, so that the hand applies the predicted force and remains robust to positional uncertainty (Sec. B.3). Because the exoskeleton replicates the hand, no robot play data are needed: post-trained on DexUMI demonstrations alone, GR00T 2 performs long-horizon tool use with both hands (Fig. 15).

DexUMI demonstrations can also be converted for a hand that the exoskeleton does not replicate. For the 22-DoF Sharpa hand, we compute the XHAND fingertip positions from the recorded joint angles and retarget them with the closed-form solver used for glove teleoperation (Sec. B.1.2). As with Gripper UMI, we warp each DexUMI wrist frame to match the Sharpa wrist camera, which sits at a different pose. The demonstrations are co-trained with Sharpa play data in the robot's action space and with its normalization statistics (Sec. B.4).

4.4.4. Human-to-Sim-to-Real

Contact-rich skills such as in-hand reorientation and forceful tool use remain difficult for WAMs to learn, and recovery behaviors are difficult to demonstrate. Reinforcement learning in simulation supplies them from a single human video (Fig. 16). The NVIDIA Video to Data pipeline (NVIDIA, 2026b) reconstructs the object assets and the hand and object motion. A state-based RL expert, trained with CHORD (Zhu et al., 2026) under identified robot dynamics and randomized physics, learns to track the object motion and complete the task. Its rollouts become image-based training data, perturbed by random action bursts and teleports so that they contain corrections, and rendered in Isaac Lab with randomized appearance, lighting, and camera calibration. GR00T 2 is fine-tuned on 1.5k such episodes per task with training-time real-time chunking (Sec. 4.6.2) and deployed on the real robot without any real-world robot data (Sec. B.5).

4.5. Simulation and Evaluation

Simulation serves GR00T 2 in two roles. Physics simulation of digital twins of our real-world tasks provides a fast, repeatable, and calibrated proxy for real-robot evaluation, which we use throughout development (Sec. 4.5.1 and 4.5.2). Neural simulation with a learned world model, adapted from recorded rollouts, provides an environment for reinforcement learning (Sec. 4.5.3). We also evaluate GR00T 2 on five public simulation benchmarks (Sec. 4.5.4).

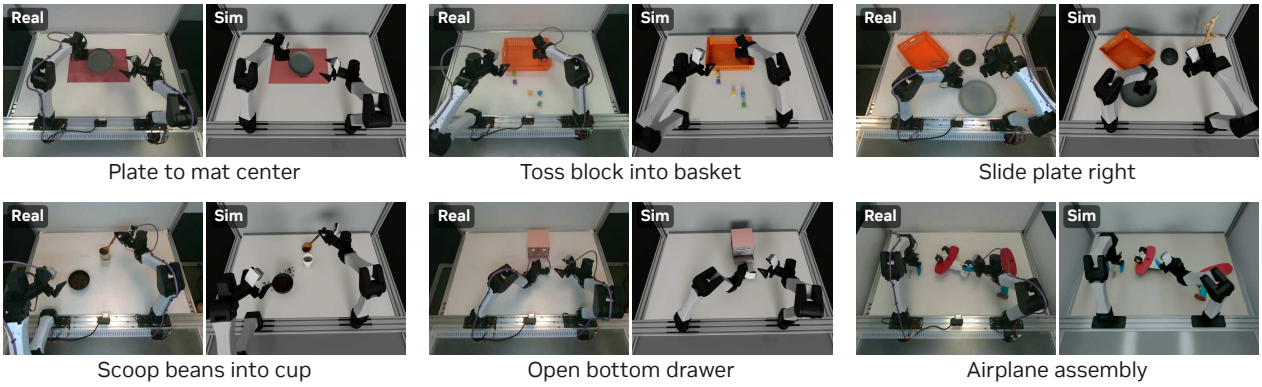


Figure 17: Real-world tasks and their simulated counterparts. Each pair shows the real workcell's top camera (Real) and the same moment when the recorded robot commands are replayed in simulation (Sim), for five general manipulation tasks and one stage of the multi-stage airplane assembly. More tasks are shown in Fig. 61.

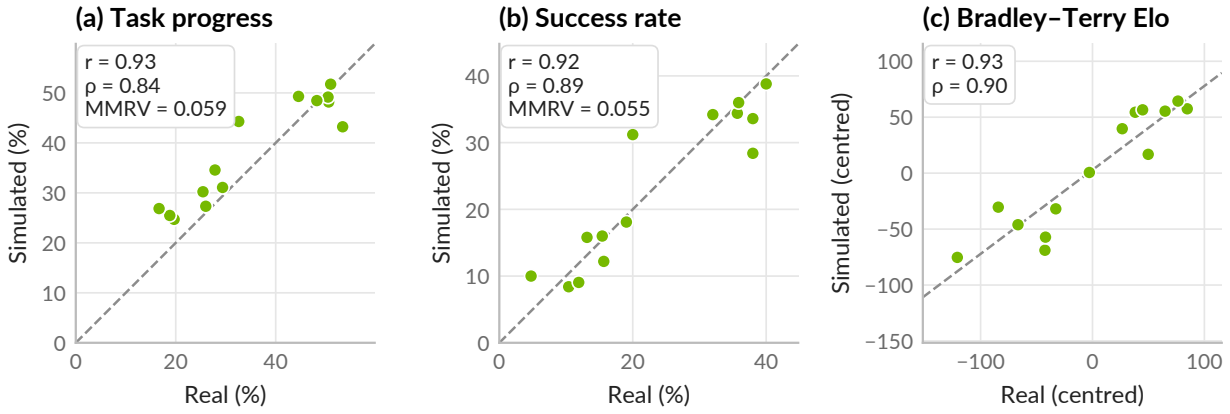


Figure 18: Simulated vs. real-world performance per policy. (a) Task progress (mean rubric score) and (b) success rate, averaged over 25 tasks (dashed: y = x). (c) Bradley-Terry Elo ratings fitted separately to pairwise outcomes in simulation and in real-world A/B evaluations (dashed: least-squares fit, since the two Elo scales differ). r: Pearson, ρ: Spearman, MMRV: mean maximum rank violation. Each point is a generalist checkpoint of GR00T 2 trained for the YAM embodiment.

4.5.1. Real-to-Sim Task Construction

We recreate in simulation the real-world tasks used to develop GR00T 2, from general manipulation tasks evaluated zero-shot to multi-stage assembly, with rigid and articulated objects, deformable materials, and particles (Fig. 17). The aim is to reproduce both the visual setup and the physical interactions that determine whether a task succeeds. Agent-assisted workflows build the environments: agents use inputs such as real robot videos, 3D scans, and photographs, along with tools from SimFoundry (Ranawaka et al., 2026), to handle scene and asset reconstruction, including geometry, appearance, and collision modeling. Human experts review the resulting content and provide feedback to guide refinement. The simulation environments run in Isaac Lab Arena (NVIDIA Isaac Lab-Arena Contributors, 2025) on Newton (The Newton Contributors, 2025) with the MuJoCo Warp solver (Google DeepMind and NVIDIA, 2025). Environments are validated through several means including grasp and drop tests (for assets), replaying real robot demonstrations, and for assembly environments, testing assembly without a robot (Sec. J.1). For the humanoid, a separate motion-centric pipeline turns teleoperated whole-body motions into simulated demonstrations (Sec. 3.5).

4.5.2. Simulation as a Proxy for Real-World Evaluation

Policies trained on real data are evaluated in these environments directly, without fine-tuning or co-training on simulation data. On 25 YAM tasks, 14 GR00T 2 checkpoints evaluated in both settings, with 5 blinded A/B

trials on the real robot and 10 simulated trials per task, give Pearson correlations of 0.93 for task progress, 0.92 for success rate, and 0.93 for Bradley-Terry Elo ratings. These values are 95 to 99% of the correlation that a perfect simulator would reach with the same number of trials (Figs. 18 and 65). Environments reconstructed from DROID setups reach 0.98, 0.95, and 0.97 for five generalist checkpoints. This calibration makes simulation a development tool: during pretraining, simulated evaluations tracked intermediate checkpoints and narrowed the final model candidates to three, which we then compared on real robots. We tested changes to the data pipeline, such as sequence packing and mixed-shard sampling, with small-scale training and closed-loop simulation before adopting them in pretraining. Inference settings were screened in simulation before robot tests. Beyond success rate, simulation testing revealed how inference settings affect motion smoothness, and we confirmed this finding in physical trials. Simulation also served to develop the agent's recovery skills on the toy plane assembly task (Sec. 3.4). While physical-robot evaluation remains the final validation for every design choice (Sec. J.2), simulation evaluation significantly increased development speed and iteration frequency.

4.5.3. Neural Simulation

Physics simulation requires reconstructing each task, whereas a learned world model can be built from recorded robot interaction. We turn DreamDojo (Gao et al., 2026), a world model initialized from pretrained video priors, into an environment in which GR00T 2 collects RL experience (Sec. 4.1). DreamDojo is first mid-trained on multi-view robot episodes, 8,310 from YAM (about 227 hours, including failed attempts and recoveries) and 57,773 from DROID (about 273 hours), with the camera views tiled into one video canvas so that they are predicted jointly. It is then adapted to the target tasks from a few hundred successful and failed rollouts of the initial policy per task. In closed loop, GR00T 2 predicts an action chunk, the world model predicts the multi-view frames that result from executing it, and these frames become GR00T 2's next observation. The world model carries its history in latent space, and the last commanded action stands in for the robot state. Because each episode chains many short predictions, the world model conditions on its own imperfect outputs. For the model used in RoboLab, perturbing the conditioning latents with noise during training exposes it to such histories. Rewards come from compact video classifiers that detect task success, trained offline on labels that a vision-language model proposes and humans review. A sparse success reward is used because dense progress scores are expensive to annotate and can be exploited by the policy. RL in this environment raises the success of GR00T 2 from 43% to 72% on average over four real-world tasks (peg insertion, rack-hole insertion, and bun placement on two cups; Fig. 52) and from 51% to 71% on ten RoboLab tasks, both measured outside the learned environment (Sec. F.5).

4.5.4. Simulation Benchmarks

We evaluate GR00T 2 on five public benchmarks with their official task sets, episode budgets, and success criteria, using for each a single checkpoint post-trained from the same pretrained model (Sec. J.3). Baseline results are taken from the official leaderboards as of November 6, 2026, except GR00T 1.7, which we retrain and evaluate ourselves. Among VLA and WAM policies on the public leaderboards, GR00T 2 achieves the highest success rate on RoboCasa365 (Nasiriany et al., 2026), a single-arm mobile manipulation benchmark (67.2%, 2.8 points above the best prior entry), and on RoboDojo (Chen et al., 2026a), a bimanual benchmark (35.5%). On RoboLab (Yang et al., 2026) and MolmoSpaces (Kim et al., 2026), it places third (47.4%) and second (45.1%) after post-training only on real DROID data, so both evaluations are zero-shot real-to-sim transfer. On the G1 Factory humanoid benchmark (NVIDIA, 2026c), it reaches 84.7% success against 69.4% for GR00T 1.7. With RL post-training and agentic execution, GR00T 2 ranks first on the RoboCasa365 and RoboDojo leaderboards, the latter evaluated on one of its three official layout banks (Sec. 3.4 and Tab. 19).

4.6. Deployment and Real-Time Inference

Deployment requires timely predictions, continuous motion, and efficient task completion, both on workstation GPUs and on edge hardware with limited compute, memory, and power. Compared with VLA policies, these requirements are more difficult to meet for a world-action model: GR00T 2 encodes multi-camera video and

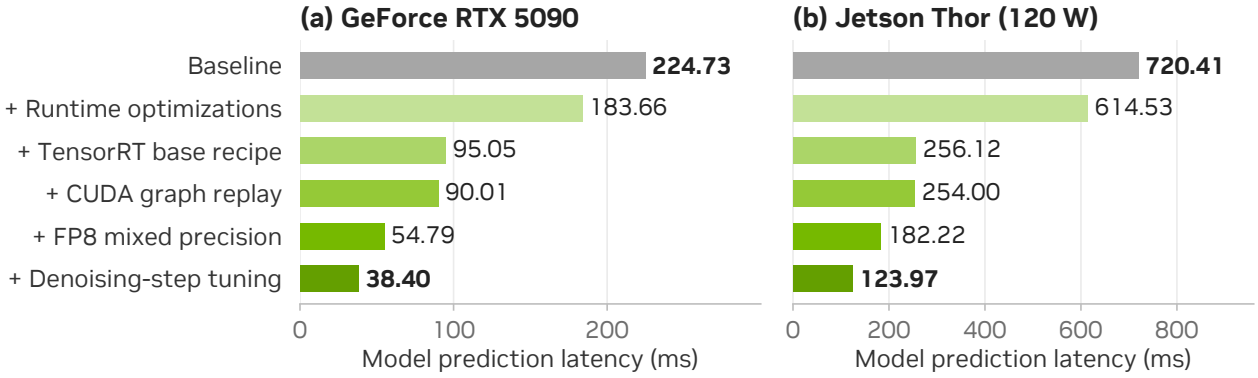


Figure 19: Model-prediction latency for GR00T 2 Turbo on G1 using (a) a GeForce RTX 5090 and (b) Jetson Thor in 120 W mode. Bars show medians of 500 predictions after 20 warmup requests, using the same checkpoint weights and input observations. The panels use independent horizontal scales. Denoising-step tuning changes 2v4a to 1v2a.

iteratively denoises video and action latents, and in our setting it incurs higher inference latency. We address them with accelerated inference (Sec. 4.6.1), training-time real-time chunking for smooth execution under the remaining delay (Sec. 4.6.2), and phase-conditioned action chunk transforms that enables faster than demonstration task completion throughput (Sec. 4.6.3).

4.6.1. Policy Serving and Accelerated Inference

GR00T 2 is served through a client-server interface: a robot client sends the camera history, robot state, and instruction, and the server returns an action chunk, which the client buffers and executes at the controller's rate while it requests the next one (Sec. K.5). The server first removes repeated computation. It reuses the text features while the instruction is unchanged, precomputes the timestep conditioning, and lets action-only denoising steps reuse the attention keys and values of the latest video step. It then compiles the video encoder and the transformer into TensorRT engines replayed with CUDA graphs, for supported checkpoints applies FP8 to selected projection and feed-forward layers after calibration on robot observations, and reduces the number of denoising steps where task success and smoothness allow. On the serving path, streaming and prefetching move video encoding ahead of the prediction request. For GR00T 2 Turbo on the G1, TensorRT compilation gives the largest single reduction on both GPUs (Fig. 19). Together, these optimizations reduce median prediction latency by $2.9\times$ to $7.2\times$ relative to eager BF16 execution of the same checkpoint (Tab. 27). On a DGX Station GB300, GR00T 2 predicts in 48 ms on Dexter and 54 ms on YAM, with video encoding prefetched. For the Unitree G1, GR00T 2 Turbo predicts in 38 ms on a GeForce RTX 5090 and in 124 ms on the robot's onboard Jetson Thor, where it plans 1.6 s chunks of whole-body motion tokens about once per second for the SONIC controller to execute at 50 Hz (Fig. 9). Numerical comparisons with eager BF16 execution help identify changes in predicted actions, but prediction error alone does not establish task success in closed-loop control. We make deployment decisions to screen inference changes and make deployment decisions through evaluation on the physical robot, considering task success, completion time, and motion quality together (Sec. I.1).

4.6.2. Smooth Execution with Real-Time Chunking

Some delay remains after acceleration, and the robot must keep moving while the next chunk is computed. Waiting for each prediction interrupts the motion, and executing predictions asynchronously without accounting for the delay creates discontinuities between successive chunks. Training-time real-time chunking (RTC) (Black et al., 2025) conditions each prediction on the prefix of actions already committed for execution. In our setting, conditioning on a clean prefix improves continuity but lowers task success at the shorter delay. We therefore use noisy-prefix RTC: during fine-tuning, the conditioning prefix is perturbed with Gaussian noise, which is intended to reduce reliance on the previous plan while retaining the context needed for a smooth continuation,

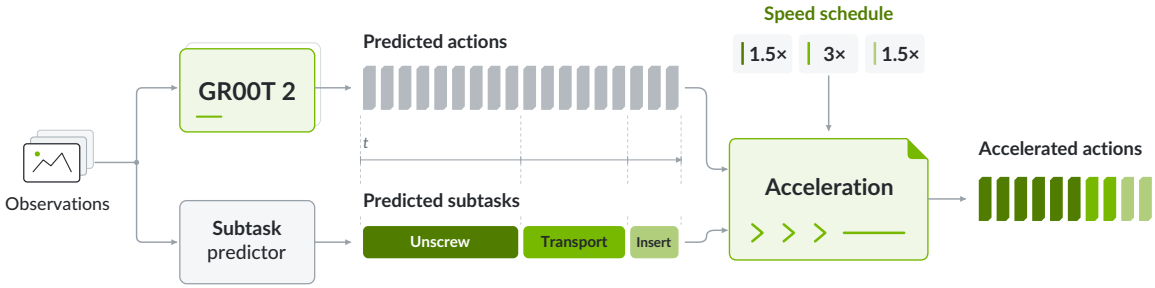


Figure 20: Phase-conditioned task-completion throughput acceleration of GR00T 2. GR00T 2 predicts a future action chunk, while a subtask predictor identifies its task phases. Using a phase-specific speed schedule, we transform the original action chunk to accelerate execution and boost task-completion throughput while preserving success rates.

and at inference the policy conditions on the clean committed actions. A short fine-tuning stage of 5k steps on a post-trained policy captures most of the continuity benefit, so RTC does not require repeating the full training pipeline. On eight RoboDojo precision tasks with simulated inference delays, noisy-prefix RTC raises success over asynchronous execution of the same weights from 32.8% to 34.8% at a 160 ms delay and from 15.0% to 21.4% at 320 ms. It reduces peak commanded jerk by 69% at 160 ms and the 95th-percentile cross-chunk jerk by 87% at 320 ms, whereas clean-prefix RTC lowers success at 160 ms to 30.7% (Sec. I.2).

4.6.3. Accelerating Task Throughput Beyond Demonstrations

Continuous execution and low latency do not by themselves boost task completion throughput, which remains bound to the pace of the demonstrations. We therefore retime GR00T 2's action chunks by task phase. A task is decomposed into semantically meaningful phases, such as reaching, grasping, transporting, and screwing, which tolerate different degrees of acceleration. An agent searches for a per-phase speed schedule by evaluating accelerated rollouts and refining the schedule after observed failures. At deployment, a lightweight subtask predictor, trained on phase annotations generated by an agent, predicts from the current observation the phases spanned by each action chunk, and the chunk is retimed according to the schedule (Fig. 20). Because retiming changes the timing between successive chunks, temporal chunk alignment and cross-chunk jerk smoothing keep the execution continuous (Sec. I.3).

4.7. Infrastructure

Training GR00T 2 runs a joint video-action model over multi-camera video from human, UMI, and robot sources. Its infrastructure must provide the memory and compute for this model at scale, deliver decoded video windows to every GPU without stalling synchronous training, and turn raw recordings into training-ready data at the scale of the pretraining corpus. We describe the pretraining compute and distributed training runtime (Sec. 4.7.1), the training data loader (Sec. 4.7.2), and the multimodal data processing pipeline (Sec. 4.7.3). Policy serving is described with deployment (Sec. 4.6.1), and the online RL infrastructure with RL post-training (Sec. 4.1).

4.7.1. Pretraining Compute and Distributed Training

GR00T 2 is pretrained on NVIDIA GB300 NVL72 systems, chosen for GPU memory capacity, Tensor Core throughput, and acceleration of collective communication. Each Blackwell Ultra GPU provides up to 288 GB of HBM3e with 8 TB/s of bandwidth and about 2.5 PFLOPS of dense BF16/FP16 Tensor Core compute. A rack connects 72 GPUs into one NVLink domain with 1.8 TB/s of bidirectional bandwidth per GPU, in which NVLink SHARP offloads reductions to the switches, and racks communicate over a RoCE fabric with one 800 Gb/s ConnectX-8 SuperNIC per GPU. The cluster has 2,016 GPUs in 28 racks, and the main pretraining run used 1,024 of them, while the rest ran other training experiments concurrently (Sec. K.2.1).

Training uses synchronous data parallelism with DeepSpeed ZeRO-2: every GPU holds the full weights, gradients and optimizer state are partitioned across ranks, and BF16 gradient synchronization overlaps with computation. Because the model fits without pipeline parallelism, the GPU memory keeps the parameters, each rank's optimizer partition, and the activations of a per-device batch of 16 resident, and the large per-device batch gives the Tensor Cores more matrix-multiplication work and amortizes kernel-launch overhead. Part of each step runs in frozen feature encoders rather than the trainable model. Depending on the recipe, the runtime uses TensorRT for video encoding, fused text-encoder operations, efficient attention kernels that keep the model's masking rules, PyTorch compilation, and activation recomputation, and it can encode the next batch's features while the current batch synchronizes gradients and updates the optimizer. We judge efficiency across the whole step, from input retrieval and feature encoding to gradient synchronization and the optimizer update. A hardware-counter-based utilization metric, Overall FLOP Utilization (Pedersen et al., 2026), is monitored alongside step time, but elapsed time and model quality decide whether a change is kept, since an optimization can shorten the step without moving the utilization signal (Sec. K.4).

4.7.2. Training Data Loading

Training data are stored in LanceDS, built on the Lance columnar format (Pace et al., 2025; Lance Contributors, 2026). Each dataset holds a step table with state, actions, annotations, and quality-control (QC) signals, an episode table, and a media table with the original videos. Lance versioning preserves earlier snapshots when episodes are added or annotations and QC signals change, and new QC columns are added without rewriting existing ones. A training run pins one snapshot per dataset, so its data, normalization statistics, and metadata are reproduced together. All datasets share one schema, and the recipe's embodiment tag selects at read time the state and action groups, cameras, annotations, and normalization statistics of that embodiment, as well as the model's embodiment-specific projection layers. QC rules are not applied in the loading path: they are evaluated offline over the temporal window that each training sample reads, its observations and its action targets, and stored as views of the eligible sample starts. The views are rebuilt with every new snapshot and carry normalization statistics computed over the samples they keep. Sampling is drawn on demand from deterministic per-rank seeds and mixes shards so that each global step follows the recipe's dataset proportions. Batch preparation runs in three asynchronous stages: video shards are prefetched from shared storage to node-local NVMe, frames are decoded (about 90% of preparation time in a three-view shard benchmark), and samples are augmented, collated, and copied to the GPU from pinned memory. Variable-length token packing reduces padding. Over 100 steady-state pretraining steps on 1,024 GB300 GPUs, reading 25 dataset stores with 15.6 million episode records and 46.2 billion time steps, the loader delivered about 12,000 sample windows per second, and the mean exposed wait for input was 0.37 ms (0.61 ms at the 99th percentile), or 0.049% of the measured step time (Tab. 35). These numbers characterize input delivery, not the model's FLOP utilization. In the measured run, packing filled 94.5% of the positions in 32,768-token rows. Loader changes are validated in closed-loop simulation before adoption (Sec. 4.5.2), and Sec. K.4.3 details the loader.

4.7.3. Multi-Modal Data Processing

Before they can be loaded, raw human demonstration recordings are converted into training-ready data. Multi-camera video is retrieved, decoded, and resized. Motion records are transformed and, where needed, retargeted to the training hand representation. Observations, actions, and annotations are then aligned in time and packaged as indexed episodes in training-ready shards. The pipeline processes petabyte-scale input each month, with individual production jobs using nearly 10,000 vCPUs. The central difficulty is that these conversions share inputs and compete for limited working storage. A long video can contain many episodes, and one episode can depend on several camera files and separately stored motion records, so a shared input can be reclaimed only after its slowest consumer finishes, and a slow conversion or upload can block the retrieval of unrelated inputs and leave workers idle. The pipeline therefore ties the lifetime of each input to execution progress. Files stream from object storage directly to the processing nodes, without staging on a shared filesystem, so conversion starts before a collection has been fully retrieved. New inputs are admitted

against each node's storage budget, counting resident files, pending transfers, and output headroom, and work is dispatched to the node that holds its inputs. An episode releases its claim on an input only after its outputs are uploaded, so the completion of consumers, rather than prefetching, governs when new inputs enter. After an interruption, lost inputs are rebuilt through the same admission path, and durable checkpoints let the resumed job reuse completed conversions and published shards. In three production jobs on 100 nodes with 96 vCPUs each, the pipeline processed 722 to 780 demonstration-hours of recordings per hour, published the first training-ready shard within 28 minutes, and averaged 97.8% CPU utilization during conversion. Across five interrupted jobs, 92.4% of the completed conversion time went to work that did not have to be repeated, and publication resumed 2.4 to 5.3 minutes after a restart (Sec. K.6 and Tab. 36).

5. Conclusion and Open Problems

GR00T 2 is a world-action model: a single 2B-parameter transformer that predicts future observations and robot actions together, pretrained on egocentric human video, handheld UMI demonstrations, and teleoperation of gripper arms, dexterous hands, and a humanoid. Predicting the future gives all of these sources a common objective, including human video that carries no robot actions. On 50 dexterous tasks new to the robot, the resulting model succeeds zero-shot in 40% of trials and reaches 95% with five demonstrations (Sec. 3.1). It learns robot tasks without teleoperated demonstrations of those tasks (Sec. 3.2). Real-world RL raises its success on contact-rich skills to 88 to 98% (Sec. 3.3). With corrective data and a frontier agent, it completes cluttered long-horizon assemblies in at least 97% of trials (Sec. 3.4). GR00T 2 Turbo runs onboard a humanoid and directs a learned whole-body controller (Sec. 3.5).

Three observations from building GR00T 2 shape the problems that follow.

- *Much of what the robot learns need not come from robot data.* Robot success on new tasks more than doubles with human video in pretraining and does not level off at the largest scale we tested (Fig. 10). For glove and Gripper UMI demonstrations, task-agnostic data from the target robot specify the embodiment, while the demonstrations specify the task (Sec. 3.2).
- *Reliability comes from complementary mechanisms, not from pretraining alone.* Pretraining provides breadth. Corrective demonstrations and RL close gaps in precision and contact. The agent supplies memory, verification, and recovery over long horizons.
- *Progress depended on measurement.* Blinded head-to-head evaluation and calibrated simulation (Sec. 4.3 and 4.5.2) turned close comparisons into decisions.

Each observation also points to problems that remain open. We group them by part of the robot learning system, and start each group from the basic constraint that makes it hard.

Model. A robot policy learns from data dominated by observation rather than action, and it must act at the rate of the physical world. Joint video-action prediction turns every video into supervision, but most of the tokens it predicts encode future video, much of which does not bear on the next action. GR00T 2 Turbo predicts a spatial latent grid four times smaller than that of GR00T 2. After post-training, it loses 6.7 points of success on YAM but half of its success on dexterous tasks, falling from 68.6% to 34.3% (Tab. 1). The comparison does not isolate the cause, but it suggests that the visual detail a policy must predict depends on the task and the embodiment. Four questions follow.

- *What to predict.* It is unknown which aspects of the future a policy should predict, and at what resolution and horizon. It is also unclear whether prediction helps mainly as a training signal or also as imagination at inference time.
- *Memory.* GR00T 2 observes 0.8 s of history, and tasks that require memory are delegated to the agent (Sec. 3.4). In imitation learning, longer raw context can invite shortcuts such as copying past actions (de Haan et al., 2019; Wen et al., 2020), which suggests that memory should be structured rather than simply longer.
- *Force and touch.* GR00T 2 handles force only as an output: from DexUMI demonstrations, it predicts

joint torques that set the hand's compliance (Sec. 4.4.3). It does not take touch as input, and tactile recordings are scarce compared with video. How to add force and touch as inputs without eroding visual priors, and at which control rate, is open.

- *Interfaces in the control hierarchy.* Image-space points tell GR00T 2 where to act but not how, so requirements such as pouring or scooping still rest on language and learned skills (Sec. 3.4). Below GR00T 2 Turbo, the interface to SONIC is a motion latent decoded into joint position targets (Sec. 3.5), so contact force is not represented explicitly. Which representation should pass between reasoning, policy, and controller, and how it transfers to new embodiments, is unsettled.

Data. Every hour of robot data requires a robot and an operator, whereas human experience is abundant but recorded in a different body. GR00T 2 pretrains on about 400,000 hours of human video, 10,000 hours of UMI demonstrations, and 23,500 hours of teleoperation (Sec. 4.2). Human video brings coverage, UMI brings task-focused demonstrations in a robot-compatible format, and teleoperation grounds the embodiment.

What is missing is an exchange rate between these sources. It is not known how much robot performance an hour of each source yields, or how that yield changes with the data already present. It is also not known how little target-robot data suffices to specify a new embodiment. Simulation raises the same question. Compute widens coverage, but agreement with the real world still requires measurement and robot time. The scaling with human video shows only that one of these rates is positive (Fig. 10). Until they are measured, collection budgets cannot be allocated on principle.

A second question is what makes data useful beyond its quantity. Demonstrations show mostly successful behavior, whereas RL, world models, and verifiers also need failures and recoveries; the world model we use for RL is trained on robot episodes that include both (Sec. 4.5.3). Labels of how well each demonstration was executed could serve as a further training signal.

Evaluation. A success rate measured on real robots is a binomial estimate, and its uncertainty shrinks only with the square root of the number of trials. Our post-training benchmarks use up to 20 trials per task (Tab. 1). That is enough to resolve the large gaps between GR00T 2 and its baselines, but not the differences that matter in deployment. Distinguishing 95% from 99% success takes about 285 trials per policy, and 99% from 99.9% more than 1,000.[1] Absolute rates also drift with station, operator, and day, which is why we rank checkpoints by blinded head-to-head comparisons (Sec. 4.3). Even then, the independent units are tasks rather than trials, and 30 to 35% of comparisons are ties (Sec. D.7). Three directions follow.

- *Proxy evaluation at deployment scale.* Across 25 YAM tasks, simulation predicts the real-robot task progress of 14 generalist checkpoints with a Pearson correlation of 0.93 (Sec. 4.5.2). It remains to be shown whether simulation is as predictive for contact-rich assembly, for RL-trained policies that exploit simulator errors, and for comparisons between training recipes rather than checkpoints.
- *Automated verification.* In our long-horizon assemblies, three of the eight failures were screws left partly driven that both GR00T 2 and the agent judged complete (Sec. G.1). Verifying subtle physical state is a prerequisite for autonomous evaluation, for RL rewards, and for deciding when to ask a person for help.
- *Cost to reliability.* Robot learning rarely reports the quantity that determines deployment: the robot time and human time needed to bring a new task to a target reliability. The same accounting, under matched budgets, would settle when RL is worth more than additional demonstrations or corrections.

Hardware. A model can learn only what the hardware allows people to demonstrate, sensors to record, and robots to repeat. Our teleoperation cannot demonstrate fast, targeted motions such as the conveyor toss, which GR00T 2 learned from human demonstrations instead (Sec. 3.2). Capture devices that match the robot shrink the gap the model must bridge. DexUMI replicates the joints and contact surfaces of the robot hand, so on that hand its demonstrations need no robot play data (Sec. 4.4). This argues for designing robots and capture devices together, with shared kinematics, camera placement, and tactile formats.

[1] Two-sided two-proportion z-test at significance level 0.05 with 80% power.

Learning from interaction makes further demands. Physical interaction and scene resets limit how fast real-world RL collects experience (Sec. F.3), so autonomous improvement needs stations that reset themselves, detect success, and withstand thousands of contact-rich attempts. Onboard compute and power, finally, bound the model itself: GR00T 2 Turbo exists because a humanoid must run its policy on an edge GPU (Sec. 2).

Infrastructure. Joint video-action training processes multi-camera video at every step, and video dominates the cost of storage, decoding, and computation. Pretraining GR00T 2 used 1,024 GB300 GPUs and a data pipeline that processes petabyte-scale input each month (Sec. 4.7). It is not yet known how world-action models scale with model size, data, and compute, or how compute should be divided between video and action prediction. Scaling laws of this kind, under the constraint that the deployed model must run in real time, would guide the design of the next model.

The larger shift is from offline to online learning. RL post-training already couples robots, training, and evaluation in one service that retains experience across interruptions (Sec. F.3). An agent found the basis of our RL recipe through about 35 days of autonomous experimentation in simulation (Sec. 4.1). Extending this loop from single stations to fleets requires infrastructure in which robot experience, simulation, and world-model rollouts feed training continuously while every model and dataset version stays reproducible.

Deployment. The value of a deployed robot is set by how often it needs a person, how fast it works, and whether it is safe. GR00T 2 predicts an action chunk for Dexter in 153 ms on a GeForce RTX 5090 workstation GPU, and GR00T 2 Turbo predicts one for the G1 humanoid in 124 ms on its onboard Jetson Thor (median prediction latency, Sec. 2 and Tab. 27). Phase-conditioned retiming shortens mean completion time by 15 to 26% on three dexterous tasks (Tab. 29). Reliability is less settled. On long-horizon tasks, post-training reaches about 70% average success (Tab. 1). From cluttered starts, assembly success of at least 97% required corrective data and an agent that monitors and repairs (Sec. 3.4). That agent is slow relative to the policy. In visually steered trash sorting on YAM, a median agent turn takes 7.5 s against about 0.6 s per unaccelerated policy call, and the robot waits for the agent during a median 38% of an episode (Sec. G.2). Three problems stand out.

- *Routine improvement.* RL and world-model training still require rollouts and adaptation for each task (Sec. 4.5.3). A loop in which deployment exposes failures, agents propose corrections, and successful behaviors are distilled back into the policy (Sec. G.3) would turn this per-task cost into a by-product of operation.
- *Latency across the hierarchy.* Reasoning, verification, and recovery must move closer to the policy's control rate, through faster agents, distilled verifiers, or a policy that absorbs recurring corrections.
- *Safe physical action.* Safety training of language models constrains what they say, and it is not known to carry over to what a robot does. A policy that agents can steer needs its own checks on the physical consequences of each action, at the rate at which actions execute.

GR00T 2 shows that one model, pretrained to predict how the physical world unfolds, can learn from human experience, transfer skills across embodiments, and serve as the manipulation layer of an agentic system. The open problems above share one structure. Each asks how to turn a scarce resource, whether robot hours, human time, or onboard compute, into reliable physical capability, and how to measure the result with enough confidence to act on it.

References

NVIDIA, Arslan Ali, Junjie Bai, Maciej Bala, Yogesh Balaji, Aaron Blakeman, Tiffany Cai, Jiaxin Cao, Tianshi Cao, Elizabeth Cha, Yu-Wei Chao, Prithvijit Chattopadhyay, Mike Chen, Yongxin Chen, Yu Chen, Shuai Cheng, Yin Cui, Jenna Diamond, Yifan Ding, Jiaojiao Fan, Linxi Fan, Liang Feng, Francesco Ferroni, Sanja Fidler, Xiao Fu, Ruiyuan Gao, Yunhao Ge, Jinwei Gu, Aryaman Gupta, Siddharth Gururani, Imad El Hanafi, Ali Hassani, Zekun Hao, Jacob Huffman, Joel Jang, Pooya Jannaty, Jan Kautz, Grace Lam, Xuan Li, Zhaoshuo Li, Maosheng Liao, Chen-Hsuan Lin, Tsung-Yi Lin, Yen-Chen Lin, Huan Ling, Ming-Yu Liu, Xian Liu, Yifan Lu, Alice Luo, Qianli Ma, Hanzi Mao, Kaichun Mo, Seungjun Nah, Yashraj Narang, Abhijeet Panaskar, Lindsey Pavao, Trung Pham, Morteza Ramezanali, Fitsum Reda, Scott Reed, Xuanchi Ren, Haonan Shao, Yue Shen, Stella Shi, Shuran Song, Bartosz Stefaniak, Shangkun Sun, Shitao Tang, Sameena Tasmeen, Lyne Tchapmi, Wei-Cheng Tseng, Jibin Varghese, Andrew Z. Wang, Hao Wang, Haoxiang Wang, Heng Wang, Ting-Chun Wang, Fangyin Wei, Jiashu Xu, Dinghao Yang, Xiaodong Yang, Haotian Ye, Seonghyeon Ye, Xiaohui Zeng, Jing Zhang, Qinsheng Zhang, Kaiwen Zheng, Andrew Zhu, and Yuke Zhu. World simulation with video foundation models for physical AI, 2025a. URL https://arxiv.org/abs/2511.00062. 5

Wan Team, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models, 2025. URL https://arxiv.org/abs/2503.20314. 5, 6

NVIDIA, Alisson Azzolini, Junjie Bai, Hannah Brandon, Jiaxin Cao, Prithvijit Chattopadhyay, Huayu Chen, Jinju Chu, Yin Cui, Jenna Diamond, Yifan Ding, Liang Feng, Francesco Ferroni, Rama Govindaraju, Jinwei Gu, Siddharth Gururani, Imad El Hanafi, Zekun Hao, Jacob Huffman, Jingyi Jin, Brendan Johnson, Rizwan Khan, George Kurian, Elena Lantz, Nayeon Lee, Zhaoshuo Li, Xuan Li, Maosheng Liao, Tsung-Yi Lin, Yen-Chen Lin, Ming-Yu Liu, Xiangyu Lu, Alice Luo, Andrew Mathau, Yun Ni, Lindsey Pavao, Wei Ping, David W. Romero, Misha Smelyanskiy, Shuran Song, Lyne Tchapmi, Andrew Z. Wang, Boxin Wang, Haoxiang Wang, Fangyin Wei, Jiashu Xu, Yao Xu, Dinghao Yang, Xiaodong Yang, Zhuolin Yang, Jingxu Zhang, Xiaohui Zeng, and Zhe Zhang. Cosmos-Reason1: From physical common sense to embodied reasoning, 2025b. URL https://arxiv.org/abs/2503.15558. 5

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t. 5

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054. 7, 8, 50, 53, 69, 124, 125

NVIDIA. NVIDIA Isaac GR00T 1.7: Open reasoning VLA model for humanoid robots. https://github.com/NVIDIA/Isaac-GR00T, 2026a. 7, 8, 69, 124, 125, 126

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination?, 2026. URL https://arxiv.org/abs/2603.16666. 7

Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, Weikai Huang, Xiang Fan, Wei-Chuan Tsai, Shirui Chen, Yi Ru Wang, Shanli Xing, Jaemin Cho, Jae Sung Park, Ainaz Eftekhar, Peter Sushko, Karen Farley, Angad Wadhwa, Cole Harrison, Winson Han, Ying-Chun Lee, Eli VanderBilt, Rose Hendrix, Suveen Ellawela, Lucas Ngoo, Joyce Chai, Zhongzheng Ren, Ali Farhadi, Dieter Fox, and Ranjay Krishna. Molmoact2: Action reasoning models for real-world deployment, 2026. URL https://arxiv.org/abs/2605.02881. 7, 125

Black Forest Labs. Flux 3 action, 2026. URL https://bfl.ai/models/flux-3-action. 7, 125

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control, 2026. URL https://arxiv.org/abs/2601.21998. LingBot-VA. 8

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300. 9, 64, 84, 89

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, 2026. 10, 22, 88, 99, 107, 124

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026a. 10, 22, 99, 115, 124

Runyu Lu, Yubo Wu, Ethan Kou, Haotian Lin, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Guanzhi Wang. ASPIRE: Agentic /Skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272. 11, 98

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 11, 98, 100

Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Fernando Castañeda, Sirui Chen, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, Jinhyung Park, David Sami, Zi Wang, Xingye Da, Runyu Ding, Cyrus Hogg, Lina Song, Edy Lim, Eugene Jeong, Tairan He, Haoru Xue, Wenli Xiao, Simon Yuen, Jan Kautz, Yan Chang, Umar Iqbal, Linxi "Jim" Fan, and Yuke Zhu. Sonic: Supersizing motion tracking for natural humanoid whole-body control. *Science Robotics*, 11(117):eaed4592, 2026. doi: 10.1126/scirobotics.aed4592. URL https://www.science.org/doi/abs/10.1126/scirobotics.aed4592. 13, 109

Feng Wang and Zihao Yu. Coefficients-preserving sampling for reinforcement learning with flow matching. *arXiv preprint arXiv:2509.05952*, 2025. 14, 84

XDOF. ABC-130k. https://huggingface.co/datasets/XDOF/ABC-130k, 2026. 16, 62

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024. 16, 62, 107, 120, 125

Sicong Leng, Yunze Man, Zhiqi Li, Jing Wang, Yuhe Zhang, Ryan Punamiya, Siddharth Patki, Kaiyuan Zheng, Zi-ang Cao, Zhiding Yu, Sifei Liu, Danfei Xu, Ben Levin, Jan Kautz, Yuke Zhu, Johan Bjorck, and Linxi Fan. FLORA: An open model suite for scaling robotics data engines. *arXiv preprint arXiv:2610.31779*, 2026. URL https://arxiv.org/abs/2610.31779. 16, 63

Macrodata Labs. Segmenting robot video into actionable subtasks. https://macrodata.co/blog/annotating-robot-video-subtasks, 2026. June 25, 2026; accessed September 15, 2026. 16, 64

OpenAI. GPT-6 Astra system card. https://deploymentsafety.openai.com/gpt-6-astra, September 2026. OpenAI Deployment Safety Hub. 16, 64

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. 17, 68, 121

Zhenyang Chen, Chuizheng Kong, Chuye Zhang, Yuanshao Yang, Lawrence Y Zhu, Shreyas Kousik, and Danfei Xu. Warp: Whole-body retargeting for learning from offline human demonstrations. *arXiv preprint arXiv:2606.29940*, 2026b. 18, 44

Chuizheng Kong, Yunho Cho, Wonsuhk Jung, Idris Wibowo, Parth Shinde, Sundhar Vinodh-Sangeetha, Long Kiu Chung, Zhenyang Chen, Andrew Mattei, Advaith Nidumukkala, et al. A closed-form geometric retargeting solver for upper body humanoid robot teleoperation. *arXiv preprint arXiv:2602.01632*, 2026. 18, 44

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 *Proceedings of Robotics: Science and Systems (RSS)*, 2024. 19, 48, 73

NVIDIA GEAR Team. GR00T data hardware prototypes. https://research.nvidia.com/labs/gear/hardware-prototypes/, 2026. NVIDIA GEAR research page, accessed November 7, 2026. 19, 50, 73, 74, 81

Mengda Xu, Han Zhang, Yifan Hou, Zhenjia Xu, Linxi Fan, Manuela Veloso, and Shuran Song. DexUMI: Using human hand as the universal manipulation interface for dexterous manipulation. In *Conference on Robot Learning (CoRL)*, pages 437–459. PMLR, 2025. 19, 50, 73, 82

Yifan Hou, Zeyi Liu, Cheng Chi, Eric Cousineau, Naveen Kuppuswamy, Siyuan Feng, Benjamin Burchfiel, and Shuran Song. Adaptive compliance policy: Learning approximate compliance for diffusion guided control. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4829–4836. IEEE, 2025. 20, 52

NVIDIA. Video to Data (V2D). GitHub repository, 2026b. URL https://github.com/nvidia-isaac/video_to_data. 20, 54

Xinghao Zhu, Zixi Liu, Shalin Jain, Chenran Li, Milad Noori, Michael Andres Lin, Huihua Zhao, John Welsh, Mrinal Verghese, Wei Liu, Tingwu Wang, Xingye Da, Zhengyi Luo, Vishal Kulkarni, Naema Bhatti, Yuke Zhu, Linxi Fan, Bowen Wen, Danfei Xu, Soha Pouya, and Yan Chang. Learning dexterous manipulation using contact wrench guidance from human demonstration. In *Proceedings of the 10th Annual Conference on Robot Learning (CoRL)*, Austin, USA, November 2026. URL https://arxiv.org/abs/2607.00033. 20, 54, 55

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, et al. SimFoundry: Modular and automated scene generation for policy learning and evaluation. *arXiv preprint arXiv:2606.28276*, 2026. 21, 119, 120

NVIDIA Isaac Lab-Arena Contributors. Isaac Lab-Arena: Composable environment creation and policy evaluation for robotics, 2025. URL https://github.com/isaac-sim/IsaacLab-Arena. 21, 119

The Newton Contributors. Newton: Gpu-accelerated physics simulation for robotics and simulation research. Software, 2025. URL https://github.com/newton-physics/newton. 21, 119

Google DeepMind and NVIDIA. MuJoCo Warp. Software, 2025. URL https://github.com/google-deepmind/mujoco_warp. Accessed 3 November 2026. 21, 119

Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, et al. DreamDojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026. URL https://arxiv.org/abs/2602.06949. 22, 84, 92

Jenai Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A High-Fidelity Simulation Benchmark for Analysis of Task Generalist Policies. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026. URL https://arxiv.org/abs/2604.09860. 22, 125

Yejin Kim, Wilbert Pumacay, Omar Rayyan, Max Argus, Winson Han, Eli VanderBilt, Jordi Salvador, Abhay Deshpande, Rose Hendrix, Snehal Jauhri, et al. Molmospaces: A large-scale open ecosystem for robot navigation and manipulation. *arXiv preprint arXiv:2602.11337*, 2026. 22, 125

NVIDIA. G1 Factory: Whole-body humanoid locomanipulation benchmark for industrial tasks, 2026c. URL https://github.com/NVlabs/G1-Factory. 22, 126

Kevin Black, Allen Z. Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking, 2025. URL https://arxiv.org/abs/2512.05964. 23, 113, 115

Connor Pedersen, Dong H. Ahn, Michel Migdal, Collin Neale, and Nik Konyuchenko. Instant GPU efficiency visibility at fleet scale. arXiv preprint arXiv:2605.20799, May 2026. URL https://arxiv.org/abs/2605.20799. 25, 132

Weston Pace, Chang She, Lei Xu, Will Jones, Albert Lockett, Jun Wang, and Raunak Shah. Lance: Efficient random access in columnar storage through adaptive structural encodings. *arXiv preprint arXiv:2504.15247*, 2025. doi: 10.48550/arXiv.2504.15247. URL https://arxiv.org/abs/2504.15247. 25, 89, 130

Lance Contributors. Lance table format. https://lance.org/format/table/, 2026. Accessed October 30, 2026. 25, 89, 130

Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems*, 2019. 26

Chuan Wen, Jierui Lin, Trevor Darrell, Dinesh Jayaraman, and Yang Gao. Fighting copycat agents in behavioral cloning from observation histories. In *Advances in Neural Information Processing Systems*, 2020. 26

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. Sam 3: Segment anything with concepts, 2025. URL https://arxiv.org/abs/2511.16719. 54, 101

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D pose estimation and tracking of novel objects. In *CVPR*, 2024. 54

Qirui Wu, Stan Birchfield, Hesam Rabeti, Angel X. Chang, and Bowen Wen. GATOR: Generative and agentic 3D object reconstruction from casual images, 2026. URL https://arxiv.org/abs/2610.11215. 54

Patrick Rim, Kevin Harris, Braden Copple, Shangchen Han, Xu Xie, Ivan Shugurov, Sizhe An, He Wen, Alex Wong, Tomas Hodan, and Kun He. SHOW3D: Capturing scenes of 3D hands and objects in the wild. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://arxiv.org/abs/2603.28760. 54

Kushal Kedia, Tyler Ga Wei Lum, Jeannette Bohg, and C. Karen Liu. Simtoolreal: An object-centric policy for zero-shot dexterous tool manipulation, 2026. URL https://arxiv.org/abs/2602.16863. 54

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. Sapg: Split and aggregate policy gradients. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, Proceedings of Machine Learning Research, Vienna, Austria, July 2024. PMLR. 55

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. 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. 55, 111

Simar Kareer, Karl Pertsch, James Darpinian, Judy Hoffman, Danfei Xu, Sergey Levine, Chelsea Finn, and Suraj Nair. Emergence of human to robot transfer in vision-language-action models, 2025. URL https://arxiv.org/abs/2512.22414. 57

Ryan Punamiya, Simar Kareer, Zeyi Liu, Josh Citron, Ri-Zhao Qiu, Xiongyi Cai, Alexey Gavryushin, Jiaqi Chen, Davide Liconti, Lawrence Y. Zhu, Patcharapong Aphiwetsa, Baoyu Li, Aniketh Cheluva, Pranav Kuppili, Yangcen Liu, Dhruv Patel, Aidan Gao, Hye-Young Chung, Ryan Co, Renee Zbizika, Jeff Liu, Xiaomeng Xu, Haoyu Xiong, Geng Chen, Sebastiano Oliani, Wenkai Xuan, Chenyu Yang, Xi Wang, James Fort, Richard Newcombe, Josh Gao, Jason Chong, Garrett Matsuda, Aseem Doriwala, Marc Pollefeys, Robert Katzschmann, Xiaolong Wang, Shuran Song, Judy Hoffman, and Danfei Xu. EgoVerse: An egocentric human dataset for robot learning from around the world, 2026. URL https://arxiv.org/abs/2604.07607. 57

Ruijie Zheng, Dantong Niu, Yuqi Xie, Jing Wang, Mengda Xu, Yunfan Jiang, Fernando Castañeda, Fengyuan Hu, You Liang Tan, Letian Fu, Trevor Darrell, Furong Huang, Yuke Zhu, Danfei Xu, and Linxi Fan. Egoscale: Scaling dexterous manipulation with diverse egocentric human data, 2026. URL https://arxiv.org/abs/2602.16710. 57

Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. 57

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In *International Conference on Machine Learning (ICML)*, 2024. 65

David R. Hunter. MM algorithms for generalized Bradley–Terry models. *The Annals of Statistics*, 32(1):384–406, 2004. 68

Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Publishing, New York, 1978. 68

Anthony C. Davison and David V. Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, 1997. 68

Nathaniel Schenker and Jane F. Gentleman. On judging the significance of differences by examining the overlap between confidence intervals. *The American Statistician*, 55(3):182–186, 2001. 68

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. *Advances in neural information processing systems*, 38:45650–45685, 2025. 84, 86, 95

Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. Performant robotic manipulation with real-world reinforcement learning. *Science Robotics*, 11 (116):eaed6267, 2026. doi: 10.1126/scirobotics.aed6267. URL https://www.science.org/doi/abs/10.1126/scirobotics.aed6267. 84, 85, 86

Allen Ren, Justin Lidard, Lars Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *International Conference on Learning Representations*, 2025. 84

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024. 85

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 86

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 2025. arXiv:2410.21845. 87

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018. 87

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, Zixiao Huang, Mingjie Wei, Yuqing Xie, Ke Yang, Bo Dai, Zhexuan Xu, Jiakun Du, Xiangyuan Wang, Xu Fu, Letong Shi, Zhihao Liu, Kang Chen, Weilin Liu, Gang Liu, Boxun Li, Jianlei Yang, Zhi Yang, Guohao Dai, and Yu Wang. RLinf: Flexible and Efficient Large-Scale Reinforcement Learning via Macro-to-Micro Flow Transformation. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 829–846, 2026a. URL https://www.usenix.org/conference/osdi26/presentation/yu-chao. 90

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Peihong Wang, Hua Yuan, Yixian Zhang, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, Zhihao Liu, Kang Chen, Wenhao Tang, Quanlu Zhang, Weinan Zhang, Chao Yu, and Yu Wang. RLinf-VLA: A Unified and Efficient Framework for Reinforcement Learning of Vision-Language-Action Models. In *Proceedings of Robotics: Science and Systems XXII (RSS)*, 2026a. doi: 10.15607/RSS.2026.XXII.089. URL https://www.roboticsproceedings.org/rss22/p089.html. 90

Hongzhi Zang, Shu'ang Yu, Hao Lin, Tianxing Zhou, Zefang Huang, Zhen Guo, Xin Xu, Yuze Sheng, Jiakai Zhou, Shizhe Zhang, Feng Gao, Wenhao Tang, Yufeng Yue, Quanlu Zhang, Xinlei Chen, Chao Yu, and Yu Wang. USER: A Unified and Extensible System for Online Real-World Policy Learning in Embodied AI. In *Proceedings of Robotics: Science and Systems XXII (RSS)*, 2026b. doi: 10.15607/RSS.2026.XXII.037. URL https://www.roboticsproceedings.org/rss22/p037.html. 90

Yi Wang, Xinchen Li, Pengwei Xie, Pu Yang, Buqing Nie, Yunuo Cai, Qinglin Zhang, Chendi Qu, Jeffrey Wu, Jianheng Song, Xinlin Ren, Jingshun Huang, Mingjie Pan, Siyuan Feng, Zhi Chen, and Jianlan Luo. Learning While Deploying: Fleet-Scale Reinforcement Learning for Generalist Robot Policies, 2026. URL https://arxiv.org/abs/2605.00416v4. Version 4. 90

Zhenqian Chen, Baoquan Zhong, Xiang Li, Qing Dai, Xinkui Zhao, Miao Ye, Ren Cheng, Lufei Zhang, and Jianwei Yin. RobustRL: Role-Based Fault Tolerance System for RL Post-Training. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1407–1424, 2026c. URL https://www.usenix.org/conference/osdi26/presentation/chen-zhenqian. 90

Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/416f9cb3276121c42eebb86352a4354a-Abstract-Conference.html. 94

Awomo-WAM Team. Awomo-0.5 technical report, 2026. URL https://github.com/Awomo-WestlakeDI/Awomo-0.5. 99, 124

Renbin Shu, Jing Yang, Xiangyuan You, Guolin Qiu, Yufei Xia, Yiming Zhu, Nanjun Dai, Fei Zhang, Feng Wu, and Keke Tang. Axiom-0: Hierarchical self-refinement for vision-language-action policies, 2026. URL https://arxiv.org/abs/2610.14155. 99, 124

Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE Robotics and Automation Letters*, 5(2), 2020. doi: 10.1109/LRA.2020.2965869. 105

Zhenyang Chen, Alan Tian, Liquan Wang, Benjamin Joffe, Yingyan Celine Lin, Yuxiao Chen, Siddharth Karamcheti, and Danfei Xu. ReSteer: Quantifying and refining the steerability of multitask robot policies. *arXiv preprint arXiv:2603.17300*, 2026d. 106

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025. 107

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 36, pages 44776–44791, 2023. 107

Shashank Maheshwari, Suhas Hariharapura Sheshadri, Leela Subramaniam Karumbunathan, and Aayush Pathak. Introducing NVIDIA Jetson Thor, the ultimate platform for physical AI. https://developer.nvidia.com/blog/introducing-nvidia-jetson-thor-the-ultimate-platform-for-physical-ai/, August 2025. NVIDIA Technical Blog, August 25, 2025. 109

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003. 111

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. 111

Eftychios Sifakis and Jernej Barbic. FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, pages 1–50. ACM, 2012. 119

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 119

Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, et al. PolaRiS: Scalable real-to-sim evaluations for generalist robot policies. *arXiv preprint arXiv:2512.16881*, 2025. 120

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024. 120

Agustin Lage-Castellanos, Giancarlo Valente, Elia Formisano, and Federico De Martino. Methods for computing the maximum performance of computational models of fmri responses. *PLoS computational biology*, 15(3): e1006397, 2019. 121

NVIDIA. Isaac Sim. https://github.com/isaac-sim/IsaacSim, 2025. Open-source release, Apache-2.0 license. 124

Yanjiang Guo, Haodong Yan, Zhide Zhong, Zhongru Zhang, Qingyuan Yang, Qingzhou Lu, Xiaoyu Chen, Yen-Jen Wang, Shuying Deng, Chenghan Yang, Puzhen Yuan, Chenxin Liu, Tun Ban, Xiang Zhu, Yichen Liu, Kun Feng, Haoang Li, and Jianyu Chen. Video prediction policy 2: Predict better, act better, 2026. URL https://arxiv.org/abs/2610.10270. 124

Shijun Shi, Zhe Liu, Zhi Hou, Yinan Chen, Tao Zhang, Kai Hu, Yong Liu, and Jiangning Zhang. Atomic-wam: Execution-grounded atomic-motion reasoning for world action models, 2026. URL https://arxiv.org/abs/2610.20157. 125

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-XL: A universe of 10M+ 3D objects. *arXiv preprint arXiv:2307.05663*, 2023. 125

Yang Zhang, Jiangyuan Zhao, Chenyou Fan, Fangzheng Yan, Tian Li, Haitong Tang, Sen Fu, Xuan'er Wu, Qizhen Weng, Weinan Zhang, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. Prts: A primitive reasoning and tasking system via contrastive representations. *arXiv preprint arXiv:2604.27472*, 2026. 125

Ryan Yu, Pushi Zhang, Starrick Liu, Brae Liu, Miracle Kang, Shalfun Li, Lights Shi, Ellie Ma, Ping Yang, Chris Pan, Jerry Chen, Dongxiu Liu, Rain Sun, Miles Guo, Byron Zhang, Hugo Zhou, Zach Xu, Vincent Chen, Harrison Huang, James Wang, Dance Kuzi, Andy Zhai, Hang Su, Roy Gan, Lucy Liang, Hao Wang, and Qian Wang. Wall-oss-0.5 technical report, 2026b. URL https://arxiv.org/abs/2605.30877. 125

NVIDIA Cosmos Team. Cosmos 3: Omnimodal world models for physical AI. Technical report, NVIDIA, 2026. URL https://research.nvidia.com/labs/cosmos-lab/cosmos3/technical-report.pdf. 125

Kevin Lin, Ajay Mandlekar, Caelan Reed Garrett, Nikita Chernyadev, Yu Fang, Runyu Ding, Yuqi Xie, Justin Tran, Linxi Fan, and Yuke Zhu. Humanoidmimicgen: Data generation for loco-manipulation via whole-body planning. *arXiv preprint arXiv:2605.27724*, 2026. 126

Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. SkyPilot: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 437–455, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng. 127, 129

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages

561–577. USENIX Association, 2018. URL https://www.usenix.org/conference/osdi18/presentation/moritz. 127, 134

Kyle Aubrey and Nick Stam. Inside NVIDIA Blackwell Ultra: The chip powering the AI factory era. NVIDIA Technical Blog, August 2025. URL https://developer.nvidia.com/blog/inside-nvidia-blackwell-ultra-the-chip-powering-the-ai-factory-era/. Published August 22, 2025; accessed November 2, 2026. 128

NVIDIA. NVIDIA GB300 NVL72. Product specifications, n.d.a. URL https://www.nvidia.com/en-us/data-center/gb300-nvl72/. Accessed November 2, 2026. 128

NVIDIA. NCCL user guide: Environment variables. NVIDIA documentation, n.d.b. URL https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html. See NCCL_NVLS_ENABLE; accessed November 2, 2026. 128

NVIDIA. NVIDIA L40 GPU Datasheet. NVIDIA datasheet, January 2023. URL https://images.nvidia.com/content/Solutions/data-center/vgpu-L40-datasheet.pdf. Accessed November 3, 2026. 128

NVIDIA. NVIDIA L40S GPU Specifications. NVIDIA product specifications, n.d.c. URL https://www.nvidia.com/en-us/data-center/l40s/. Accessed November 3, 2026. 128

NVIDIA. AI Cluster Runtime (AICR). NVIDIA AI Cluster Runtime documentation, n.d.d. URL https://docs.nvidia.com/aicr/latest/overview/introduction/. Accessed November 2, 2026. 128

NVIDIA. DCGM Exporter. NVIDIA DCGM documentation, n.d.e. URL https://docs.nvidia.com/datacenter/dcgm/latest/reference/command-line-reference/dcgm-exporter.html. Accessed November 2, 2026. 132

Alex Poms, William Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Transactions on Graphics*, 37(4), 2018. URL https://graphics.stanford.edu/papers/scanner/. 134

Daoyuan Chen, Yilun Huang, Xuchen Pan, Nana Jiang, Haibin Wang, Yilei Zhang, Ce Ge, Yushuo Chen, Wenhao Zhang, Zhijian Ma, Jun Huang, Wei Lin, Yaliang Li, Bolin Ding, and Jingren Zhou. Data-Juicer 2.0: Cloud-scale adaptive data processing for and with foundation models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-2768. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/76dbd7e897be2eb883ede598b91270fb-Paper-Datasets_and_Benchmarks_Track.pdf. 134

Frank Sifei Luan, Ron Yifeng Wang, Yile Gu, Ziming Mao, Charlotte Lin, Amog Kamsetty, Hao Chen, Cheng Su, Balaji Veeramani, Scott Lee, SangBin Cho, Clark Zinzow, Eric Liang, Ion Stoica, and Stephanie Wang. Ray Data: A dynamic and scalable system for heterogeneous data processing. arXiv preprint arXiv:2501.12407v6, 2026. URL https://arxiv.org/abs/2501.12407v6. 134

Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 21)*. ACM, 2021. doi: 10.1145/3458817.3476181. URL https://spcl.inf.ethz.ch/Publications/index.php?pub=426. 134

A. Fast-Adapt with GR00T 2

A.1. Benchmark Tasks

Figs. 21 and 22 show all tasks of the two post-training benchmarks in Sec. 3.1: the six YAM tasks of Tab. 1a and the seven Dexter tasks of Tab. 1b.

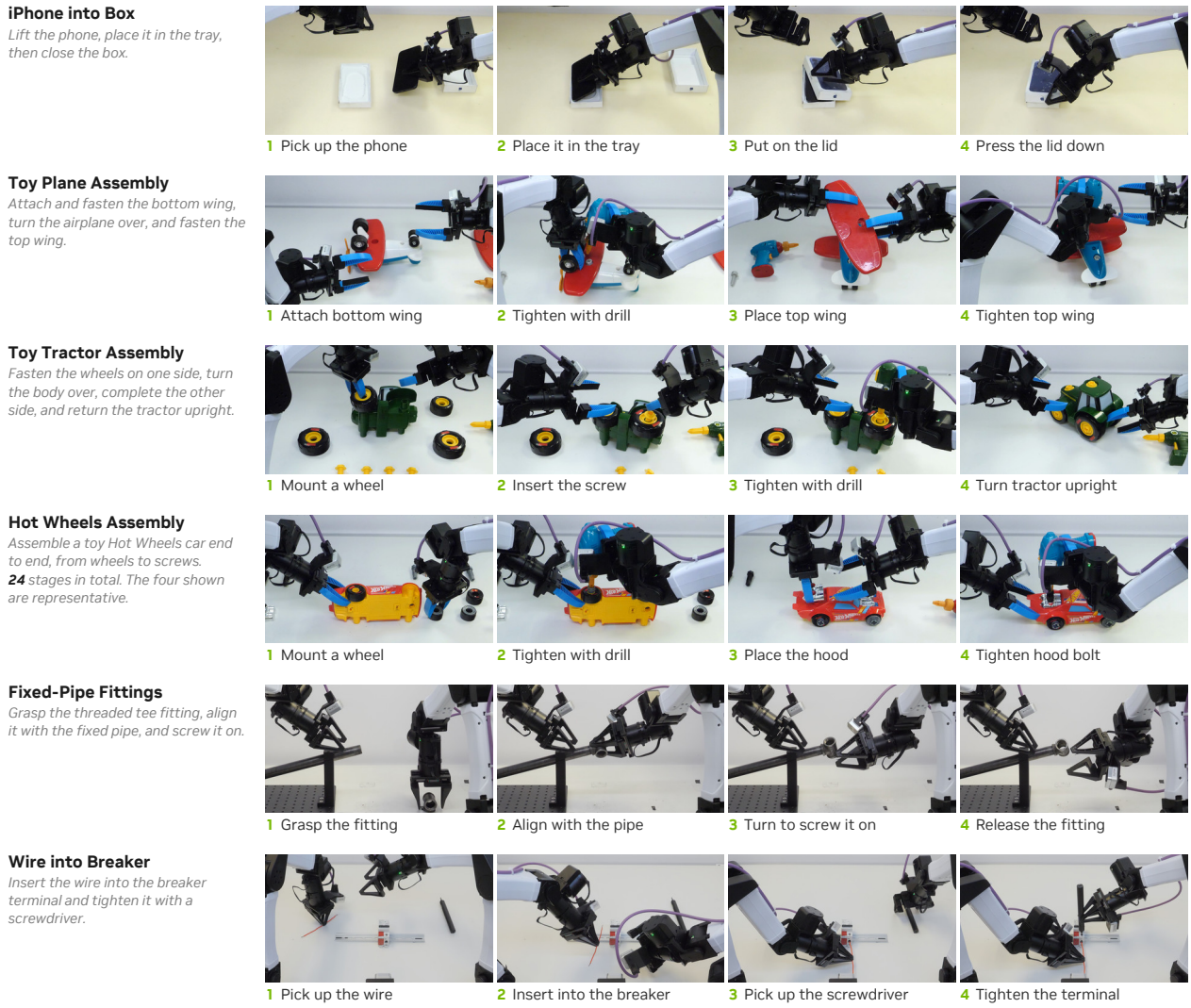


Figure 21: YAM post-training benchmark tasks. The six tasks of Tab. 1a, one per row. Left: task name and a short description. Right: key frames from four representative stages of each task, which has more stages than shown (Hot Wheels Assembly has 24).

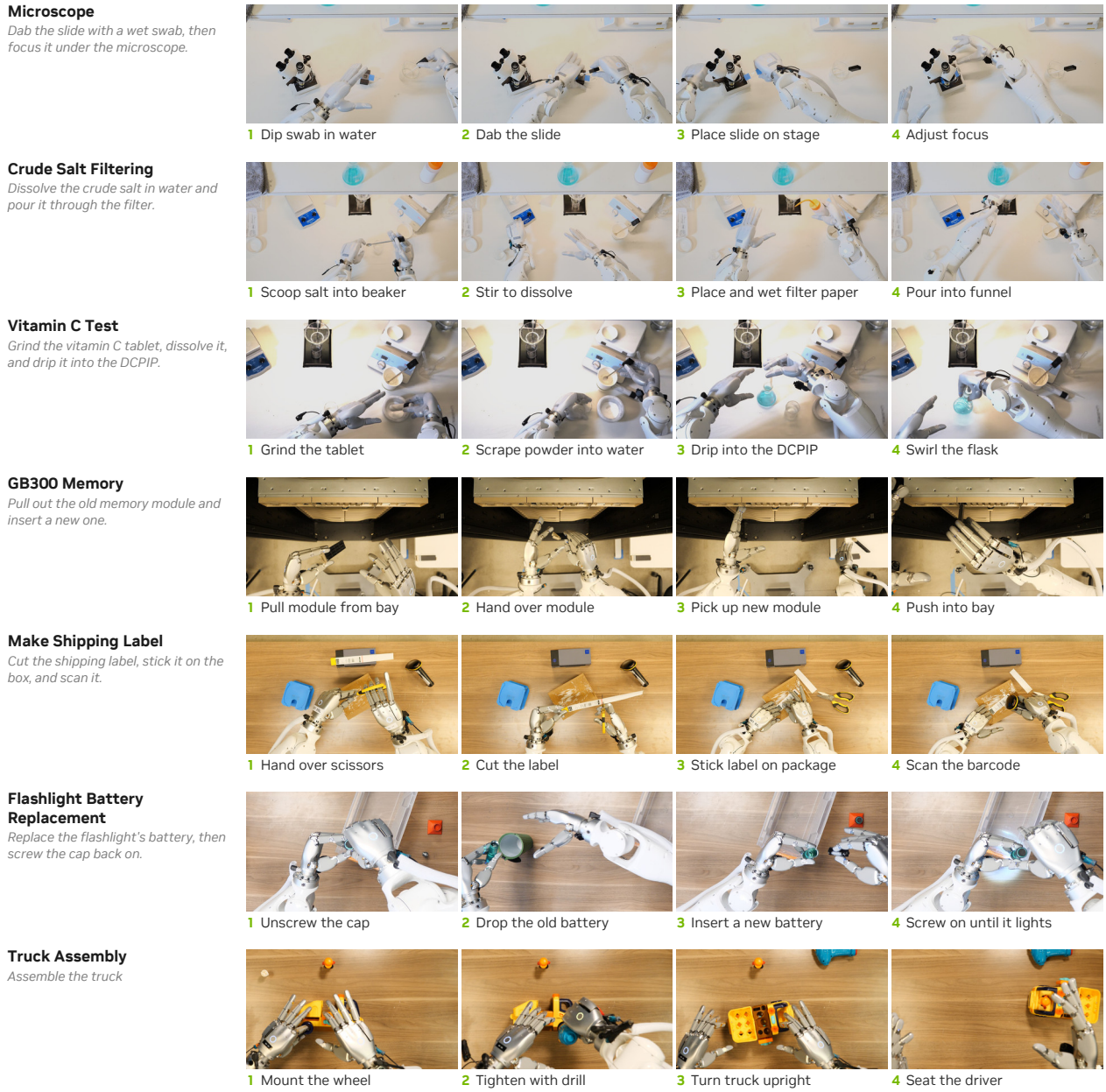


Figure 22: Dexter post-training benchmark tasks. The seven tasks of Tab. 1b, one per row. Left: task name and a short description. Right: key frames from four representative stages of each task.

Fig. 23 plots the average task progress of each model in Tab. 1a against its inference latency.

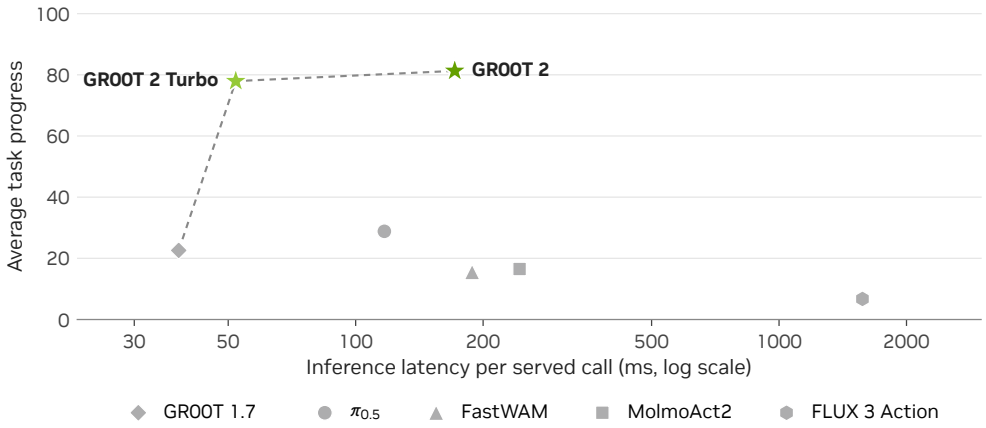


Figure 23: YAM post-training Pareto frontier. Average task progress across the six tasks in Tab. 1a versus inference latency per served call on a GeForce RTX 5090 workstation. The dashed line indicates the Pareto frontier.

A.2. Zero-Shot Generalization with GR00T 2

YAM Zero-Shot Benchmark. We evaluate the pretrained policies without task-specific post-training on the YAM V2 benchmark. For $\pi_{0.5}$, we use a checkpoint adapted on in-domain data. We report results for the three unseen evaluation pools: Language Following, Motion Primitives (Easy), and Motion Primitives (Medium/Hard), comprising 14, 10, and 14 tasks, respectively. The benchmark evaluates language grounding and a range of manipulation motions, including actions beyond pick-and-place (Tab. 2).

Table 2: YAM V2 zero-shot evaluation. For each model, Prog. is the average task score and SR is the success rate, both on a 0 to 100 scale. We pool graded trials across available evaluations and weight tasks equally within each group. Trial counts vary across tasks and models, and results include graded trials from ongoing evaluations. †GR00T 1.7 results are preliminary. –: no result available. Bold marks the best Prog. and SR in each row.

	GR00T 2		**GR00T 2 Turbo**		GR00T 1.7†		$\pi_{0.5}$		MolmoAct2	
Task group	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR
Language Following	45.2	**31.4**	41.2	31.0	**47.7**	30.4	21.5	17.9	–	–
Motion Primitives: Easy	**54.1**	**36.6**	53.6	33.0	47.5	27.5	16.9	14.2	–	–
Motion Primitives: Medium/Hard	**40.6**	**29.9**	33.2	19.7	13.6	5.4	1.8	0.0	–	–

Dexter Zero-Shot Benchmark. We further evaluate zero-shot transfer on Dexter, a bimanual robot with two dexterous hands, again without task-specific post-training. The evaluation covers 71 tasks: 49 dexterous manipulation tasks and 22 out-of-distribution instruction tasks that probe language grounding through color, size, ordinal position, negation, spatial relations, counting, and named targets. Each model is run for three trials per task, and we report results pooled over all tasks (Tab. 3).

Table 3: Dexter zero-shot evaluation. For each model, Prog. is the average task progress and SR is the success rate, both on a 0 to 100 scale, pooled over 71 tasks (49 manipulation tasks and 22 out-of-distribution instruction tasks) with three trials per task. The four models run interleaved in the same sessions, and the operator who labels success is not told which model is running. Task progress is rated per episode by a vision-language model from the head and wrist camera videos. Bold marks the best Prog. and SR in each row.

	GR00T 2		**GR00T 2 Turbo**		GR00T 1.7		$\pi_{0.5}$	
Task group	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR
All 71 tasks	**72.9**	**49.8**	61.4	36.6	43.3	18.3	56.2	26.3

A.3. Dexter-50: Emergent Cross-Embodiment Zero-Shot and Few-Shot Transfer.

GR00T 2 transfers tasks across embodiments. We evaluate it on Dexter, a bimanual robot with two 7-DoF arms and two 22-DoF Sharpa hands, on Dexter-50: 50 tasks that never appear in Dexter's training data but do appear in data from other embodiments, namely YAM parallel-gripper teleoperation, human video, or both. In the zero-shot setting, the pretrained model receives only the task instruction. In the few-shot setting, we adapt the same checkpoint to each task with 1 or 5 teleoperated Dexter episodes, a few minutes of robot data. We train rank-32 LoRA adapters on every linear layer of the transformer and keep the text encoder, video VAE, action and state heads, and all pretrained weights frozen. The adapters receive the same joint video and action flow-matching loss used in pretraining. We cut each episode into 100 evenly spaced training windows and sample them in proportion to the frozen model's loss on each window, so that updates concentrate on the hard phases of the task, such as grasping or insertion. All tasks share the same hyperparameters, with no per-task tuning. One run fits on a single RTX 5090 and takes about 30 minutes. We merge the adapters into the base weights after training, so the adapted model is served exactly like the pretrained one.

Across the 50 tasks, GR00T 2 succeeds 40% of the time zero-shot, 71% with one episode, and 95% with five (Fig. 24). Zero-shot, it completes 40 of the 50 tasks at least once and succeeds in every trial of four, such as unloading a dish rack and unstacking Rubik's cubes, skills it has seen only on a parallel-gripper robot or in human video. With five episodes, 37 of the 50 tasks reach 100%, including tasks the zero-shot model never completes, such as emptying a bag, turning a lazy susan, and unfolding a tripod. In most of the failures shown in Fig. 25, the zero-shot model goes for the right steps, but its new hands fail at one of them, such as lifting a teapot lid or opening a drawstring bag. One or five Dexter episodes fix this.

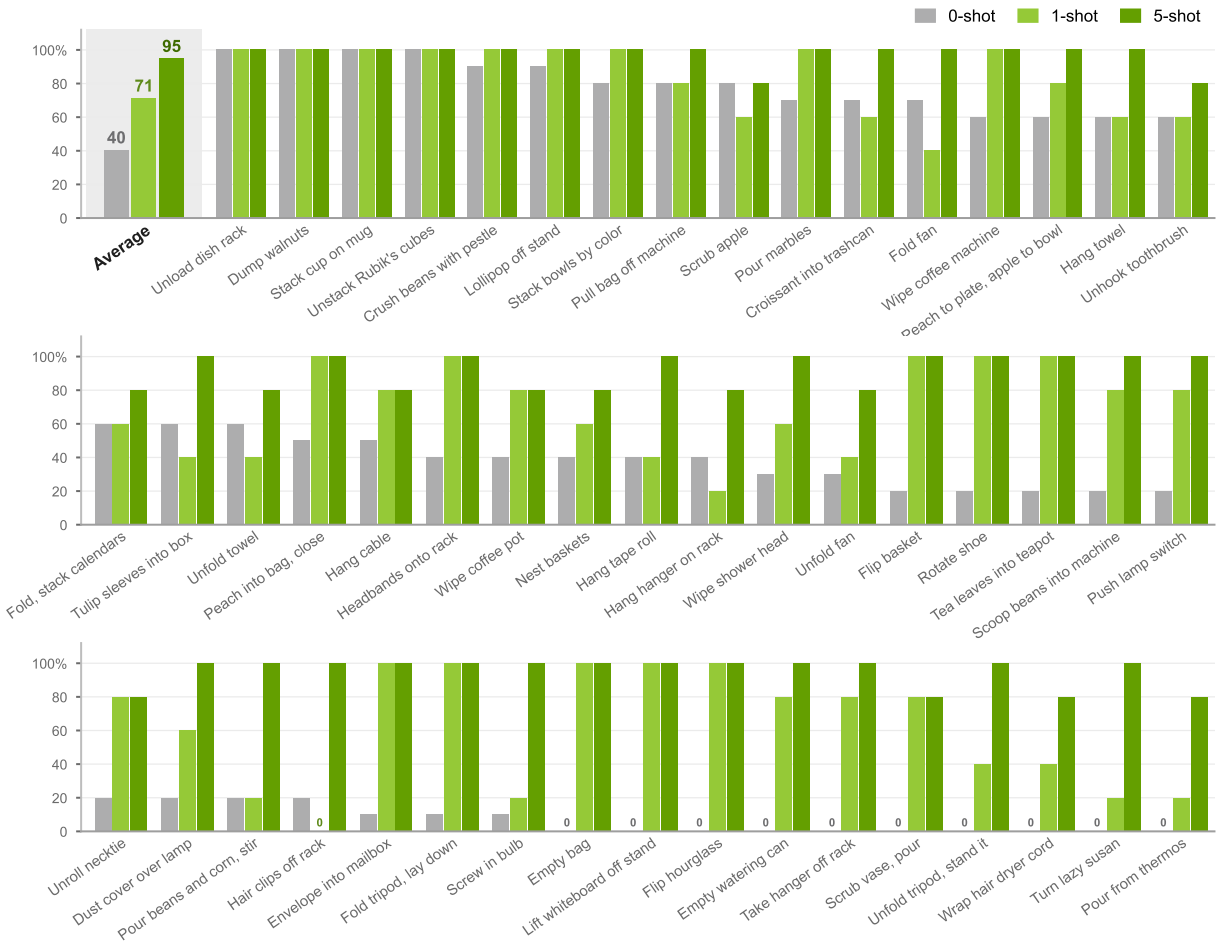


Figure 24: Cross-embodiment transfer to Dexter with 0, 1, and 5 Dexter episodes, per task. Success rate on the Dexter-50 tasks, which have no Dexter training data but appear in YAM teleoperation data, human video, or both, sorted by zero-shot success. Zero-shot runs the pretrained model directly, while 1-shot and 5-shot fine-tune it with LoRA on 1 or 5 Dexter episodes. The first group (gray band) is the average over all 50 tasks. Tasks are those the model learns with five episodes (at least 80% success).

B. X-Adapt with GR00T 2

This section details the cross-embodiment transfer summarized in Sec. 3.2 and the devices and pipelines described in Sec. 4.4.

B.1. Human → Tianji + Sharpa

GR00T 2 can also learn a dexterous task on a robot from human demonstrations alone. Here every task-specific demonstration comes from a person recorded with our human data capture pipeline, which pairs head and wrist cameras with wrist trackers and data gloves, and the resulting policy runs on the Tianji, a bimanual robot with two 7-DoF arms and two 22-DoF Sharpa hands. The embodiment gap now centers on the hand. Appearance differs because the cameras observe a human hand during demonstration and a robot hand at deployment. Kinematics differ because the larger Sharpa hand does not share the size or joint structure of the human hand, so retargeting must map every finger motion onto the 22-DoF robot hand. Dynamics differ because the human moves at natural speed, which teleoperation cannot reproduce on the robot. Three components bridge these gaps: the capture pipeline records robot-aligned views (Sec. B.1.1), a geometric retargeter converts every demonstration into robot actions (Sec. B.1.2), and co-training with task-agnostic play data, collected with our teleoperation system (Sec. C.3.1), grounds the model in the robot (Sec. B.1.3).

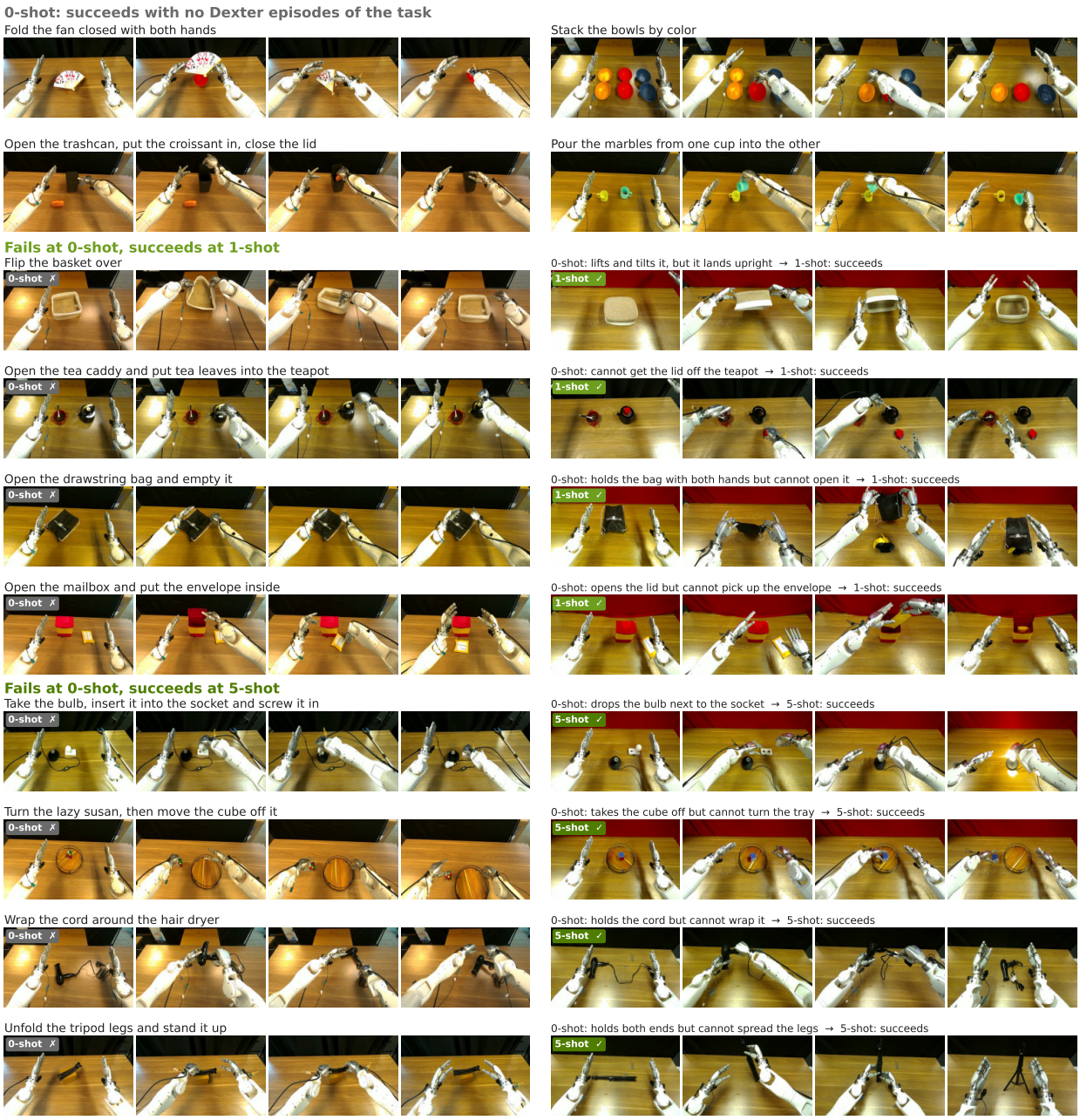


Figure 25: Example Dexter rollouts. Top: zero-shot successes. Middle and bottom: tasks the zero-shot model attempts but fails, mostly at a step its new hands cannot yet do (left, with the failure noted above each row), and completes after fine-tuning on one or five Dexter episodes (right). Rollouts ran on different Dexter stations, so the backgrounds differ.

We evaluate on four tasks (Fig. 26). Filter Paper Setup, Dropper-to-Beaker Dispensing, and Microscope Slide Dabbing are chemistry-lab tasks that handle small, often transparent labware and leave little tolerance for fingertip error. Conveyor Package Tossing is a dynamic task: the robot grasps each package from a moving belt and throws it over the belt into the bin assigned to that package, a targeted throw that teleoperation cannot demonstrate. For each task, we co-train the retargeted human demonstrations with task-agnostic Tianji play data. The play data contain no demonstrations of the target tasks. All models are post-trained with an identical data recipe and number of optimization steps. Without a single teleoperated demonstration of the target tasks, GR00T 2 succeeds in 65%, 30%, 10%, and 50% of 20 trials on the four tasks, whereas GR00T 1.7 and GR00T 2 without cross-embodiment pretraining fail every chemistry-lab trial and reach only 5% and 20% on tossing

(Fig. 27). Adding 10 teleoperated demonstrations raises GR00T 2 to 95%, 70%, and 50% on the chemistry-lab tasks, against 45%, 40%, and 30% without cross-embodiment pretraining and 0% for GR00T 1.7. One hundred human demonstrations per task therefore suffice to deploy precise and dynamic dexterous tasks on a robot that has never been teleoperated for them. Sec. B.1.4 details the protocol, and Sec. B.1.5 analyzes why the transfer works.

B.1.1. Human Data Capture and Conversion

Human data capture. Each demonstration is recorded with a lightweight egocentric rig: a head-mounted camera provides the ego view, wrist-mounted cameras provide close-up views of hand-object contact, and the hands are tracked by a motion-capture glove (Manus) paired with a 6-DoF tracker (PICO) rigidly attached to each wrist. The glove provides per-finger joint angles, while the tracker provides the global wrist pose. Together they yield a full 3D hand pose at every timestep without any robot in the loop. Because the tracker's IMU does not coincide with the anatomical wrist, we calibrate the tracker-to-wrist rigid transform per session: an annotator clicks fingertip locations in a handful of frames spanning diverse hand poses, and we solve for the transform that minimizes the reprojection error of the glove-predicted fingertips. Camera intrinsics and extrinsics are verified in the same tool by overlaying the tracked controller pose onto the video, so that calibration errors are caught before any data is converted. The raw stream (ego and wrist video, wrist poses, finger joints, and language annotations) is captured at 60 Hz, stored in a time-synchronized format and quality-checked by visualizing the reconstructed hand trajectory against the video.

From human data to robot data. The captured human trajectories are converted into robot-space supervision in a single offline pass, with no task-specific robot teleoperation. First, each wrist pose is anchored globally in the gravity-aligned robot base frame: the human's head camera is placed at the robot's head camera, and a per-task offset shifts the hands into the robot workspace. The converted trajectories therefore carry absolute wrist poses in the robot frame, not wrist displacements alone. Second, the human finger configuration is retargeted to the robot hand via inverse kinematics on fingertip targets, producing joint commands that respect the robot's kinematic limits while preserving the task-relevant contact geometry rather than the literal human pose. Third, the ego and wrist camera streams are kept as the policy's visual inputs, so that the resulting episodes, downsampled to 30 Hz, have the same observation and action schema as native robot data. During post-training, we sample converted human episodes and generic, non-task-specific robot play data with equal probability. This co-training grounds the model in the robot's appearance and dynamics, while human demonstrations provide all task-specific supervision.

B.1.2. Geometric Retargeting

The role of our retargeting solver is to bring sub-millimeter accuracy to robot finger tips with kHz computation speed. Given that the conversion is solely offline without human supervision, we ask for automated singularity handling near workspace boundary and robust redundancy DoF management. To achieve this we make use of a closed-form geometric inverse kinematic solving technique introduced in Chen et al. (2026b) and Kong et al. (2026) to handle both the 7-DoF arms and 22 DoF robot hands. Given 6D pose data of ego camera and left / right wrist mount, we first find the ideal placement of human data in robot frame for each task. The solver then retargets the hands before the arm, and both stages track fingertip targets. A unified "palm-box" hand frame introduced in Kong et al. (2026), anchored at the index fingertip, aligns human and robot hands. The 22-DoF hand solver places the robot thumb tip relative to the solved index fingertip, which preserves the pinch geometry, while the other fingers follow the human bone directions. The arm's geometric inverse-kinematics solver then tracks the palm-box origin, so the robot index fingertip lands on the human index fingertip.

Our combined arm-hand solver achieves approximately $118{,}000$ bimanual solves per second on a single core of an Intel Core Ultra 9 285K CPU, with a mean native computation time of $8.48\,\mu\mathrm{s}$, excluding Python interface overhead. Across $1{,}024$ synthetic in-range wrist targets, maximum forward-kinematics-verified pose errors remain below $10^{-15}\,\mathrm{m}$ and $10^{-15}\,\mathrm{rad}$. This throughput supports real-time teleoperation (Sec. C.3.1) and offline human-data conversion.

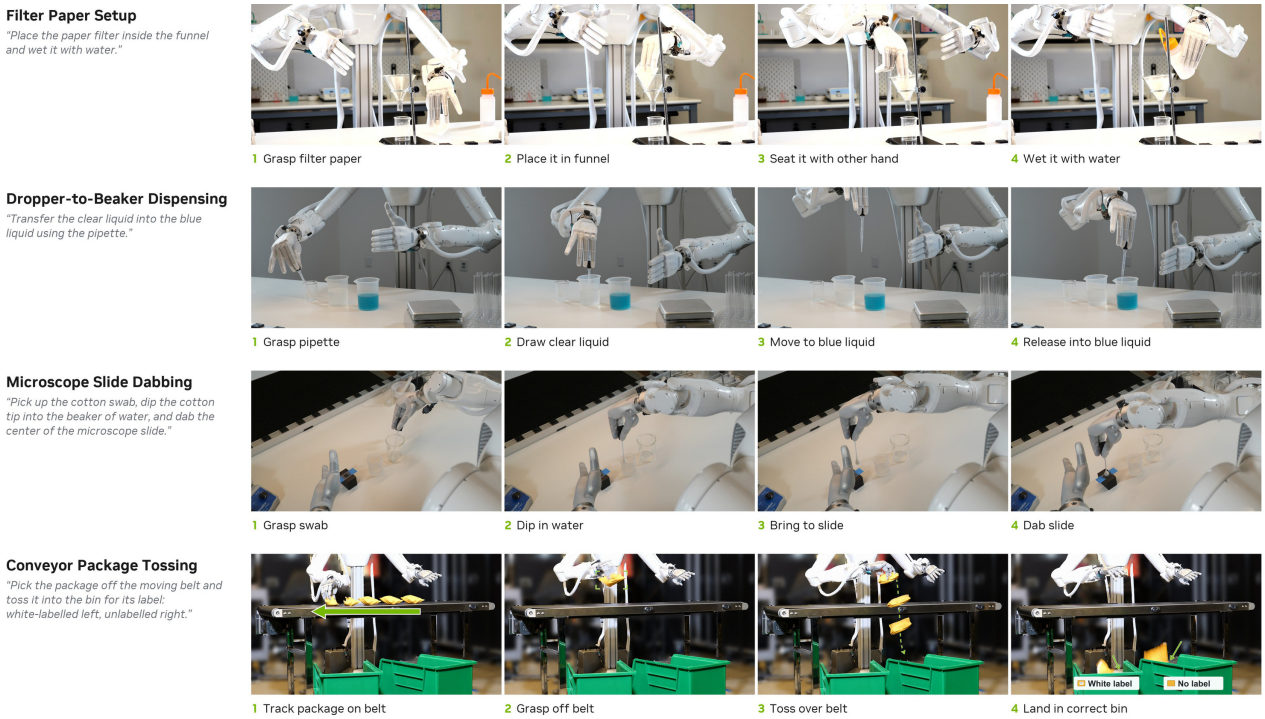


Figure 26: Sensorized human transfer benchmark tasks. Each row shows the ordered stages of one task, with the language instruction that GR00T 2 receives.

B.1.3. Co-Training with Robot Play Data

GR00T 2 turns retargeted human demonstrations into a robot policy by co-training them with robot play data. Starting from its cross-embodiment pretrained checkpoint, GR00T 2 fine-tunes on two complementary sources: retargeted human demonstrations of the target task, which contain the task but not the robot, and task-agnostic play data from the target robot, collected with the teleoperation system of Sec. C.3.1, which contain the robot but not the task. Human data teach GR00T 2 *what* to do: the task intent, the object interactions, and the motion strategy, such as which hand leads, which fingers make contact, and how fast the wrist moves. Play data teach GR00T 2 *how* this robot looks and moves: its appearance, kinematics, and dynamics, which stay the same across tasks, so one play pool serves every task. Without teleoperation, the play data are the only robot data in post-training. Each new task therefore adds only cheap human demonstrations.

Human and robot episodes share one data format, so the two sources mix freely within a training batch. Both provide three camera views (ego and two wrists) at 30 Hz, a 62-D state of absolute wrist poses in the robot base frame and hand-joint positions, and a 62-D action chunk of wrist poses relative to the current state and absolute hand-joint positions.

B.1.4. Learning New Tasks without Teleoperation

GR00T 2 learns new robot tasks from human demonstrations alone. Four tasks on the Tianji, a bimanual robot with two 7-DoF arms and two 22-DoF Sharpa hands, test this claim and stress two parts of the embodiment gap. Three *chemistry-lab tasks* handle small, often transparent labware and test precision under the kinematic gap, where retargeting error and the larger robot hand both consume the tolerance. The *dynamic task* tests the kinodynamic gap on a fast, targeted throw that teleoperation cannot demonstrate (Fig. 26).

Tasks and metrics. A rubric splits each task into K ordered stages, and we grade each trial by the number of stages k it completes in order. A trial succeeds when $k = K$, its task progress is k/K, and any human intervention counts as a failure.

- **Filter Paper Setup** (K=6): grasp the filter paper, place it in the funnel, unfold and seat it in the funnel,

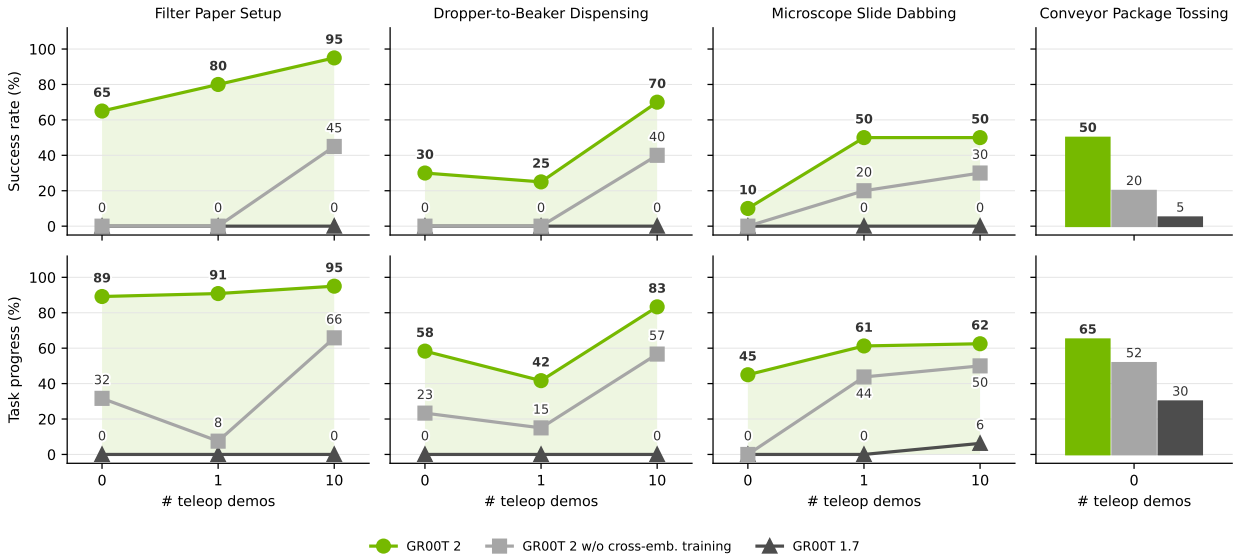


Figure 27: Learning new tasks with zero or few teleoperated demonstrations. Success rate (top) and task progress (bottom) over 20 trials per point, with 0, 1, or 10 teleoperated demonstrations of the target task added to post-training. At 0, no training data contain a robot trajectory of the task. All models share the human demonstrations, robot data, and post-training recipe. Each baseline changes one factor relative to GR00T 2, either the model formulation (GR00T 1.7, a VLA) or the pretraining data (GR00T 2 w/o cross-embodiment pretraining, pretrained on target-robot data alone, Sec. B.1.5). Conveyor Package Tossing is evaluated at 0 only, since teleoperation cannot demonstrate its fast, targeted throw.

grasp the water bottle, wet the filter paper, and put the bottle back.

- **Dropper-to-Beaker Dispensing** (K=3): grasp the pipette, draw the clear liquid, and release it into the blue liquid.
- **Microscope Slide Dabbing** (K=4): grasp the swab, dip it into the water, bring it to the slide, and dab the slide.
- **Conveyor Package Tossing** (K=3): grasp a package from the moving belt, toss it over the belt, and land it in the bin assigned to that package.

All models post-train on one mixture, retargeted human demonstrations and robot play data sampled 1:1, with one recipe and step budget and no per-task tuning or checkpoint selection. To measure what teleoperation still adds, we also add 1 or 10 teleoperated demonstrations of the target task to the mixture. Each point averages 20 trials.

Learning without teleoperation. Without a single teleoperated demonstration, GR00T 2 succeeds in 65%, 30%, 10%, and 50% of trials on Filter Paper Setup, Dropper-to-Beaker Dispensing, Microscope Slide Dabbing, and Conveyor Package Tossing (Fig. 27). Both baselines fail every chemistry-lab trial in the same setting and succeed only on tossing, with 20% without cross-embodiment pretraining and 5% for GR00T 1.7. The human demonstrations carry the task: without them, the same GR00T 2 recipe succeeds in 0% of trials on every task. Tossing shows that this transfer extends beyond teleoperation: GR00T 2 learns the throw entirely from humans.

Teleoperation adds to human data. Teleoperated demonstrations refine the task that human demonstrations teach rather than replace them. With 10 demonstrations, GR00T 2 succeeds in 95%, 70%, and 50% of chemistry-lab trials, compared with 45%, 40%, and 30% without cross-embodiment pretraining and 0% for GR00T 1.7. On Filter Paper Setup, GR00T 2 without teleoperation already outperforms the baseline trained with 10 teleoperated demonstrations.

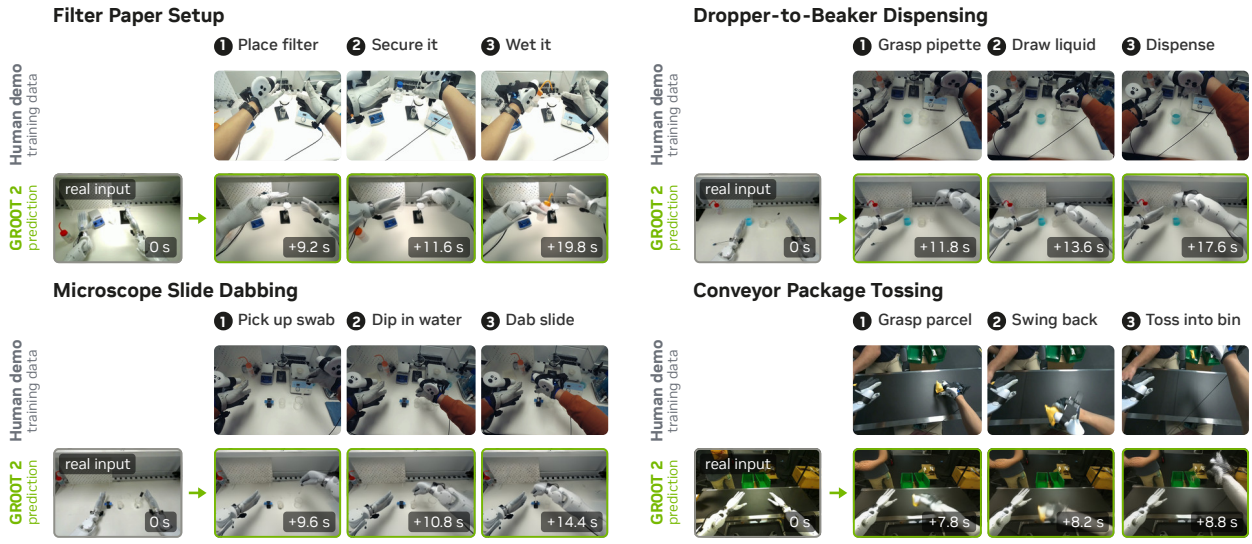


Figure 28: Predicted futures for tasks that only humans demonstrated. Given one robot start frame and the instruction, GR00T 2 predicts the robot's ego view autoregressively. Each GR00T 2 row comes from a single rollout that starts at the first frame of an evaluation episode, and times are measured from that frame. Human demonstrations (top rows) appear only in training.

B.1.5. Why GR00T 2 Learns without Teleoperation

Two baselines test the two properties that let GR00T 2 learn without teleoperation (Fig. 27). Each replaces one component of GR00T 2, the model formulation or the pretraining data, and keeps the post-training data, sampling weights, step budget, and evaluation protocol fixed.

World-action model. GR00T 2 learns from human-only task data because it is a world-action model. Unlike a VLA that must learn task and embodiment in one observation-to-action map, from human-hand observations paired with retargeted actions, GR00T 2 factorizes the problem into two parts. Future-video prediction learns *what* should happen to the scene, a change that human and robot largely share, while action prediction learns *how* this robot produces that change, from robot play data. To test the formulation, we compare GR00T 2 with GR00T 1.7, a VLA pretrained on the same data and post-trained with the same recipe. The two models also differ in backbone and initialization, so the comparison tests the WAM design as a whole rather than the prediction target alone. GR00T 1.7 fails every chemistry-lab trial, even with 10 teleoperated demonstrations: under the same data and step budget, the VLA learns these precise tasks neither from human demonstrations nor from a few robot demonstrations. The factorization also makes a direct prediction: for a task that only humans demonstrated, GR00T 2 should predict robot hands rather than human hands. Conditioned on one robot start frame, GR00T 2 predicts the robot's own hands reaching each milestone of all four tasks (Fig. 28).

Cross-embodiment pretraining. GR00T 2 transfers task knowledge from the human hand to the robot hand because it pretrains across embodiments. Unlike target-only pretraining that ties task knowledge to one body, cross-embodiment pretraining teaches GR00T 2 to express one task through many bodies. To test this property, we fix the GR00T 2 model and its post-training and replace the full pretraining mixture with target-robot data alone, at a matched pretraining budget. The full mixture adds robot embodiments, human data, and scale at once, so this comparison measures the value of the GR00T 2 mixture as a whole. Without cross-embodiment pretraining, GR00T 2 fails every chemistry-lab trial without teleoperation and trails the full model at every number of teleoperated demonstrations, 45%, 40%, and 30% against 95%, 70%, and 50% at 10.

Human-level motion beyond teleoperation. GR00T 2 achieves human-level motion agility, because learning directly from human demonstrations removes the speed ceiling that teleoperation imposes on robot data. A teleoperator perceives the scene through delayed video, feels no contact, and acts through a mismatched

impedance, so teleoperated demonstrations are inherently slower and more conservative than natural human motion, and policies trained on them inherit this limit. Conveyor Package Tossing tests this directly: it is a targeted throw, not a simple toss. The robot must identify each package, grasp it from the moving belt, and release it at the correct phase of a fast arm swing so that it lands in the bin assigned to that package, a skill that is impractical to teleoperate. The demonstrator performs the task at natural speed in the capture rig, and GR00T 2 learns it from the converted demonstrations and robot play data alone, with no task-specific robot data, succeeding in 50% of trials. Removing the teleoperator thus does more than lower collection cost: it expands the set of behaviors a robot can learn to those that require human-level speed.

B.2. UMI → YAM

Suppose a bimanual gripper robot must learn to take apart an electronics assembly on a factory bench it has never seen: unscrew a nut, lift out a gear and two housings, unplug the connectors and cables, and place every part in a tray. No teleoperation data exist for this robot, and collecting them would tie up the robot, a station, and an operator for the full campaign. With GR00T 2 the task can instead be demonstrated with Gripper UMI, a handheld parallel gripper in the lineage of the Universal Manipulation Interface (UMI, Chi et al., 2024 and Sec. E.2): a person holding the device performs the job with natural motions at natural speed, while robot-style wrist cameras and a width encoder record the demonstration in robot coordinates, with no robot in the loop (Sec. E.1). The difficulty is that the handheld device and the robot are different embodiments. Whereas prior UMI deployments avoid this by construction, with a robot end-effector that reproduces the handheld device's cameras and fingers (Chi et al., 2024), the target robot here, YAM, shares none of these components, and the two embodiments differ in all three respects at once: in sensing because YAM's wrist cameras share neither the lens nor the viewpoint of the handheld device, in actuation because YAM's arm, gripper, and control space have no counterpart in handheld data, and in environment because the demonstrations are collected in the wild and the policy runs in a robot cell the demonstrations never show (Fig. 42).

Closing this gap decouples data collection from the deployment platform, and the decoupling pays off along three axes. The first is generality: the same handheld demonstrations transfer to off-the-shelf robots with their own cameras, grippers, and controllers, so a task collected once need not be re-collected for each platform. The second is capability: the handheld device records motion at human speed with natural contact feedback, so dynamic skills such as tossing, which teleoperation renders slow and conservative, become learnable. The third is efficiency: adding a task requires no robot, no instrumented cell, and no operator training. Demonstrations are recorded in the wild after a few dozen minutes of setup, whereas teleoperation ties up a robot, a station, and a trained operator for the entire collection. The demonstration budget itself is small: with roughly 20 handheld episodes, the policy still completes Toy Dumptruck Assembly end to end (10% success, 81 progress) and still sorts 84% of the conveyor items on Conveyor Belt Sorting and Tossing, and even 10 episodes sort 70% (Tab. 17).

What makes the transfer work. Three design choices carry the transfer, each closing one gap, and Sec. E.3 ablates all of them.

- **Co-training with task-agnostic robot data** closes the embodiment and scene gap. GR00T 2 is post-trained on the UMI demonstrations together with YAM teleoperation data from four general in-lab corpora that contain no episode of the target tasks, typically at a 0.6 : 0.4 ratio: the YAM data supply the embodiment and the scene, the UMI data supply the task. The effect is visible in the model's own predictions, which render YAM's arms rather than the demonstrator's once YAM data are present (Fig. 14).
- **Aligning the wrist view** closes the geometric part of the sensing gap. We re-render the UMI fisheye as a virtual pinhole camera that imitates YAM's wrist view, fitted in Isaac Sim on 112 paired renders (Fig. 43). The alignment is only approximate, yet without it end-to-end success falls to 0% on both long-horizon tasks.
- **Sharing the action head** grounds the task in the robot's own action space. UMI actions are re-expressed in YAM's semantics and decoded by YAM's weights, so the task evidence lands directly in the statistics

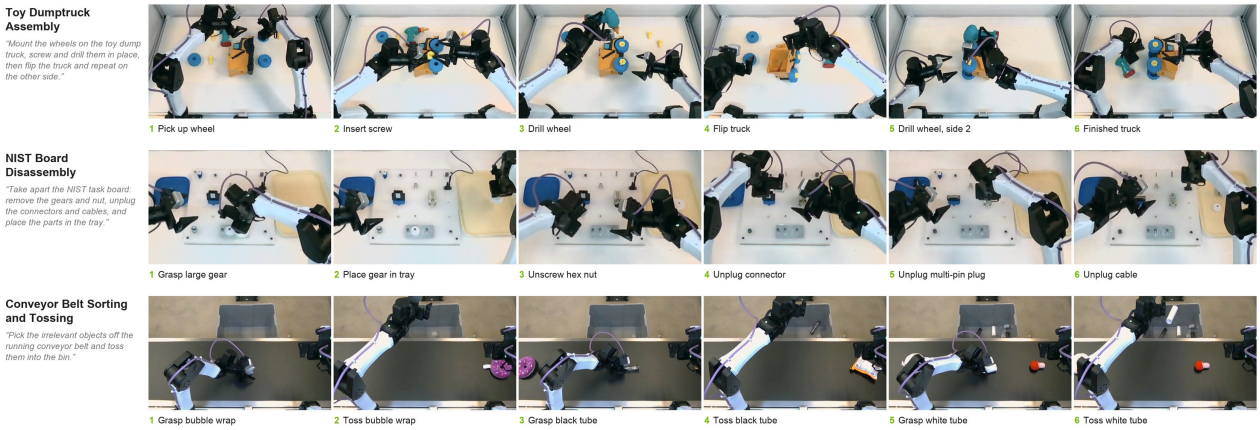


Figure 29: Zero-teleop transfer from UMI to YAM. Each row shows one task with the language instruction GR00T 2 receives and six keyframes of a YAM rollout, numbered by stage: Toy Dumptruck Assembly, NIST Board Disassembly, and Conveyor Belt Sorting and Tossing. Every task is demonstrated only with the handheld Gripper UMI, and YAM has no teleoperation data of these tasks.

the controller executes.

Tasks and protocol. We evaluate on three tasks (Fig. 29): Toy Dumptruck Assembly and NIST Board Disassembly, two long-horizon tasks, an assembly and a disassembly, that chain unscrewing, part removal, flipping, and drilling, and Conveyor Belt Sorting and Tossing, a dynamic task paced by the belt. Each task is demonstrated with 41 to 75 handheld episodes and co-trained with YAM teleoperation data that contain no episode of the target tasks. All models share the post-training recipe and step budget, and the policy predicts 48-step action chunks, of which 30 are executed.

Table 4: Zero-teleop transfer from UMI to YAM. For each model, Prog. is the average substage score and SR the end-to-end success rate on YAM, both on a 0 to 100 scale. Every task is demonstrated only with Gripper UMI (41 to 75 episodes) and co-trained with task-agnostic YAM teleoperation data. All models share the post-training recipe and step budget. GR00T 2 w/o UMI pretraining starts from an earlier GR00T 2 pretraining checkpoint whose mixture contains no UMI data (Sec. E.2). Toy Dumptruck Assembly and NIST Board Disassembly use 10 trials. Conveyor Belt Sorting and Tossing scores each of 50 items as sorted or missed, so its SR is the share of items sorted and it has no separate progress score. The last row averages SR over the three tasks and therefore mixes this per-item rate with the per-trial success rates of the two long-horizon tasks. Bold marks the best Prog. and SR in each row.

	$\pi_{0.5}$		LingBot-VA		GR00T 1.7		**GR00T 2** w/o UMI pretraining		**GR00T 2**	
Task	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR
Toy Dumptruck Assembly	0.0	0.0	0.0	0.0	0.0	0.0	84.0	20.0	**96.0**	**60.0**
NIST Board Disassembly	43.0	0.0	28.0	0.0	50.0	0.0	65.0	0.0	**86.0**	**50.0**
Conveyor Belt Sorting and Tossing	–	0.0	–	0.0	–	28.0	–	34.0	–	**86.0**
Average SR		0.0		0.0		9.3		18.0		**65.3**

Results. With 41 to 75 handheld episodes per task, GR00T 2 reaches 60% and 50% end-to-end success on the two long-horizon tasks, with progress scores of 96 and 86, and sorts 86% of the 50 conveyor items, on a robot never teleoperated for any of these tasks, whereas the baselines $\pi_{0.5}$, LingBot-VA, and GR00T 1.7 complete neither long-horizon task and sort at most 28% of the conveyor items (Tab. 4). The decisive ingredient lies upstream of the recipe: starting from a pretraining checkpoint without UMI data, with identical post-training data, drops success to 20% and 0% on the long-horizon tasks and the share of conveyor items sorted to 34%. A few dozen handheld demonstrations therefore suffice because the embodiments were aligned before post-training began. Sec. E.3 quantifies the three gaps and ablates each component of the recipe.

Table 5: DexUMI post-training results. Fraction of real-robot trials that reach the end of each subtask. *Overall* is the fraction that completes the whole task.

Task	Subtask	$\pi_{0.5}$	GR00T 1.7	GR00T 2
Prepare Banana Yogurt	1. Place banana on board	0.00	0.10	**0.95**
	2. Draw knife, slice, open lid	0.00	0.00	**0.90**
	3. Pieces into bowl, bowl to board	0.00	0.00	**0.60**
	Overall	0.00	0.00	**0.60**
Scoop Granola	1. Pick up jar, hand over, unscrew lid	0.00	0.05	**0.90**
	2. Grab spatula, scoop granola	0.00	0.00	**0.80**
	3. Pass jar back, set it down	0.00	0.00	**0.80**
	Overall	0.00	0.00	**0.80**
Fire Chemistry	1. Grab match and box, strike	0.00	0.10	**0.80**
	2. Light lamp, put match in water	0.00	0.00	**0.75**
	Overall	0.00	0.00	**0.75**

B.3. DexUMI for Complex, Long-Horizon, and Dynamic Post-Training

DexUMI (Xu et al., 2025) extends UMI from parallel grippers to five-fingered hands. We use the GEAR DexUMI exoskeleton (NVIDIA GEAR Team, 2026), which replicates the kinematics, contact surfaces, and camera mount of the XHAND1 Pro-U robot hand, so that every recorded finger joint, fingertip force, and camera view has a robot counterpart. Unlike the original DexUMI, which sensed force with force-sensitive resistors and removed the exoskeleton from video with SAM2 and ProPainter, GEAR DexUMI maps its 16 degrees of freedom one-to-one onto the robot's joints (Sec. E.4), measures three-axis fingertip force in the robot hand's own tactile format, and leaves the video unedited (Sec. E.6). DexUMI provides post-training data for complex, long-horizon, and dynamic dexterous tasks, which GR00T 2 learns with compliance labels derived from fingertip force (Eqs. (1) and (3)).

Because the exoskeleton replicates the robot hand, we post-train GR00T 2 from the GR00T 2 pretraining checkpoint (Sec. E.5) on DexUMI demonstrations alone, without robot play data, and evaluate it on three bimanual tasks on a Tianji robot with two 7-DoF arms and two XHAND1 Pro-U hands (Fig. 30). Prepare Banana Yogurt and Scoop Granola are the two halves of a long make-breakfast task. One policy learns both, and each half is evaluated separately. Fire Chemistry is a short, dynamic tool task: the robot strikes a match, lights a lamp under a flask with it, and puts the match out in water. Each task is scored by subtask:

- **Prepare Banana Yogurt:** (1) place a banana on the cutting board, (2) draw a knife from the knife block, slice the banana, set the knife down, and lift the lid off a bowl of yogurt, (3) drop the banana pieces into the bowl and move the bowl onto the board in the center of the table.
- **Scoop Granola:** (1) pick up the granola jar, pass it to the other hand, and unscrew its lid, (2) grab a spatula and scoop granola from the jar into the bowl, (3) pass the jar back and set it down on the table.
- **Fire Chemistry:** (1) pull a long match from its holder, pick up the matchbox with the other hand, and strike the match on it, (2) light the lamp with the match and put the match out in a cup of water.

The tasks combine tool use (a knife, a spatula, and a match), hand-to-hand transfers, contact-rich twisting to unscrew a lid, and a fast strike that must ignite the match.

Baselines and evaluation. We compare GR00T 2 with two vision-language-action models, GR00T 1.7 and $\pi_{0.5}$ (Physical Intelligence et al., 2025). Each starts from its own pretrained model and is post-trained on DexUMI demonstrations of the same tasks with GR00T 2's data recipe: the same action space, normalization, and prompts. We report the fraction of trials that reach the end of each subtask and the fraction that completes the whole task (Tab. 5).

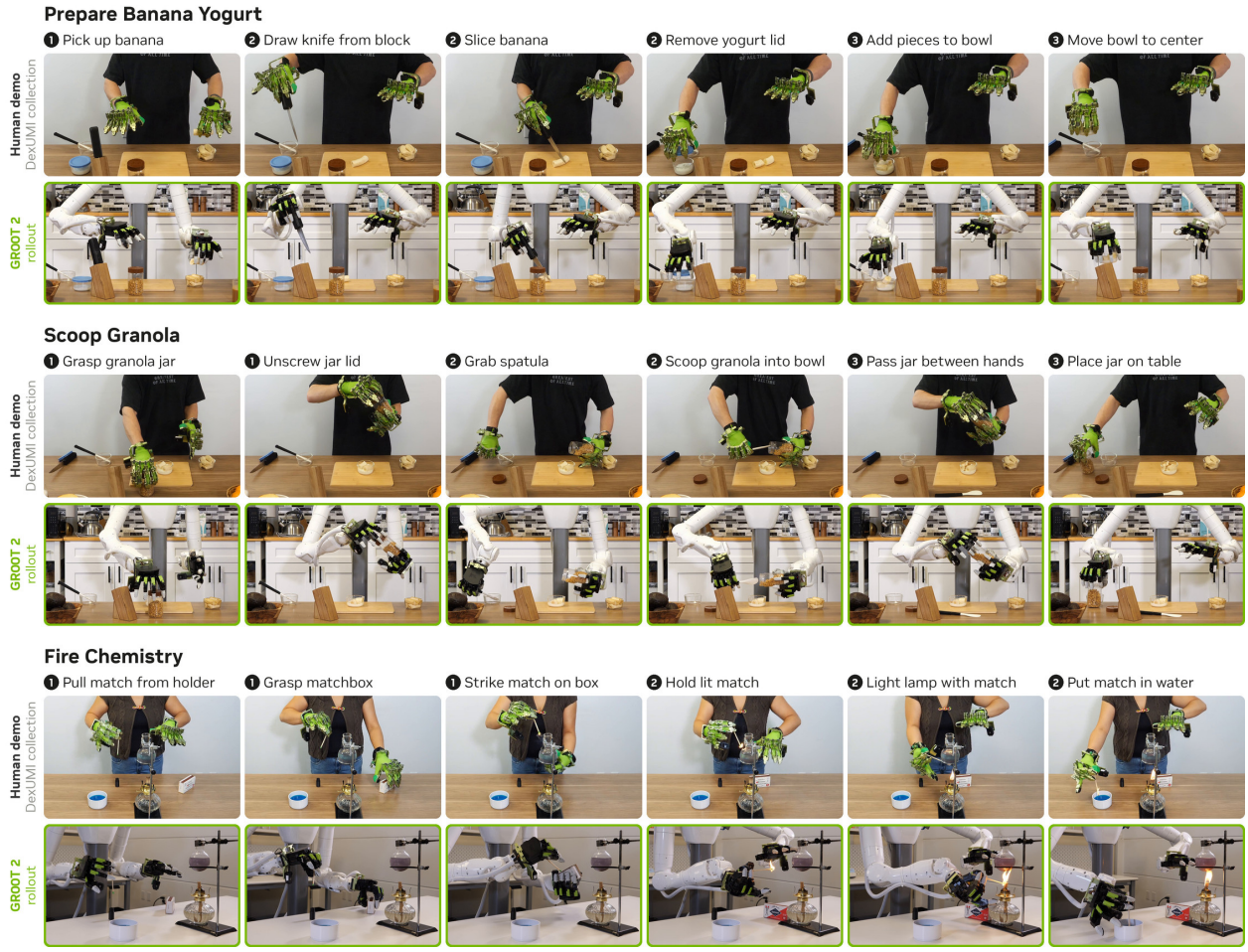


Figure 30: DexUMI tasks. Each block shows one task with six keyframes, numbered by subtask as in Tab. 5: a person demonstrating the task with the DexUMI exoskeletons (top) and a GR00T 2 rollout on a Tianji robot with XHAND1 Pro-U hands (bottom). Prepare Banana Yogurt and Scoop Granola are the two halves of one continuous make-breakfast demonstration and rollout, and every task is demonstrated only with DexUMI.

GR00T 2 performs all three tasks from DexUMI demonstrations alone: it completes Prepare Banana Yogurt, Scoop Granola, and Fire Chemistry in 60%, 80%, and 75% of trials, whereas GR00T 1.7 and $\pi_{0.5}$ complete no trial of any task.

While DexUMI captures detailed data of human manipulation motion, using position label alone is not sufficient to describe the human manipulation strategy. Two more pieces of information are required: the force being applied, and the compliance profile being used. Force encourages forming and maintaining the desired contact modes, while a suitable compliance profile make the manipulation robust against positional uncertainties and even disturbance forces. We adopt the Adaptive Compliance Policy (ACP) (Hou et al., 2025) to give our policy the ability to reason about force and compliance: the DexUMI collected force/motion data are converted offline to stiffness and virtual target labels used to augment the policy action vector. In GR00T 2, we adapt the original ACP formulation for dexterous hands:

Joint space action formulation. The original ACP assumes Cartesian pose as the robot's action, and works the best when the robot has six or more degrees-of-freedoms (DOFs). For dexterous hands with typically three to five joints per finger, the Cartesian space ACP formulation could produce actions that the hand cannot execute. Instead, we define the ACP action in joint space, including position target, virtual target and stiffness.

Joint torque as the unified hand force representation. Different hand or DexUMI hardware may use different force sensing solutions, such as joint torque sensor, force-torque sensor, or tactile sensors. To work with our joint space action, we choose joint torque as the unified force representation for dexterous hands, and project the Cartesian space force measurements, if any, back to the joint space. In the case of the XHAND1 Pro-U, force $f_{i,t}$ is estimated for each finger i from the fingertip tactile sensors, with exponential moving average and a 0.1N cutoff. The fingertip force is then projected onto the actuated joints through the robot hand's Jacobians,

$$\tau_t = B^\top \sum_i J_i(q_t)^\top f_{i,t}, \tag{1}$$

where q_t denotes the finger joint configuration, B maps the actuated joints to the full configuration. We augment the policy action space with the predicted finger joint torque.

Stiffness and virtual targets. Following ACP, we scale the time-varying per joint stiffness value based on the joint torque magnitude:

$$K_{\text{eff}} = s_{\text{acp}}\, k_p, \tag{2}$$

where k_p is the default position gain of the joint, s_{acp} is the adaptive factor that falls linearly from 1.0 to $s_{\min} = 0.3$ as joint torque τ_t rises from 0.1 to 0.4 N m, softening the hand under load. The intuition is that large force can only come from rigid contacts, for which the robot needs to exhibit more compliance. With the policy predicting both position and force, the stiffness 2 is computed online, then used to update the position set point. This is the virtual target a_t in ACP:

$$a_t = q_t + \Delta q_t, \qquad \Delta q_t = \text{clip}\big(K_{\text{eff}}^{-1}\tau_t,\ \pm 1.0\ \text{rad}\big)\,. \tag{3}$$

The hand can achieve the predicted force when tracking the virtual target with the computed stiffness K_{eff}.

B.4. DexUMI → Tianji + Sharpa

GR00T 2 can also learn a task on a dexterous hand from demonstrations recorded with an exoskeleton built for a different hand. The DexUMI exoskeleton replicates the 16-DoF XHAND1 Pro-U hand (Sec. B.3), and the policy runs on the Tianji, a bimanual robot with two 7-DoF arms and two 22-DoF Sharpa hands. Kinematics differ because the Sharpa hand has a different joint layout from the XHAND, with 22 degrees of freedom instead of 16, so the recorded joint angles must be retargeted before they can be used as Sharpa commands. Camera viewpoints differ because the wrist cameras are mounted on the DexUMI wrist during data collection and on the Sharpa wrist during evaluation, and because the head camera is worn by the operator during data

collection but fixed on the robot during evaluation. Appearance differs because the demonstrations show the exoskeleton and the operator's arms.

We evaluate on two tasks. Microscope Slide Dabbing is the chemistry-lab task of the same name in Fig. 26. In Conveyor Package Selection and Tossing, the robot must pick out the packages from mixed objects on a moving belt and toss them into the bin, leaving the other objects untouched on the belt. Each task is demonstrated with 100 DexUMI episodes and co-trained with Tianji + Sharpa play data that contain no demonstration of the target task. We compare against GR00T 1.7 and $\pi_{0.5}$ (Physical Intelligence et al., 2025), post-trained on the same data with the same recipe. GR00T 2 succeeds in 60% and 65% of 20 trials, whereas neither baseline completes a single trial (Tab. 6).

In the DexUMI experiments above, the exoskeleton replicates the robot hand joint for joint, so the recorded joint angles are used directly as hand commands. This setting corresponds to UMI direct (Sec. E.2). Here we use DexUMI demonstrations on a hand that the exoskeleton does not replicate, the 22-DoF Sharpa hand on the Tianji, which is the robot used in the human-to-robot experiments (Sec. B.1). This setting corresponds to UMI $\rightarrow$ YAM transfer (Sec. B.2), and it avoids building a separate exoskeleton for each robot hand.

The recorded joint angles and fingertip forces correspond to the XHAND. The force labels (Eq. (1)) are computed with the XHAND Jacobians and do not apply to the Sharpa hand, so the policy is trained on joint positions only, and the hand tracks position targets in its admittance mode with fixed gains.

Fingertip retargeting. We compute fingertip positions from the recorded joint angles with the XHAND forward kinematics and treat them as the fingertip readings of a motion-capture glove. The closed-form geometric solver used for glove teleoperation (Sec. B.1.2) then places the Sharpa fingertips at these positions, with the two hands aligned at the thumb-index pinch point, and solves the 7-DoF arm for the resulting hand pose. The thumb, index, middle, and ring fingertips are matched to their targets, and the pinky follows the alignment of the hand.

Wrist-view reprojection. The wrist cameras are mounted differently on the DexUMI and Sharpa wrists, and retargeting also changes the wrist pose. As a result, the Sharpa wrist camera ends up about 31° away from the DexUMI wrist camera. We calibrate both camera mounts by hand-eye calibration, solving for each camera's pose relative to the wrist from its image motion and the tracked wrist motion. We then reproject each DexUMI wrist frame into the Sharpa wrist camera, mapping each pixel through a fronto-parallel plane at the depth of the pinch point. During training, random rotations and zooms of the wrist views account for the remaining calibration error.

Co-training with robot play data. The converted demonstrations use the robot's 62-D state and action, consisting of relative wrist poses and absolute Sharpa joint positions, as in the human-to-robot pipeline (Sec. B.1.3). We resample them from 45 to 30 Hz and assign them the robot's embodiment tag, so that they share its action encoder, action decoder, and normalization statistics. They are co-trained with Tianji + Sharpa play data that contain no demonstration of the target task, sampled 75 : 25 for slide dabbing and 50 : 50 for package selection and tossing.

Evaluation. Microscope Slide Dabbing is scored with the rubric of Sec. B.1.4, and a trial succeeds when it completes every stage. A Conveyor Package Selection and Tossing trial succeeds when the packages are tossed into the bin and no other object is touched. Any human intervention counts as a failure. GR00T 1.7 and $\pi_{0.5}$ each start from their own pretrained model and share the post-training mixture, recipe, and step budget of GR00T 2. We report the fraction of 20 trials per task that complete the whole task (Tab. 6).

B.5. Human $\rightarrow$ Simulation $\rightarrow$ Tianji + Sharpa

Despite their broad capabilities, WAMs still struggle with contact-rich dexterous manipulation, such as in-hand object reorientation. Reinforcement learning (RL) in simulation offers a complementary source of skills, including forceful tool use, precise contact interactions, and recovery behaviors that are difficult to demonstrate

Table 6: DexUMI → Tianji + Sharpa results. Fraction of real-robot trials that complete the whole task, over 20 trials per task. No training data contain a robot trajectory of either task.

Task	$\pi_{0.5}$	GR00T 1.7	GR00T 2
Microscope Slide Dabbing	0.00	0.00	**0.60**
Conveyor Package Selection and Tossing	0.00	0.00	**0.65**

reliably through teleoperation or kinematic retargeting. We introduce a **human-to-sim-to-real** pipeline that enables GR00T 2 to acquire these skills from a single human demonstration video per task.

Building on NVIDIA's Video to Data (V2D) (NVIDIA, 2026b) pipeline, we reconstruct 3D object assets, hand–object motion, and contacts from stereo video, then train a state-based RL expert using CHORD (Zhu et al., 2026). We collect expert demonstrations with native simulation rendering, physical and visual randomization, and state perturbations that broaden state coverage and elicit recovery behaviors beyond nominal trajectories. Fig. 16 illustrates the full pipeline.

We fine-tune GR00T 2's diffusion transformer (DiT) on 1.5K simulation episodes, incorporating training-time Real-Time Chunking (RTC) for continuous execution. The resulting policy is deployed zero-shot using RGB observations, robot state, and task instructions, without additional real-world robot data or scene-specific reconstruction or alignment. By expanding one human demonstration into diverse robot rollouts, this approach provides a scalable source of dexterous manipulation data. We hypothesize that GR00T 2's pretrained priors support compositional generalization: fine-tuning combines simulation-learned contact-rich skills with the perceptual and behavioral knowledge acquired during large-scale pretraining.

We evaluate nail hammering, flask handover, and table sweeping, comparing GR00T 2 against GR00T 1.7 and $\pi_{0.5}$. GR00T 2 demonstrates successful transfer of dexterous manipulation skills from simulation to the real world. Although $\pi_{0.5}$ performs well in simulation, its real-world execution is limited to grasping and lifting, with no successful completion of the full tasks, which require subsequent reorientation and object interactions.

Human and Object Reconstruction. Stable simulation and policy training require accurate, geometrically consistent reconstructions of hand motion, object trajectories, and their contacts. Independent estimates of hands and objects can be inconsistent, particularly when the supplied object mesh differs from the object observed in the video. We use V2D's agentic pipeline for 4D hand-object reconstruction, coordinating tools such as SAM3 (Carion et al., 2025), FoundationPose (Wen et al., 2024), and GATOR (Wu et al., 2026) for segmentation, object tracking, and hand tracking. To resolve geometric inconsistencies, the reconstructed hands are refined against the supplied object mesh while keeping the mesh and reference object motion fixed. Contact-wrench objectives and geometric checks guide alignment, followed by penetration correction and finger repair. This formulation adapts hand configurations and contact locations to the supplied geometry while targeting the interaction's mechanical effect on the object. These anchors provide persistent references for downstream robotic grounding. Evaluated on 15 stereo trajectories from SHOW3D (Rim et al., 2026), a dataset of hand-object interactions captured in the wild with annotated 3D hand and object poses, the reconstruction pipeline achieves an average hand reconstruction error of 17.5 mm and an object Chamfer distance of 3.0 mm.

RL in Simulation. Using the reconstructed object assets and trajectories, we train a state-based RL expert to perform dexterous manipulation in simulation. The policy observes privileged robot and object states and outputs arm and hand joint targets. Training follows CHORD, with a warm-up procedure following SimToolReal (Kedia et al., 2026).

We calibrate the simulated robot's joint dynamics through system identification of stiffness, damping, armature, friction, and delay. To improve robustness to physical variation, we randomize robot dynamics, object properties, and scene geometry during training. We also vary initial robot and object configurations, fixture geometry, and table dimensions, while ensuring valid, collision-free initialization.

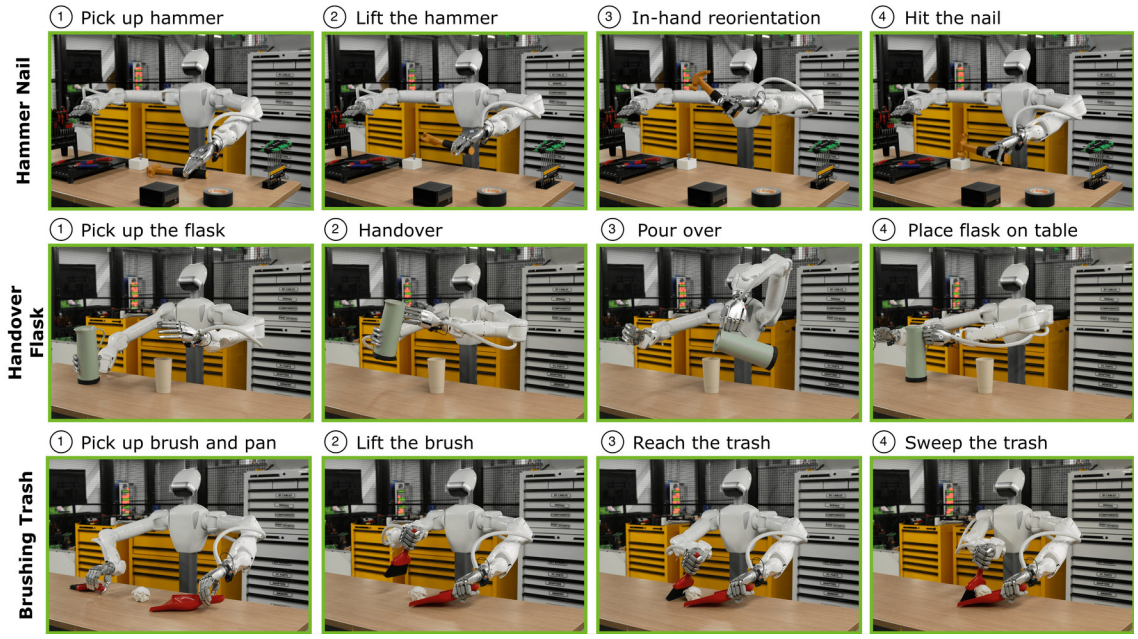


Figure 31: GR00T 2 learns dexterous manipulation through human-to-sim-to-real transfer. Starting from a single human demonstration per task captured with an egocentric stereo camera, the Video-to-Data pipeline generates simulation data for SFT of GR00T 2. The resulting policies transfer zero-shot to the real world on Hammer Nail, Handover Flask, and Brushing Trash, demonstrating dexterous in-hand object reorientation, handover, tool use, and bimanual coordination.

The reward follows CHORD (Zhu et al., 2026) and combines task execution, motion regularization, and contact quality to encourage reliable, human-like tool interactions. Task rewards include keypoint-based tracking of the demonstrated object trajectory, subgoal completion, task progress, and terminal success. Joint-speed penalties regularize motion. Contact rewards use contact wrenches to encourage human-like grasps, improve grasp quality, and sustain object holding. We optimize the policy using a PPO-based SAPG (Singla et al., 2024).

Harvest Dexterity from Simulation. We convert the RL expert's skills into demonstrations for supervised fine-tuning (SFT). Each rollout records RGB observations, robot states, expert actions, and task instructions. The privileged tool and goal keypoints available to the RL expert are excluded from GR00T 2's inputs. We continue to randomize physical properties and scene configurations during collection to capture expert behavior across diverse conditions.

To broaden state coverage and capture recovery behavior, we introduce random-action bursts, object teleports, and robot-joint teleports during data collection. These perturbations move the system away from nominal trajectories and elicit corrective behavior from the expert. The resulting demonstrations cover states that GR00T 2 may encounter when its own actions deviate, helping mitigate the narrow state coverage and compounding errors observed with naive expert-rollout collection.

We also randomize visual appearance using native Isaac Lab (Mittal et al., 2025) rendering, without visual inpainting or style transfer models. Randomization covers object and scene colors, textures, material properties, illumination, environment lighting, and camera calibration. Small perturbations to camera poses and intrinsics model calibration uncertainty, while image noise models sensor variation.

SFT with Simulation Data. We fine-tune GR00T 2 for each task using 1.5k simulation episodes, sourced from one human demonstration video. During SFT, we update the DiT, retain the pretrained action normalization statistics, and freeze the action decoder and video output projection. The fine-tuned policy is deployed zero-shot in the real world using RGB observations, robot state, and a task instruction, with no additional real-world robot data. To support continuous real-time execution of action chunks, we incorporate training-time Real-Time Chunking (RTC).

Table 7: GR00T 2 learns dexterous manipulation skills from simulation demonstrations generated from human videos and transfers these skills zero-shot to the real world. Fine-tuned models are evaluated in simulation (`s`) and the real world (`r`).

Method	Hammer Nail	Handover Flask	Brushing Trash
GR00T 2 (`s`)	**20/20**	13/20	**19/20**
GR00T 1.7 (`s`)	14/20	1/20	11/20
$\pi_{0.5}$ (`s`)	9/20	**16/20**	17/20
GR00T 2 (`r`)	**9/10**	**2/10**	**8/10**
GR00T 1.7 (`r`)	0/10	0/10	0/10
$\pi_{0.5}$ (`r`)	0/10	0/10	0/10

Learning Tasks with Human-to-Sim-to-Real. We evaluate GR00T 2 on three tasks requiring dexterous manipulation, with success defined as follows:

- Hammer Nail: The nail is driven into the fixture until its head contacts the fixture's top surface.
- Handover Flask: The flask is lifted off the table, handed over, and tilted to a pouring angle without being dropped.
- Brushing Trash: Paper waste is swept into the red dustpan using the brush.

GR00T 2 is deployed with 2 action 1 video denoising steps during inference with RTC enabled at 20 Hz. $\pi_{0.5}$ is deployed with the settings recommended in the official implementation: each inference integrates the flow-matching action expert for 10 steps, and the robot executes the first 8 actions of each predicted 30-step chunk (0.4 s at 20 Hz) before synchronously querying the policy again, with $\pi_{0.5}$ observing only the most recent frame of each camera view.

We compare against GR00T 1.7 and $\pi_{0.5}$ in both simulation and the real world. Figure 31 shows execution rollouts for each task. Tab. 7 reports the number of successful episodes for each method. With simulation-only post-training, GR00T 2 achieves 20/20, 13/20, and 19/20 full-task successes in simulation on nail hammering, flask handover, and trash sweeping, respectively (Table 7). In real-world evaluation, it completes nail hammering in 9/10 trials and trash sweeping in 8/10 trials, demonstrating zero-shot transfer on both tasks. Both GR00T 1.7 and $\pi_{0.5}$ achieve 0/10 full-task successes on every real-world task despite making intermediate progress. On nail hammering, GR00T 1.7 lifts the hammer in 10/10 trials and completes in-hand reorientation in 6/10 trials, but never drives the nail to the required depth. On trash sweeping, it lifts the brush in 7/10 trials but never completes the sweep. Similarly, $\pi_{0.5}$ lifts the hammer, flask, and brush in 6/10, 4/10, and 9/10 trials, respectively, but completes none of the full tasks. These results distinguish successful grasp acquisition from full task execution, which requires subsequent reorientation and contact interactions. For the Handover Flask task, the high-frequency specialist policy succeeds in the real world. However, the handover requires precise spatiotemporal coordination and contact control, making it particularly sensitive to observation and inference delays.

C. Data for GR00T 2

This appendix details the sources and roles of data for GR00T 2 summarized in Sec. 4.2.

C.1. Human Data for GR00T 2

Prior work in scaling data for robot learning has showcased emergent transfer and generalization through scaling egocentric data (Kareer et al., 2025; Punamiya et al., 2026; Zheng et al., 2026).

Human-1M is a large and diverse egocentric dataset curated by NVIDIA, which supplies the large scale human experience data in GR00T 2's cross embodiment training mixture. Human-1M contains approximately 1 million hours of proficient human labor sensorized with egocentric cameras, dense language annotation, SLAM and hand pose estimation which are filtered down. The collection spans different environments and sensing configurations, including monocular and stereo head cameras and additional wrist views. We additionally incorporate capture setups that bring human observations closer to those of dexterous robots to examine the transfer of human data at various hierarchies.

C.1.1. Dataset Composition

The Human-1M inventory used within GR00T 2 contains 402,730 hours across approximately 13.75 million episodes, spanning over 51,500 unique collectors and 123,000 scenes.

Collection Settings. We collect demonstrations in both natural "in-the-wild" and controlled studio environments. In-the-wild collection captures proficient laborers in economically valuable scenarios across households, industry workplaces, agriculture, commercial kitchens, service, retail, and logistics. Studio collection concentrates on tabletop manipulation under standardized protocols, resembling demonstration style robotics datasets. Studio data comprises of assembly tasks that involve assembling various kits, curated tasks selected to cover target robot skills and task distributions. Within studio data, "in-lab" collection follows a stricter protocol where demonstrators keep both hands in view, avoid idle periods, and perform motions at speeds chosen to match the robot. Approximately 80,000 hours of the total mixture is studio data. These settings serve complementary purposes. Natural environments broaden the range of interactions observed during pretraining for the world-action model, while controlled collection makes it possible to target particular skills and capture them under more consistent conditions for better transfer to the downstream robot embodiments.

Egocentric Cameras. The dataset includes monocular and stereo egocentric recordings. Monocular setups use a single head-mounted RGB camera. Stereo setups provide synchronized two camera RGB head views with optional additional multi-view cameras for better hand pose estimation.

Wrist Cameras. Approximately 90,000 hours of the collection also includes wrist cameras, which provide close-up views of manipulation and hand-object interactions to reduce partial observability during fine-grained manipulation. Additionally, a portion of the studio data contains robot-aligned wrist viewpoints to address the visual alignment gap.

Even when humans and robots perform the same grasp, wrist-camera placement can change the apparent size and orientation of the hand and object, as well as the visibility of their interaction. We studied this alignment process with the Sharpa platform. We screened 13 initial mount designs and conducted a controlled comparison of four mounts across four operators. The human and robot perform the same four thumb-finger grasps and their viewpoints are compared in a 640x480 image frame. The evaluation measured hand-object overlap, object scale, camera-to-object distance, and camera orientation. We further optimized image rotation and scaling for visual alignment, and used ChArUco (Garrido-Jurado et al., 2014) calibration together with directional motions to optimize angular alignment.

Annotations. Each egocentric episode contains language annotations along with frame-aligned estimations of hand and head motion following prior work on learning from egocentric human experience (Kareer et al., 2025; Punamiya et al., 2026).

Hand and Camera Pose. Hand motion is represented by 21 three-dimensional keypoints per hand: the wrist and four landmarks along each finger, including the fingertips. Keypoint positions are expressed in meters in the reference head-camera coordinate frame and aligned with the corresponding video frames. Head motion is represented by the six-degree-of-freedom pose of the head-mounted reference camera, estimated using simultaneous localization and mapping (SLAM) in a canonical per episode world frame.

Language Annotation. Language describes demonstrations at two levels. Tier 1 (T1) annotations provide a one- to two-sentence episode summary identifying the primary goal, major actions, and key objects. Tier 2 (T2) annotations describe individual actions in contiguous, timestamped segments of 1-10 seconds. Each description is self-contained and specifies the relevant hand, action, and objects. These annotations support language-conditioned learning and task-based data curation. The language annotations are validated by human quality control checks. The highest quality annotations are used for the training mixture of Flora pipeline described in Sec. C.4. The Flora pipeline is then used to further enrich or provide T2 language annotations across the dataset.

C.1.2. Data Filtering and Quality Control

The usefulness of a human demonstration for tabletop manipulation settings depends on both the interaction it contains and the reliability of its recorded motion. We therefore apply quality control at two levels. Batch-level review checks the content and annotations of a delivery, while quantitative filtering selects timesteps with sufficiently stable observations and hand trajectories. Additional details about the quality control are in the accompanying Human-1M dataset release.

Human review. Human auditors check language-annotation validity, task quality and idle time as per our collection specs at a 95% per episode pass rate. These criteria screen deliveries before individual training timesteps are selected by the quantitative filters.

Quantitative Filtering. At each candidate training timestep, we evaluate a window extending 24 frames into the past and 48 frames into the future. We retain the timestep only if all quality criteria pass over this window. The checks constrain head rotation and translation, hand visibility, consistency of reconstructed bone lengths, and wrist speed (Tab. 8). The bone-length error of a hand in a frame is the median absolute deviation of its 20 finger-bone lengths from their per-episode medians, divided by the mean of those medians, so the threshold of 0.05 corresponds to 5% of the mean bone length.

Table 8: Quantitative criteria for selecting training timesteps.

Signal	Window statistic	Acceptance criterion
Head angular speed	Mean	$\leq 20°/s$
Head translational speed	Mean	$\leq 0.2\,m/s$
Hand bone-length error	Maximum	≤ 0.05 (relative)
Hand visibility	Minimum	Both hands in frame
Wrist speed	Maximum	$\leq 1.0\,m/s$

C.1.3. In-Lab Human Data

In-lab data is the part of human data that transfers most directly to a robot policy. Two gaps separate a human demonstration from a robot rollout, and both can be narrowed at collection time rather than left to the model. The *observation gap*: in-the-wild wrist cameras are not placed with reference to any robot, so the hand appears at a different angle, scale, and position than in the robot's wrist view, and the model must imagine across this viewpoint change before it can imagine across embodiments (Sec. B.1.5). The *motion gap*: natural human execution is faster than teleoperated robot execution, interleaves pauses and task-irrelevant gestures, and uses motions the robot hand cannot reproduce. In-lab collection follows our studio collection guidelines, which fix camera placement, motion, and task protocol with the target robot in mind.

Collection protocol. Demonstrators keep the head still, keep both hands and the manipulated objects visible in every frame, avoid idle time, rehearse each task before recording, and follow a reference video where one exists. Pace is controlled through bounds on wrist speed, which exclude motion too fast for the robot on one side and near-stationary hands on the other. Each task is recorded by several operators in several environments, and episodes that a robot could not reproduce are rejected before delivery. The complete criteria, including the acceptance thresholds for the wrist camera and for hand-pose quality, are listed in Sec. C.1.4.

Alignment to the target embodiment. The target is Tianji with Sharpa hands, the platform of the human-to-robot transfer (Sec. B.1), whose teleoperation system is described in Sec. C.3.1. On the observation side, the human wrist camera is mounted to reproduce the view of the robot's wrist camera. It is accepted only when the hand and the object appear at the same scale and position as in the robot view. The head camera stands in for the robot's head camera and is held still, and demonstrators whose hands are close to the robot hand in size are preferred. On the motion side, tasks mirror the robot's teleoperation tasks with the same objects and step order, robot-paired tasks come with reference videos recorded on the robot, and the speed bounds keep the pace within what the robot can execute. In-lab episodes pass through the same filtering and annotation as the rest of Human-1M (Tab. 8, Sec. C.4), and their hand keypoints can be retargeted with the solver of Sec. B.1.2 into the observation and action schema of a teleoperated Tianji episode (Sec. B.1.3).

Scale and diversity. In-lab collection has delivered about 3,000 hours and 700k episodes. The tasks are drawn from more than 2,000 task specifications, which span short single-goal tabletop tasks that each isolate a grasp or a skill, robot-paired tasks with fixed objects and reference videos that stress dexterity and dynamics, longer multi-step chores, and tasks paired with UMI reference demonstrations (Sec. C.2), so that the same task exists as human, UMI, and robot data. Episodes are short and dense, and each task is demonstrated hundreds of times by different operators, which gives the model many views of the same manipulation rather than many unrelated clips.

C.1.4. In-Lab Human Data Collection Criteria

This section lists the acceptance criteria of our studio collection guidelines (v1.3.2) for the in-lab human data of Sec. C.1.3. The wrist-camera criterion is checked once per mount design: the demonstrator repeats, with the same 4.4 cm calibration block, the calibration motions recorded from the robot's wrist camera (a thumb pinch with each finger and a whole-hand grasp), and the human video is overlaid on the robot video. The remaining criteria are checked per delivery. Tab. 9 summarizes them.

Table 9: In-lab collection criteria.

Criterion	Requirement
Head motion	Stable head pose, no rotation
Hand visibility	Both hands in 100% of frames
Idle motion	None, brief task-relevant inspection allowed
Wrist speed	$< 5\%$ of samples at ≥ 0.40 m/s, $\leq 60\%$ below 0.03 m/s, follow the reference video pace
Wrist camera	Calibration block at 0.8–1.2$\times$ its size in the robot view, human-robot hand IoU > 0.5 with arms extended
Hand-pose quality	Speed-conditioned jitter $\leq 4\times$ the robot reference per speed band, per-frame confidence, audited reprojections
Operators	One per environment, hand size close to the robot hand preferred
Demonstrations	400 per task: 8 environments, 50 each
Objects	Robot-paired tasks: specified SKU, or 3 variants of similar shape and size
Deliverables	Head and wrist intrinsics, stereo extrinsics, per-delivery wrist-speed statistics

C.1.5. Scaling Human Video

GR00T 2 pretrains on egocentric human video from Human-1M as well as on robot data. To measure how much the human video contributes to robot performance, we train four models that differ in how much human video they see. We test each one on new tasks on Dexter, a bimanual robot with two 7-DoF arms and two 22-DoF Sharpa hands, and on the video rollouts each model generates for the same tasks.

Setup. The four models share the GR00T 2 architecture and train on the same Dexter teleoperation data with the same recipe. They differ only in the human video they see. The 0% model sees none. The other three start from a common human-video pretraining checkpoint, then continue on 1%, 10%, or 100% of the tasks in the multi-view human video of Human-1M (0.87k, 8.1k, and 84.4k hours after the timestep filters of Tab. 8) and co-train on it with the Dexter data. The 0% model does not have this shared stage, so comparisons with it measure the shared stage and the human share together, whereas the 1%, 10%, and 100% models differ only in the share and the matching training length. Training length is matched in epochs. Each human share is pretrained for about 0.5 epoch of that share, so the larger shares also take more steps, and every model, including the 0% model, is then trained for 3 epochs of the Dexter data, where one epoch passes once over every step of every 30 Hz trajectory. During this stage, the 1%, 10%, and 100% models draw 45% of their training samples from Dexter and 55% from human video, while the 0% model trains on the Dexter data alone. Multi-view human video is recorded with wrist cameras as well as a head camera, like the robot. GR00T 2 itself pretrains on 403k hours of Human-1M. We pick each model's checkpoint offline with the same rule for all four, without robot evaluation.

We test on 72 Dexter tasks, giving each model only the task instruction. 22 of them are out-of-distribution (OOD) tasks. They use objects and pick-and-place skills from Dexter's training data, with new kinds of instructions such as color, negation, size, order, count, and spatial relations. The other 50 never appear in Dexter's training data, and 47 of them are in Dexter-50. Each model runs 3 episodes per task, for 864 episodes in total. On each task, the four models run back to back on the same station and scene, in the order 0%, 1%, 10%, 100%. One operator grades all of them on whether the task's end state is reached. We also score task progress, the fraction of the task completed, with a vision-language model on the real robot rollout video from the head and both wrist cameras.

GR00T 2 also predicts video, so we evaluate the same 72 tasks without a robot as well. From 3 recorded start scenes per task, each model generates video rollouts for the task instruction with 10 generation seeds, for 2,160 videos per model. The same vision-language model judges whether a rollout performs the commanded action on the target object (video success) and scores its task progress with the same prompt as the real episodes.

Results. All four metrics rise with the amount of human video (Fig. 10). On the robot, success goes from 18% without human video to 27%, 28%, and 41% with 1%, 10%, and 100%, and task progress from 50% to 58%, 61%, and 70%. In video rollouts, success goes from 30% to 33%, 38%, and 51%, and task progress from 64% to 66%, 72%, and 77%. Using all of the multi-view human video instead of none raises success on the robot by 23.1 points (95% confidence interval 13.4 to 33.3, paired over tasks) and task progress by 20.1 points (12.3 to 27.9). In video rollouts it raises success by 21.3 points (14.3 to 28.2) and task progress by 12.4 points (7.8 to 17.1).

From 1% to 100%, task progress on the robot gains about 6 points for each tenfold increase in human video (2.5 to 9.6), and video success about 9 points (5.9 to 12.0). The step from 10% to 100% is clear on every metric. On the robot, success gains 12.5 points (4.2 to 21.3). The step from 1% to 10% is within noise on the robot (success +0.9 and progress +3.2 points), but shows in video rollouts (success +4.7 points, 0.1 to 9.4, and progress +6.2 points, 2.6 to 9.9), which average 30 rollouts per task instead of 3. None of the metrics levels off at 100%, which suggests that more human video would help further. Each data share is a single training run, so the error bars show how much the results vary across repeated rollouts, not across training runs.

Table 10: UMI-10K for GR00T 2. Gripper UMI episodes in the pretraining corpus.

Hours	10,184
Episodes	942k
Tasks	2,897
Skills	49
Object categories	356
Episodes with a non-pick-and-place skill	88%
Episodes with a dynamic skill	29%

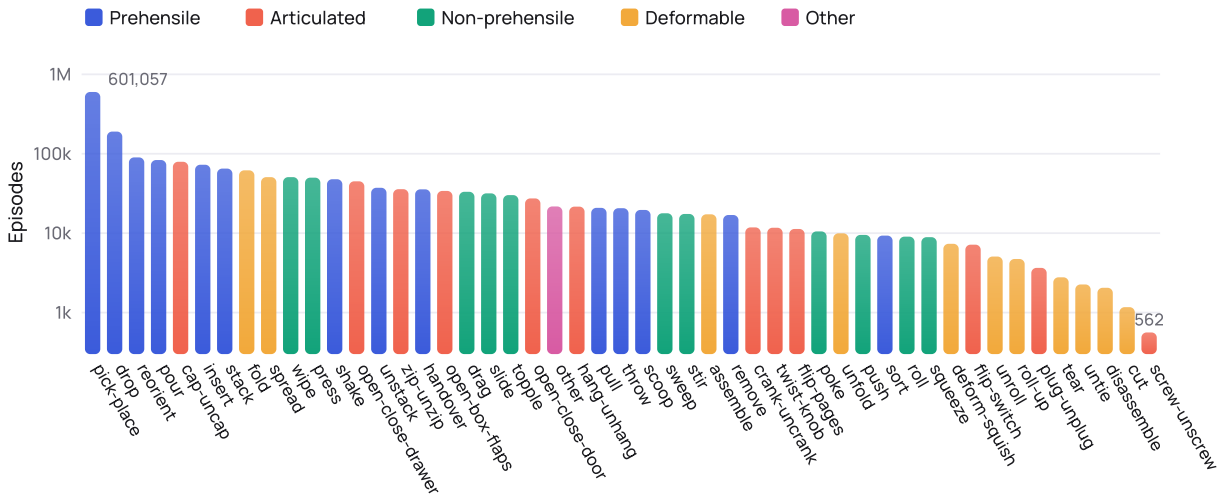


Figure 32: Skill distribution of the Gripper UMI corpus. Episodes per skill on a logarithmic scale, colored by skill group. An episode counts toward every skill its instruction exercises. Pick-and-place is the most common skill, but 88% of episodes also exercise at least one other skill.

C.2. UMI-10K for GR00T 2

Scale. The Gripper UMI corpus (App. E) contributes 942k quality-controlled episodes and 10,184 hours of bimanual parallel-gripper demonstrations across 2,897 tasks to GR00T 2 pretraining (Tab. 10). The corpus is built for breadth, focusing on short to mid-horizon tasks with wide behavior coverage, rather than depth on a few specialized tasks: no task accounts for more than 1.2% of the hours, and the instructions span 356 object categories and 49 skills in our manipulation vocabulary.

Task distribution. We characterize the corpus by tagging every task instruction with a 49-skill vocabulary organized in four groups by manipulation and object type: prehensile, non-prehensile, articulated, and deformable, the same vocabulary that drives task generation (below). The skill distribution is shown in (Fig. 32): pick-and-place appears in 64% of episodes, while 88% of episodes also exercise at least one other skill to ensure diversity and complexity of actions captured. 29% of episodes involve a dynamic skill such as throwing, toppling, shaking, or sweeping, and 17% involve a precision skill such as insertion or capping. Articulated-object skills, drawers, lids, zippers, box flaps, doors, knobs, and cranks, appear in 29% of episodes, and deformable-object skills, folding, spreading, rolling, tearing, and tying, in 15%. Tasks are graded into three difficulty tiers along two axes, the number of sequential subgoals and the share of more complex motion beyond pick-and-place, which both increase with difficulty tier. By hours, 41% of the corpus is simple (two or more subgoals, any motion), 38% moderate (three or more state-dependent subgoals with at least one non-pick-and-place skill), and 21% complex (four or more state-dependent subgoals dominated by non-pick-and-place skills).

Task design. Tasks are generated using a language-model based agent which pairs skills from the vocabulary

Table 11: Robot teleoperation data in the GR00T 2 pretraining mixture. Share is the fraction of robot-data samples drawn from each source.

Source	Platform (end effector)	Episodes	Hours	Share
Bimanual, parallel gripper				
YAM	YAM (gripper)	610.7k	15,146	51.2%
ABC-130k (XDOF, 2026)	YAM (gripper)	130.7k	3,591	10.4%
Bimanual, dexterous hands				
Sharpa	Bimanual arms (22-DoF Sharpa hands)	202.6k	2,890	20.6%
Wuji	Tianji 7-DoF arms (20-DoF Wuji hands)	80.9k	1,256	14.0%
Humanoid				
G1	Unitree G1 (gripper)	57.7k	275	1.3%
Single arm				
DROID (Khazatsky et al., 2024)	Franka Panda (gripper)	76k	350	2.5%
Total		1.16M	23.5k	100%

with objects from an object inventory into task templates, which are linted, deduplicated against every existing task. A human then inspects each task and manually accepts or rejects it. Each template is instantiated into concrete tasks by sampling different object, count, color, and position, which yields counterfactual pairs that share a scene but differ in one attribute of the instruction, and each generation round under-samples skills that are already common. The result is broad compositional coverage: 2,798 distinct skill-object pairs, with 70% of the top-24-skill by top-30-object grid populated.

Quality control. Every episode passes a three-stage quality pipeline before it enters the corpus. First, an autonomous station-side check rejects recordings with uneven frame timing, dropped frames, or tracking and gripper faults at the point of capture. Second, an agentic QC pipeline reviews each remaining episode from its videos and instruction and scores four aspects: task completion (every step of the instruction is carried out on the objects, counts, and targets the instruction names, and the result is left in the specified state), motion quality (no micro-adjustments, jerky or abrupt movements, or idle hesitation), scene validity (a clean workspace without distracting or occluding objects or people), and view quality (both grippers and the task area visible in every camera); an episode is rejected on any critical failure or on a low aspect score. Third, human reviewers audit the agent's verdicts on sampled and disputed episodes against a golden set, and their corrections feed back into the judge. Together these stages filter out roughly a quarter of recorded episodes (about 11% at the station and 16% at the agentic gate), so the corpus contains only clean, instruction-faithful demonstrations whose motion reflects natural human manipulation: the 95th-percentile gripper speed within an episode is 0.28 m/s on each arm, and peak speeds exceed 0.68 m/s in a tenth of the episodes.

C.3. Teleoperation Data for GR00T 2

GR00T 2 pretrains on about 23.5k hours and 1.16M episodes of teleoperation from six sources. They span bimanual parallel-gripper arms, bimanual arms with dexterous hands, a humanoid, and a single-arm manipulator (Tab. 11).

Embodiments and sources. Bimanual parallel-gripper data come from YAM arms and make up the largest block: 18.7k hours covering a broad range of household and tabletop tasks, including 3.6k hours from ABC-130k (XDOF, 2026). Dexterous data come from bimanual arms with multi-fingered hands. The Sharpa data (22-DoF Sharpa hands) cover 2.9k hours, and Wuji (20-DoF Wuji hands on 7-DoF Tianji arms) adds 1.3k hours. Humanoid data come from the Unitree G1 (App. H), and DROID (Khazatsky et al., 2024) contributes single-arm Franka data from 564 scenes. Every bimanual platform records a head or top-down view and two wrist views.

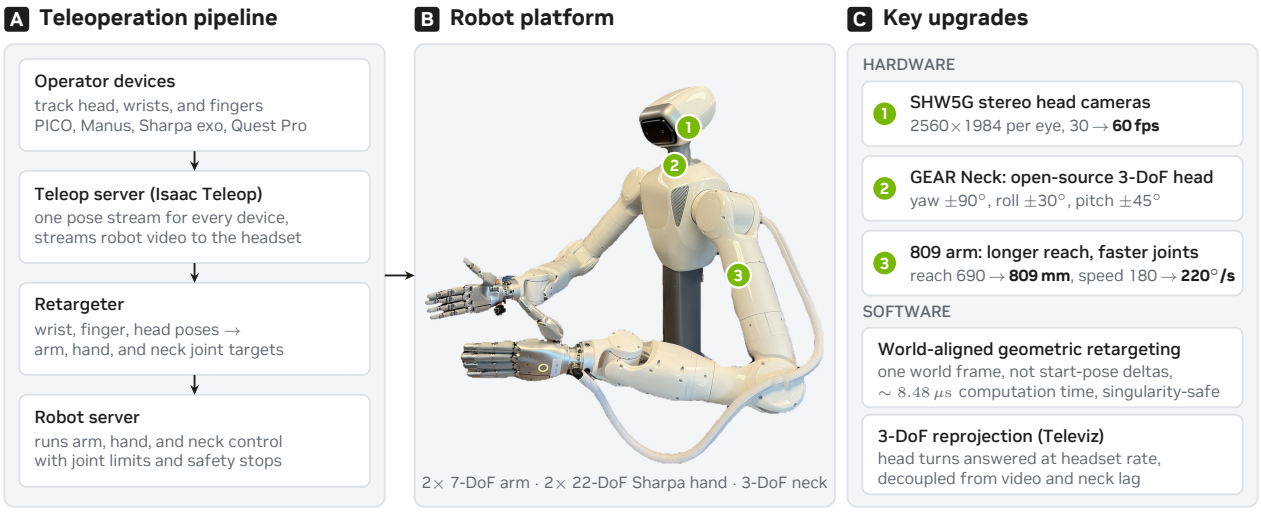


Figure 33: Teleoperation system for Tianji + Sharpa. (A) Teleoperation pipeline from operator devices to the robot. The retargeter is the solver used for human-data conversion. (B) Robot platform: Tianji arms, Sharpa hands, and the GEAR Neck. (C) Key upgrades: hardware, numbered at their location in (B), and software.

Mixture. Sampling shares are set per source rather than in proportion to hours (Tab. 11). YAM holds 64% of robot hours but receives 51% of samples. The dexterous Sharpa and Wuji data together hold 18% of hours and receive 35% of samples.

C.3.1. In-Lab Teleoperation System for Tianji + Sharpa

The sources in Tab. 11 form the pretraining mixture. In addition, we collect robot data in our lab, mainly for post-training, with the Tianji + Sharpa teleoperation system in Fig. 33. It shares its devices and retargeting with the human-data pipeline (Sec. B.1.1 and B.1.2) and collects, for example, the task-agnostic play data used to learn new tasks without teleoperation (Sec. B.1.3) and the 1 or 10 target-task demonstrations added for comparison (Sec. B.1.4).

Platform and pipeline. Tianji + Sharpa pairs two 7-DoF Tianji arms with two 22-DoF Sharpa Wave hands. A longer upper arm (288 to 400 mm) recovers the workspace in front of the chest, and a 220 °/s joint limit keeps up with human wrist motion. A teleoperation server ingests all devices through Isaac Teleop, a retargeter maps them to arm and hand commands, and a robot server executes them.

Active head without motion sickness. The cameras ride on the GEAR Neck, our open-source 3-DoF neck that follows the headset, and stream at 60 fps for lower latency and less blur. A moving head camera normally causes motion sickness, because the image lags the operator's head turn by the capture, transport, and neck delay. Our 3-DoF reprojection in Televiz renders each frame at its capture-time orientation and lets the headset re-render it against the current head pose at its own rate, so head rotation is answered locally and end-to-end latency no longer drives operator comfort.

C.4. Data Annotations

We built Flora (Leng et al., 2026), the data engine that annotates training data for GR00T 2 models. Flora consists of three models: Flora-TempSeg localizes actions in time, Flora-Captioner describes each localized action, and Flora-Rewarder produces outcome, dense, and rubric-based rewards. They are trained on open-source, licensed, and in-house data, where human annotators provide dense timestamped captions and action-quality labels under detailed labeling guidelines. The same models label human egocentric video and robot recordings from parallel-gripper and dexterous-hand embodiments, with one or more synchronized camera views.

Timestamped captions. Given an episode video, Flora outputs a set of time intervals, each paired with a caption of the action performed in it. Flora-TempSeg runs on overlapping windows and predicts the start and end time of each action in a window, together with flags that mark actions continuing past the window edges. These predictions are mapped to episode time and stitched into episode-level intervals, while concurrent actions, such as two hands doing different things, remain separate intervals. Flora-Captioner then captions each interval independently. In addition to the frames of the interval, it sees frames from the neighboring actions before and after it, which help resolve object identity and intent, while the caption describes only the target interval. Synchronized camera views are tiled into one image per time step. Each caption is a self-contained imperative instruction that names the intended action, the objects, the role of each hand or arm, and relevant spatial relations, for example "hold tea leaves with right hand, pass tea leaves to left hand". Failed attempts are captioned with their intended goal, without implying success. Both models are first fine-tuned on human-annotated captions and then trained with GRPO (Shao et al., 2024) on verifiable rewards: temporal F1 and boundary accuracy for Flora-TempSeg, and reference-grounded coverage of caption requirements with a penalty for factual errors for Flora-Captioner.

Quality and cost. On WGO-Bench (Macrodata Labs, 2026), Flora achieves the best temporal segmentation F1 of all evaluated models, 50.3 versus 46.0 for the next-best model, GPT-6 Astra (OpenAI, 2026). On our 300-episode in-house captioning benchmark, it leads all models on human episodes and ranks second to Astra on robot episodes.

Annotating GR00T 2 data. For GR00T 2, Flora turns raw episodes into training-ready supervision. These instructions provide fine-grained language conditioning for pretraining. At the production rate above, relabeling the roughly 400k hours of Human-1M data used by GR00T 2 takes about two days.

D. Real-World Benchmarking with GearArena

This appendix describes how we benchmark GR00T 2 on real robots: why we aggregate real-robot evaluations into a single leaderboard and why that leaderboard is built from blinded A/B comparisons (Sec. D.1), how the evaluation tasks are designed (Sec. D.2), how the arena is organized (Sec. D.3), the rating model (Sec. D.4), the A/B evaluation infrastructure (Sec. D.5), what the arena shows about GR00T 2 (Sec. D.6), and its limitations (Sec. D.7). Tab. 12 summarizes the evidence behind every number in this appendix.

Table 12: GearArena at a glance (snapshot of November 6, 2026, 12:28 PT). A campaign is one A/B evaluation request, a trial runs every arm once on the same scene, a rollout is one graded robot episode, and a match is one pairwise comparison inside a trial. The three suites are separate leagues whose ratings are not comparable with each other. Five YAM campaigns ran tasks of both YAM suites and are counted in both columns, so the two YAM suites hold 195 distinct campaigns. †Dex records list stations by service identifier and do not record the operator.

	YAM V1	YAM V2	Dex
Robot	YAM (bimanual)	YAM (bimanual)	Tianji arms + Sharpa hands
Task families	general, dexterous	language, motion	dexterous
Rated checkpoints	45	26	25
A/B campaigns	78	122	29
Stations / operators	6 / $\leq$17	10 / $\leq$23	5 / –†
Graded trials	2,167	3,403	721
Graded rollouts	6,373	9,671	2,146
Pairwise matches (ties)	5,732 (30%)	6,035 (35%)	1,492 (33%)
Tasks with matches	16	74 names (38 tasks)	56
Evidence window	Jul 15 – Nov 6	Sep 21 – Nov 6	Sep 29 – Oct 29

D.1. Why Aggregate Real-Robot Evaluations, and Why A/B

Real-robot evaluations are expensive but cumulative. A real-robot evaluation costs robot time, an operator, a staged scene, and grading. Each one answers a narrow question, but over months the evaluations of a project add up to far more evidence than any single study could afford. The question is whether that evidence can be pooled: a model trained in October should be comparable with one trained in July, even though they were never run side by side.

Absolute scores drift. Success rate and rubric score are not stable across evaluation sessions. They depend on the station (arm calibration, cameras, lighting, wear), on the operator (how a scene is staged and how borderline outcomes are graded), and on the day. Two checkpoints evaluated in different sessions therefore differ by their skill *and* by the drift between the sessions, and the two cannot be separated after the fact. The arena makes this concrete: it can also construct *derived* comparisons from independent evaluations of two checkpoints on the same task within 45 days, but because those comparisons cross sessions, it down-weights them to at most a quarter of a blinded comparison and excludes them from the default leaderboard.

A/B comparisons cancel the drift. In a blinded A/B trial, two to four checkpoints run in randomized order on the same staged scene, on the same station, by the same operator, within minutes, and the operator does not know which checkpoint is which (Sec. D.5). Whatever the session does to one checkpoint it does to the others, so the *comparison* is free of the drift that corrupts the absolute score. Pairwise outcomes from many such trials can then be fit jointly into one rating scale (Sec. D.4), in the same way that Chatbot Arena ranks language models from pairwise human preferences (Chiang et al., 2024). Three differences from that setting shape our design. Robot checkpoints are frozen, so all history can be pooled in one batch fit rather than updated sequentially. The comparisons from one trial share a scene and are correlated, so uncertainty must be estimated at the level of scenes. Each checkpoint has tens to hundreds of comparisons rather than thousands, so confidence intervals are essential.

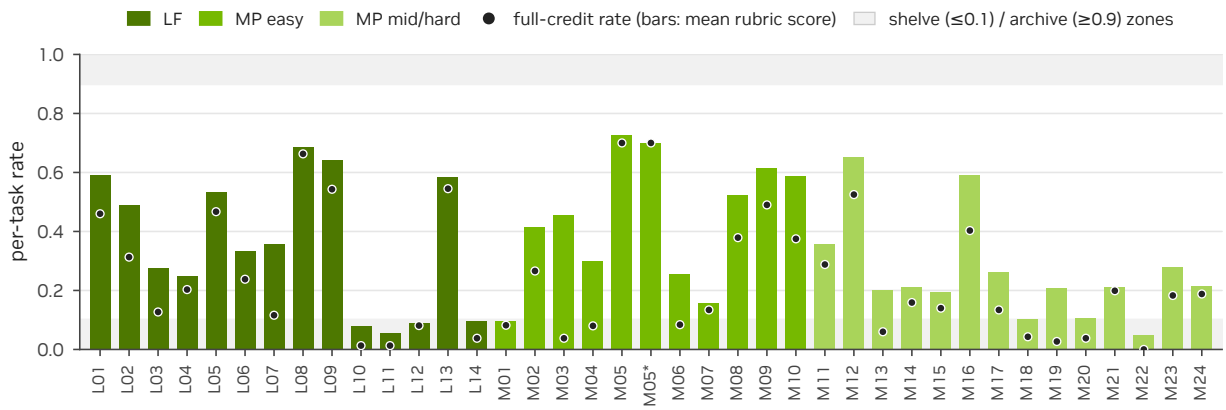


Figure 34: Gauging the current suite. Mean rubric score (bars) and full-credit rate (dots) per task over all graded A/B rollouts on YAM V2, grouped by sub-suite and ordered by task code (M05* is a variant of M05 with 20 rollouts). No task is near the ceiling, and six sit in the shelving zone, where nearly every rollout fails.

D.2. Designing Evaluation Tasks from First Principles

Want, represent, gauge. Our task suites are designed in three steps. First, we ask what the robot should be able to do: follow spatial and attribute-based language instructions, execute basic motion primitives reliably, and handle contact-rich and deformable-object skills. Second, we choose the tasks that best represent each capability, one physical arrangement per task, each with a partial-credit rubric that scores a rollout as the weighted fraction of completed sub-tasks between 0 and 1 (the task-progress metric of Sec. J.2.1). Third, we gauge every task on real checkpoints. A task that every checkpoint completes no longer separates anything and is archived, while a task that no checkpoint completes is shelved until the models catch up. Shelving is also enforced by the rating itself: a comparison in which both rollouts score exactly zero is dropped (Sec. D.4), so a task nobody can do contributes nothing. In the first suite, the zipper-closing task scored zero in every trial for every checkpoint, and drops out this way.

Two suites. The first suite (YAM V1, July to October 2026) combined general manipulation tasks (wiping, slot insertion, unscrewing, opening and closing, scooping, flattening and folding, language following) with dexterous assembly and packing tasks (airplane and car assembly, iPhone box). The current suite (YAM V2, from September 21, 2026) has 38 tasks in three sub-suites: 14 language-following tasks (LF, e.g. place the cup between the bowl and the plate, put the second block from the left in the basket), 10 easy motion primitives (MP easy, e.g. flip a cup, pour beans between bowls), and 14 medium-to-hard motion primitives (MP mid/hard, e.g. screw a lid onto a jar, thread a cable tie into a bottle, roll a towel, unhook rings). Each A/B campaign on V2 runs one sub-suite with two trials per task per checkpoint.

The current suite is unsaturated. Fig. 34 applies the gauge to V2. Across 9,402 graded rollouts (the V2 rollouts of Tab. 12 without those of VLM-based policies and of one uncoded task), per-task mean scores range from 0.05 (rolling clay) to 0.73 (cup in basket), with a median of 0.28. No task is close to saturation, and six tasks (rolling clay, wiping a circle, wiping a blue mark, picking the second block from the left, stacking the middle block, and pushing a cup left) score at most 0.10 on average, which makes them candidates for shelving. The suite therefore still separates the strongest checkpoints, which is the property that makes its comparisons informative.

Task identity. The rating treats each task name as one physical arrangement (one *scene*), so names matter. Several V2 tasks were recorded under two identifiers, a bare name and a coded one (e.g. `T_Motion_Roll_Towel` and `T_Motion_Roll_Towel_M17_U`), and the live leaderboard counts them as different tasks.[2] Admission to a

[2]Merging the two identifiers of each task (74 task names to the suite's 38 tasks) changes no rating by more than 8 Elo, leaves the top eleven ranks unchanged, and moves every reported $P(A{>}B)$ by at most 0.02 (offline re-fit on the same snapshot).

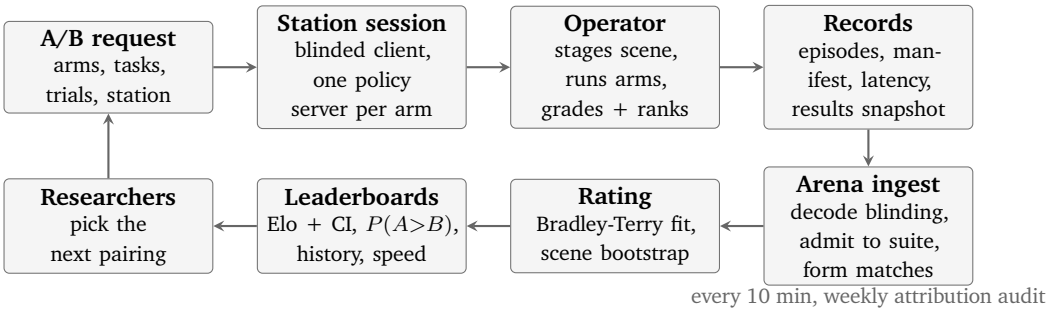


Figure 35: From an A/B request to a rating. A campaign is requested with its checkpoints, tasks, trial count, and station. The station runs a blinded session in which the operator stages each scene, runs every arm, and grades and ranks the rollouts without knowing which checkpoint produced which. Recordings, the blinding manifest, latency statistics, and a checksummed results snapshot are stored with the session. The arena ingests them, decodes the blinding, admits each graded trial to its suite, and refits the ratings. Researchers choose the next comparisons from the leaderboard.

suite is explicit: each suite lists its task identifiers and the embodiment tags it accepts, so evidence cannot leak between suites.

D.3. The Arena: Stations, Operators, Campaigns, and Suites

Campaigns and sessions. The unit of work is the *campaign*: one A/B evaluation request naming two to four checkpoints, a task list, the number of trials per task, and a station. A campaign runs in one or more recording sessions on that station, typically by one operator. In the snapshot, the two YAM suites hold 195 campaigns run on 10 YAM stations by more than 20 operators, with 16,044 graded rollouts in 5,570 trials (Tab. 12). Campaigns are collected continuously. The leaderboard counts graded trials of campaigns still in progress, and never counts an ungraded trial as a failure.

Suites as separate leagues. Each suite is its own league with its own evidence, ratings, and history: YAM V1, YAM V2, and a dexterous-hand suite on Tianji arms with Sharpa hands (Dex, 25 checkpoints, 29 campaigns, 1,492 matches). Ratings are comparable only within a suite. Within the dexterous suite, inference settings are part of a checkpoint's identity: the same weights served with two different inference regimes are rated as separate entries, and disconnected groups of checkpoints that never met are reported as separate comparison groups rather than forced onto one scale.

Provenance and audit. The arena reads each campaign's graded results from a checksummed snapshot stored with its recordings, falling back to the request record only when no snapshot exists, and records which source it used. A board history keeps every change point, so any past leaderboard can be replayed. A weekly audit re-checks every scored trial against its recording, and a campaign whose blinding is in doubt is withheld from every leaderboard (one campaign is withheld today, because its operator could see checkpoint labels).

D.4. Leaderboard Mathematics

From trials to matches. In a trial with n arms, every pair of arms forms one match, $\binom{n}{2}$ in total. The outcome is the operator's explicit ranking of the rollouts. When the operator graded two rollouts identically, the forced order between them carries no information, and the match counts as a tie. When both rollouts scored exactly zero, the match is dropped, since it bounds both checkpoints below the task's difficulty without ordering them.

Bradley-Terry model. Each checkpoint i has a strength $p_i > 0$, and

$$P(i \text{ beats } j) = \frac{p_i}{p_i + p_j}. \tag{4}$$

The strengths are fit jointly to all matches by maximum likelihood (Bradley and Terry, 1952), using the minorization-maximization update (Hunter, 2004)

$$p_i \leftarrow \frac{W_i}{\sum_{j \neq i} n_{ij}/(p_i + p_j)}, \tag{5}$$

where W_i is checkpoint i's total weighted wins (a tie counts half) and n_{ij} the total weight of matches between i and j. Each checkpoint also receives one virtual tie against a fixed anchor, a weak prior that keeps undefeated or winless checkpoints finite and pulls checkpoints with little evidence toward the middle. Because the fit is a batch estimate, it does not depend on the order in which evaluations arrived. Ratings are reported on the Elo scale (Elo, 1978),

$$\mathrm{Elo}_i = 1500 + 400 \log_{10} \frac{p_i}{\bar{p}}, \tag{6}$$

with $\bar{p}$ the geometric mean of all strengths, so 1500 is the field average and a difference of 100 Elo corresponds to a 64% expected win rate.

Task normalization. A raw fit would let a frequently evaluated task dominate. We instead normalize within each head-to-head: for every pair of checkpoints and every task, the matches of that pair on that task are rescaled to total weight one,

$$w_m = \frac{1}{\left|\{m' : \mathrm{pair}(m') = \mathrm{pair}(m),\ \mathrm{task}(m') = \mathrm{task}(m)\}\right|}, \tag{7}$$

so that within any head-to-head each task contributes the fraction of its trials won, and each distinct task counts once. Normalizing per pair, rather than per task across the whole leaderboard, keeps a checkpoint's weights independent of evaluations it took no part in: under global normalization, a checkpoint that lost its only head-to-head could be ranked above the opponent that beat it, and adding one checkpoint shifted the relative standing of others by up to 27 Elo.

Uncertainty from scenes, not trials. All trials of a task re-stage the same arrangement, so they are not independent. Confidence intervals therefore come from a cluster bootstrap (Davison and Hinkley, 1997) that resamples tasks with replacement, re-derives the weights of Eq. (7) inside every resample, and refits Eq. (4). We use 500 resamples and report the 2.5th and 97.5th percentiles (figures draw the 5th and 95th, i.e. 90% intervals, for legibility), withholding an interval when a checkpoint's matches span fewer than three tasks. Re-deriving the weights matters: computing them once, before resampling, made intervals about 30% narrower than the estimator warrants on our July evaluations.

Comparing two checkpoints. Overlapping intervals do not imply that two checkpoints are equal (Schenker and Gentleman, 2001): a lucky draw of tasks lifts both, and that shared variation cancels in their difference. We therefore compare checkpoints with the paired statistic

$$P(A > B) = \frac{1}{B} \sum_{b=1}^{B} \mathbb{1}\left[\mathrm{Elo}_A^{(b)} > \mathrm{Elo}_B^{(b)}\right], \tag{8}$$

computed over the same $B = 500$ resamples, together with the percentile interval of the gap $\mathrm{Elo}_A^{(b)} - \mathrm{Elo}_B^{(b)}$. On the current suite, the median interval is 93 Elo wide, and 176 of the 253 pairs among 23 checkpoints (the 20 in Fig. 36 and three further development checkpoints) are resolved at the 95% level.

Anchoring. Every new checkpoint's first A/B campaign is against the *reference* checkpoint, which ties it to the rest of the leaderboard and gives it full task coverage in one session. A second campaign against its warm-start parent answers whether a change helped. The reference is the best-rated checkpoint on the A/B leaderboard unless one is pinned deliberately. Because the paired comparison benefits from shared variation, checkpoints that must be told apart are best run in the same campaign.

Table 13: Paired comparisons on YAM V2. Elo difference (A minus B) with its paired 95% interval, and the share of 500 bootstrap resamples in which A rates higher (Eq. (8)). "GR00T 2" is the released checkpoint, and the expected win rate follows from the Elo difference.

A	B	ΔElo	95% interval	$P(A>B)$	expected win
GR00T 2	$\pi_{0.5}$ (Physical Intelligence et al., 2025)	$+236$	$[+171, +325]$	> 0.99	80%
GR00T 2	GR00T 1.7 (NVIDIA, 2026a)	$+95$	$[+50, +148]$	> 0.99	63%
GR00T 2	GR00T 2 Turbo (early checkpoint)	$+12$	$[-20, +48]$	0.80	52%
GR00T 2	GR00T 2 Turbo ($\star$)	-14	$[-58, +28]$	0.28	48%
GR00T 1.7	$\pi_{0.5}$	$+141$	$[+78, +227]$	> 0.99	69%
GR00T 2 Turbo	GR00T 1.7	$+109$	$[+54, +168]$	> 0.99	65%

Reproducibility. The rating is a deterministic function of the set of graded trials. Re-fitting the frozen snapshot offline with the same estimator reproduces every published rating and interval bound to within 0.1 Elo.

D.5. A/B Evaluation Infrastructure

The comparisons in this appendix are only as good as the blinding and attribution behind them. Both are handled by a station-side A/B client, and the arena's ingest is built to refuse anything it cannot attribute.

One session, several policies. The A/B client resolves every checkpoint in a campaign to a policy kind (GR00T 2, GR00T 1.7, $\pi_{0.5}$, and VLM-based policies) and starts a dedicated policy server for each arm. Every switch between arms checks that the server reports the expected kind, so a misrouted port fails loudly rather than silently serving the wrong model. Baselines run with their native settings, such as their own prompts and action filtering, so that a comparison measures the policy rather than our serving of it.

Blinding. The operator sees only what is needed to run and grade the rollouts. The visualization shows the same information for every arm. The operator's terminal shows nothing that names or times an arm: the client's command line, which names every checkpoint, is kept away from the operator, per-query timing warnings are recorded rather than printed, and warm-up messages and recording timers behave identically for every arm. The mapping from rotation slots to checkpoints is written to a session manifest that is read only after grading.

Recording and grading. Every episode is recorded and uploaded with per-arm latency statistics, which supply the inference-speed axis of Fig. 38. Each trial is graded with its rubric and an explicit ranking of the arms. The arena decodes a trial only when the request declares that its labels are rotation slots. Rows it cannot resolve to checkpoints are skipped rather than guessed, and a weekly audit compares each scored trial with the checkpoint its recording says it served.

D.6. Results: What the Arena Shows about GR00T 2

The leaderboard. Fig. 36 shows the current suite. The top four places are held by GR00T 2 and GR00T 2 Turbo checkpoints, whose intervals overlap. The GR00T 1.7 checkpoint sits about 80 to 110 Elo below them, and $\pi_{0.5}$, adapted to our robot, ranks near the bottom.

Head-to-head. Tab. 13 gives the paired comparisons. GR00T 2 is ahead of $\pi_{0.5}$ by 236 Elo and of GR00T 1.7 by 95 Elo, and both differences hold in every bootstrap resample. GR00T 2 Turbo is statistically indistinguishable from GR00T 2. The same ordering appears in the raw rubric scores of the trials in which both checkpoints ran (Tab. 14): GR00T 2's advantage over $\pi_{0.5}$ is present on every sub-suite and largest on the medium-to-hard motion primitives (0.48 vs. 0.14), and its advantage over GR00T 1.7 is concentrated there as well (0.33 vs. 0.14), with the two models level on language following and easy motions.

The climb. Fig. 37 places every checkpoint against its suite's reference by the date it was saved. On the current suite, the best checkpoint rose from 206 Elo below today's best in June to 93 and then 43 below in September, and to 14 below in mid-October (GR00T 2), before the GR00T 2 Turbo checkpoint that now leads. On the

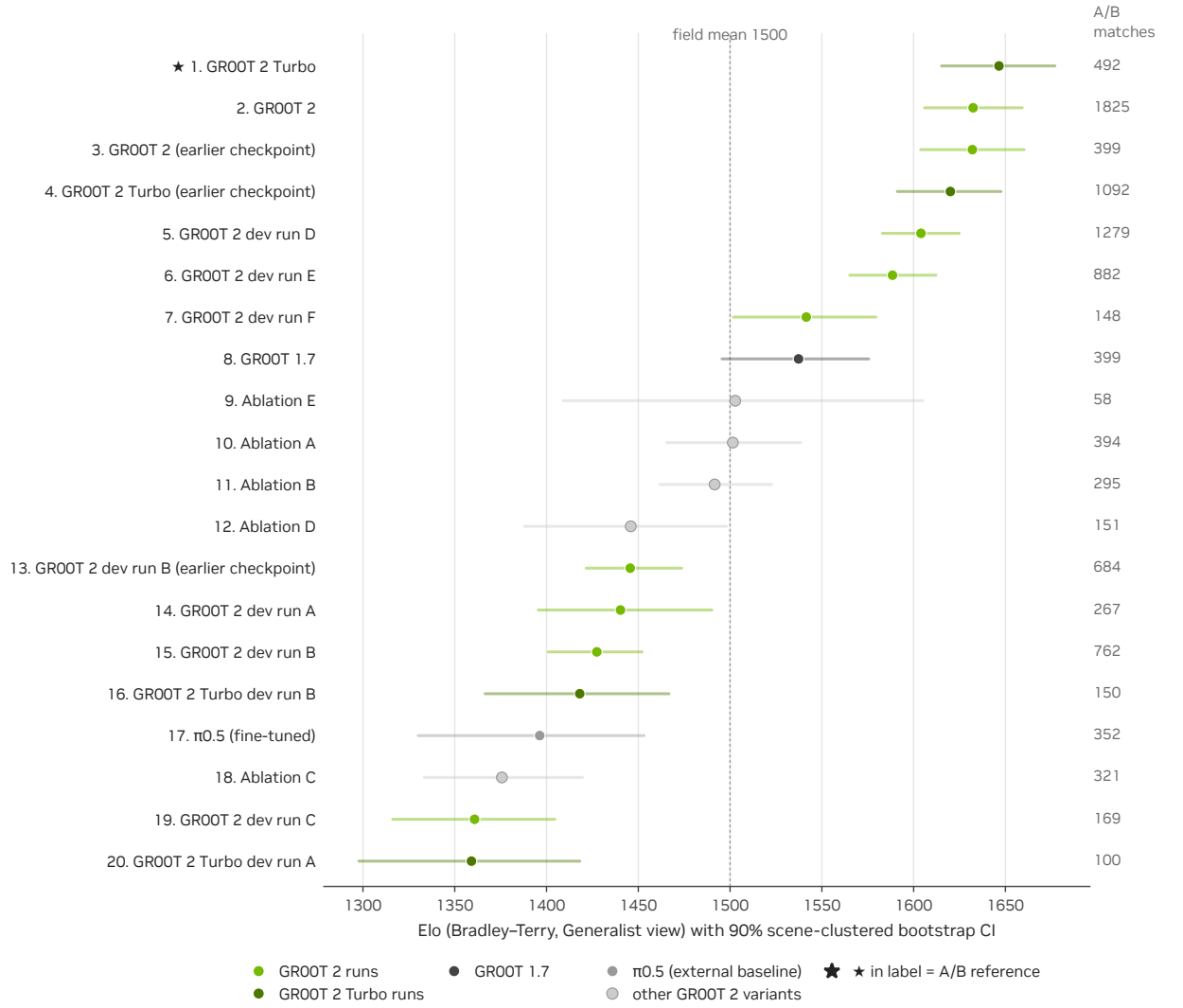


Figure 36: YAM V2 leaderboard. Elo with 90% scene-clustered bootstrap intervals for 20 checkpoints, colored by model family. Numbers on the right are A/B matches. ⋆ marks the A/B reference. Development checkpoints are shown under anonymized aliases.

first suite, the best checkpoint rose from 185 below to 87 below between June and mid-September before the current leader. Because all points come from one fit, these gaps are measured on a common scale even though the checkpoints were evaluated months apart, which is exactly what absolute scores from different sessions cannot provide (Sec. D.1).

Skill and speed. Fig. 38 adds the inference cost measured during the same A/B episodes. Among the 21 checkpoints served by the GR00T 2 policy server, the GR00T 2 Turbo reference is both the highest rated and the fastest, with no detectable loss in skill (Tab. 13). On a GeForce RTX 5090 workstation it runs about three times faster than GR00T 2 (52 vs. 171 ms per served call, Fig. 23). The timings in Fig. 38 are server compute recorded in the A/B serving logs. They rank checkpoints within one serving stack, but serving stacks differ in model size, hardware, and compilation, so the figure is not a like-for-like speed comparison across stacks.[3] These serving-log timings are also a different measurement from the median model-prediction latencies on named GPUs in Tab. 27 and from the per-call latencies of the YAM post-training models in Fig. 23.

[3]The serving stack is inferred from the model family, since each family runs its own policy server. The per-episode latency records do not include the GPU, host, or compilation settings.

Table 14: Matched rubric scores by sub-suite. Mean rubric score (0 to 1) over the trials in which both checkpoints were graded in the same campaign. Trial counts in parentheses.

Comparison	LF	MP easy	MP mid/hard	All
GR00T 2 vs. $\pi_{0.5}$	0.53 vs. 0.25 (149)	0.43 vs. 0.32 (77)	0.48 vs. 0.14 (43)	0.49 vs. 0.25 (269)
GR00T 2 vs. GR00T 1.7	0.48 vs. 0.46 (117)	0.51 vs. 0.51 (67)	0.33 vs. 0.14 (71)	0.45 vs. 0.38 (255)
GR00T 2 vs. GR00T 2 Turbo	0.43 vs. 0.48 (112)	0.55 vs. 0.66 (80)	0.44 vs. 0.34 (84)	0.47 vs. 0.49 (276)

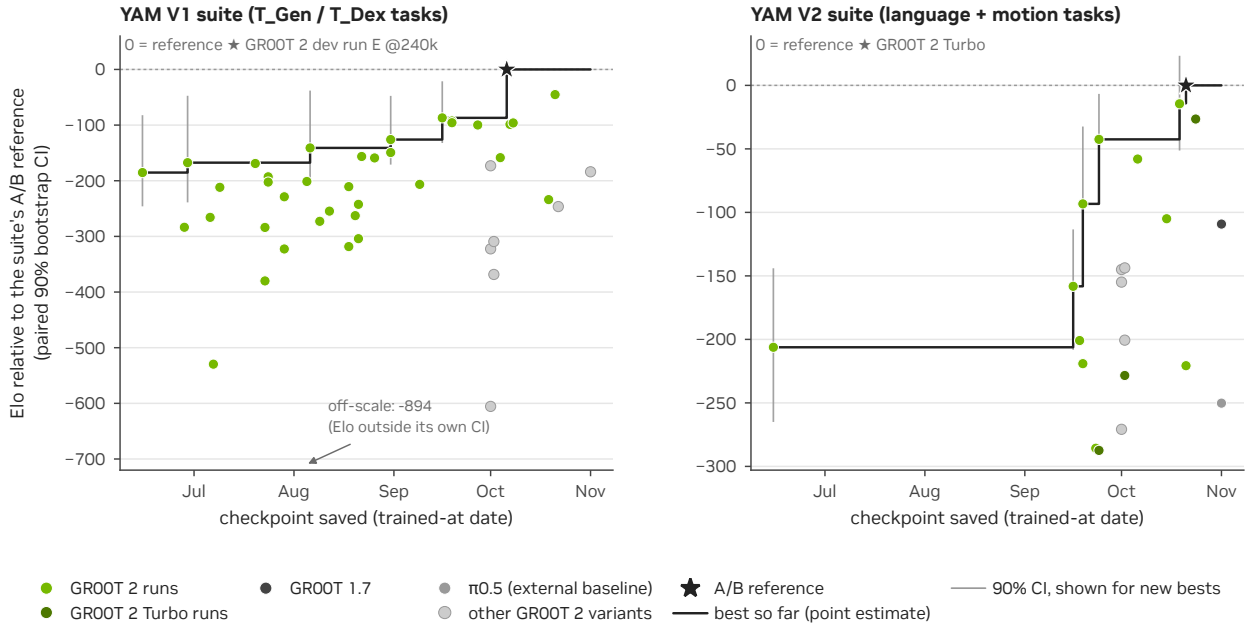


Figure 37: Climbing the hill. Every checkpoint on each suite, rated relative to that suite's A/B reference ($\star$, the top-rated checkpoint) and placed at the date the checkpoint was saved. The line tracks the best checkpoint so far. Bars are paired 90% bootstrap intervals, shown for the checkpoints that set a new best. All points in a panel come from one fit on the same evidence. The two suites are separate leagues. One V1 checkpoint lies below the plotted range.

How the arena was used. In the final months of the project, the arena was read weekly at the team's training sync, as the shared reference for which checkpoints had moved the frontier and which differences were still within noise. It separated effects that absolute scores conflated: on the dexterous suite, serving the same checkpoints with a new inference regime reordered the leaderboard (the previous leader fell from first of 13 to fifth of 9), which is why inference settings are part of a checkpoint's identity there (Sec. D.3). It also provided the real-world ground truth for calibrating simulation: the Bradley-Terry ratings from blinded A/B trials are the real-world side of the simulation-reality correlation in Sec. 4.5.2 ($r = 0.93$).

D.7. Limitations

- **Relative, suite-specific ratings.** Elo measures skill relative to the other checkpoints on the same suite. Ratings from different suites are not comparable, and adding a checkpoint shifts all ratings of a suite by a common constant.
- **Few independent units.** Independent replication comes from distinct tasks, not trials: the first suite has 16 tasks with A/B comparisons, the current one 38. Intervals are correspondingly wide (median 93 Elo on V2, 128 on V1), and some checkpoints with few tasks have none. On V1, one checkpoint's point estimate (Elo 840) lies outside its own bootstrap interval [853, 995]. We make no claims about it.

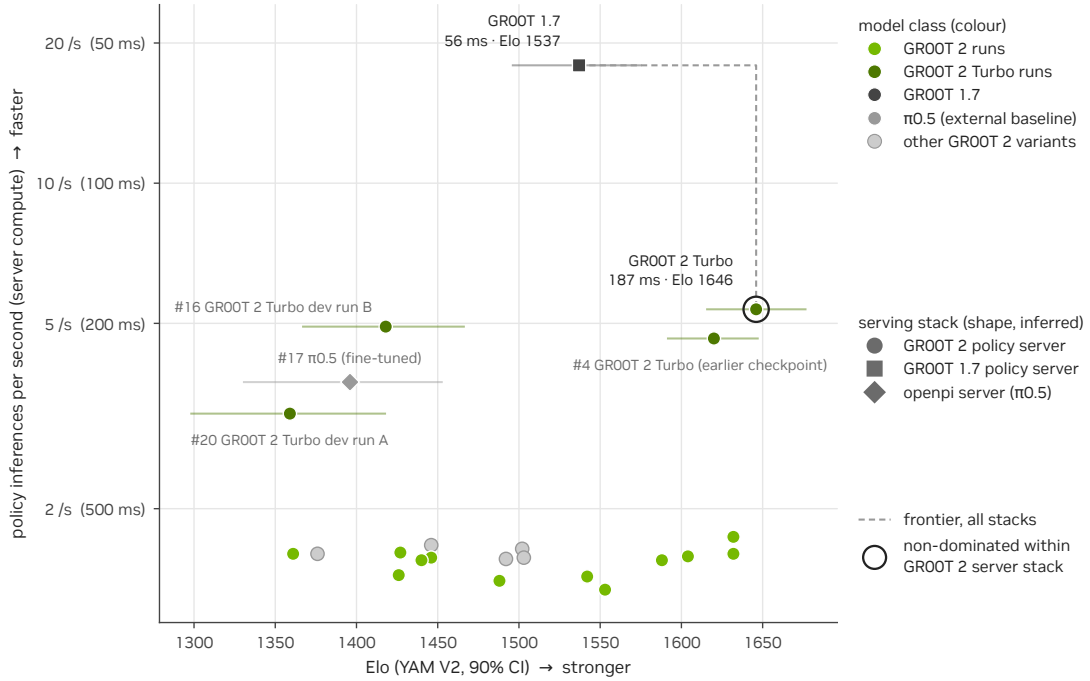


Figure 38: Skill versus inference speed on YAM V2. Elo against policy inferences per second (log scale), measured as server-side compute per inference during the same A/B episodes. Bars are 90% Elo intervals, shown for the labelled checkpoints (all intervals are in Fig. 36). Inference-rate intervals are narrower than the markers. Colour is the model family and marker shape the serving stack. Dashed: frontier of point estimates across stacks. Ringed: the checkpoint that dominates all others served by the GR00T 2 policy server.

- **Ties and grading.** Ties are 30 to 35% of matches. Rankings come from human operators, and the rubric bounds but does not remove their judgment.
- **Confounds across campaigns.** A/B trials remove drift within a campaign. Differences between stations and operators across campaigns are absorbed only through shared opponents, and the reference-first pairing policy keeps most checkpoints connected to the same opponents.
- **Live evidence.** Graded trials of campaigns still in progress count immediately. Four V2 campaigns were paused at the snapshot, and the comparisons of GR00T 1.7 and $\pi_{0.5}$ rest on campaigns from the last week before it.
- **Curated metadata.** Model-family labels, checkpoint dates (storage timestamps), and serving stacks are recorded or inferred outside the A/B record itself. Embodiment tags and serving settings, such as $\pi_{0.5}$'s separate embodiment tag or video-codec settings, are admission criteria but are not part of the rating.

E. UMIverse for GR00T 2

UMIverse is our family of robot-free interfaces that record human demonstrations in a robot's own sensory and action space. *Gripper UMI* is a handheld parallel gripper in the lineage of the Universal Manipulation Interface (Chi et al., 2024): it carries the robot's wrist cameras and measures the gripper width that the robot later commands. *DexUMI* (Xu et al., 2025) is a wearable hand exoskeleton that replicates the kinematics, contact surfaces, and camera mount of a dexterous robot hand, so that every recorded finger joint, fingertip force, and camera view has a robot counterpart. Our GEAR DexUMI (NVIDIA GEAR Team, 2026) replicates the XHAND1 Pro-U (Sec. E.4). Unlike teleoperation, neither interface needs a robot during collection. Unlike human video, both record actions in robot coordinates and observations through robot cameras.

The two interfaces play complementary roles in GR00T 2. Gripper UMI provides scale: collection stations in three regions supply thousands of hours of bimanual pretraining demonstrations, and the resulting policies deploy directly onto a matching robot (Sec. E.2) or transfer across an embodiment gap to YAM (Sec. B.2), a test of the cross-embodiment imagination that GR00T 2 is built for. DexUMI provides depth: post-training data for complex, long-horizon, and dynamic dexterous tasks (Sec. B.3), which GR00T 2 learns with compliance labels derived from fingertip force (Eqs. (1) and (3)). For DexUMI, the visual gap between the exoskeleton and the robot hand is addressed inside the world-action model rather than by editing video (Sec. E.6). DexUMI demonstrations can also be retargeted to a hand that the exoskeleton does not replicate, such as the Sharpa hand (Sec. B.4). The UMI $\rightarrow$ YAM transfer and the DexUMI post-training results are presented in Sec. B.2 and B.3. This appendix describes the hardware of both interfaces (Sec. E.1 and E.4), UMI direct (Sec. E.2), the embodiment gaps and ablations of the UMI $\rightarrow$ YAM transfer (Sec. E.3), and DexUMI's post-training recipe and WAM-native inpainting (Sec. E.5 and E.6).

E.1. Gripper UMI Hardware

The handheld gripper mirrors its robot counterpart. Relative to the original UMI gripper (Chi et al., 2024), the main mechanical change is that the pin joints, bearings, and return springs of the finger-trigger linkage are replaced by a single compliant *flexure linkage* (Fig. 39) that is printed entirely on a desktop printer. This yields the following improvements over the previous design:

- **Constant force scaling.** The finger-to-jaw force scaling is constant over the gripper's entire range of motion. In the original UMI gripper, the scaling is large when the gripper is fully open but varies nonlinearly with opening, and when the gripper is fully closed the fingertip force falls to roughly 20% of the force applied at the trigger. The constant scaling of the new design instead delivers substantially higher fingertip force at small openings, which is better suited to fine-grained dexterous manipulation.
- **Fewer parts and faster assembly.** The flexure joints are printed directly into the links, so the linkage needs no pins, bearings, or return springs and only a few fasteners, which greatly reduces part count and assembly time.
- **Simpler linkage and less backlash.** The complex multi-part linkage is replaced by a single compliant mechanism with no discrete sliding or pivoting interfaces, which significantly reduces backlash.
- **Encoder-based width sensing.** An encoder measures the gripper opening directly, which avoids the visual occlusion that can affect a camera-and-marker approach to width sensing.

Flexure linkage. The linkage is mirror-symmetric and rides on a 150 mm linear rail (MGN9). Rigid links and finger mounts are printed in PLA, and four TPU 90A flexure joints are printed directly into them. Each joint is a short, thin strip with a T-shaped head embedded in a pocket of the PLA part, so load transfers without additional fasteners. The operator's finger pinch acts on the long links, and the bottom joint is guided along the symmetry plane, so pinch maps to jaw motion through a fixed lever ratio: the jaws move about $2.9\times$ as far as the fingertips and, ideally, carry about $0.35\times$ the finger force. Because the rigid links meet the flexures directly, the operator's input reaches the jaws through a short, stiff load path with little lost motion.

Design rules from rapid prototyping.

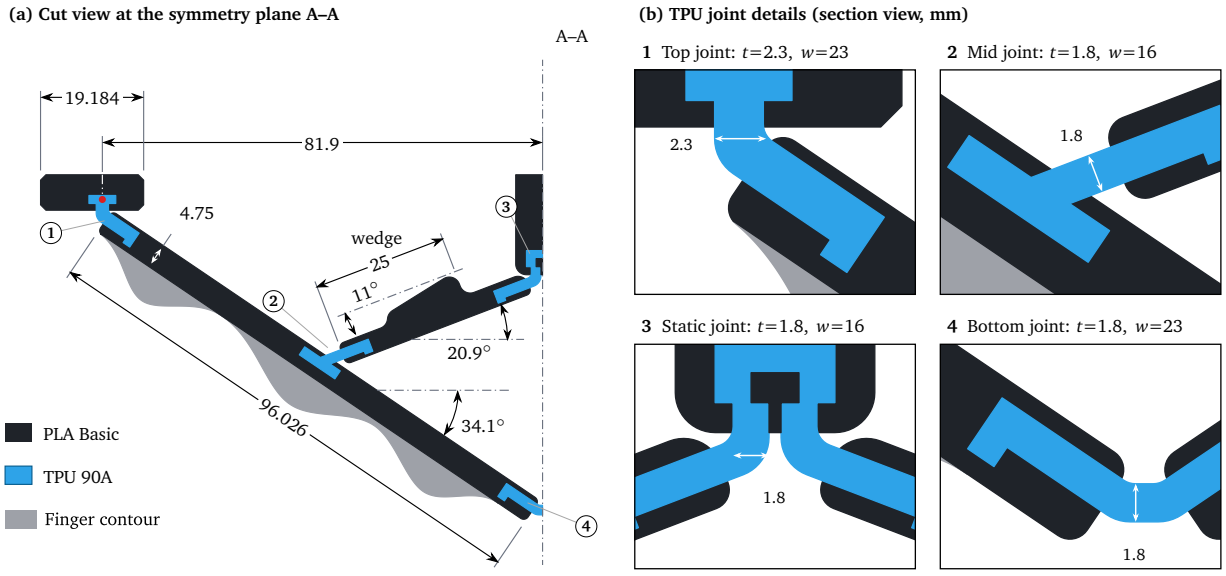


Figure 39: Flexure linkage of the handheld gripper. (a) Cut view at the symmetry plane with main dimensions in mm: black parts are PLA, blue parts are TPU 90A, the gray shape is the operator's finger contour, and the four TPU joints are numbered 1 to 4 (top, mid, static, bottom). (b) Section views of joints 1 to 4: T-shaped TPU anchors sit in pockets of the PLA parts, and the short exposed strip between them forms the flexure (strip thickness t and out-of-plane width w in mm).

- **Thin strips:** strip thickness is set by print accuracy, 1.8 mm (2.3 mm at the top joint), about four extrusion lines, so it prints reliably yet still bends.
- **Short necks:** each exposed neck is kept short so the flexure behaves like a rotary joint, with bending concentrated at one location instead of along a long leaf spring.
- **Width:** links and joints are 23 mm wide out of plane (16 mm for the short link and inner joints). Narrower variants twisted under side loads, while wider ones needed a firm grip to close.
- **Compliance:** the finite compliance of TPU absorbs small mismatches between rail, mounts, and link lengths that a rigid pin-jointed version would not tolerate.

Pre-strained open state. The linkage is printed wider than the rail allows (163.8 mm relaxed pivot span versus 118.1 mm at the rail stops), so the TPU joints are pre-strained at the open stop (Fig. 40). This provides a passive restoring force that returns the gripper to its open state and replaces separate return springs.

End effector. Apart from the added width encoder, sensor placement is kept consistent with the existing UMI configuration, which preserves compatibility with established data-collection pipelines. Further end-effector details, including sensing and assembly, are available in our open-source documentation (NVIDIA GEAR Team, 2026).

E.2. UMI Direct

UMI direct means deploying a UMI-trained policy on the corresponding UMI robot gripper, the robot counterpart of the handheld: the same sensors (the same camera board with its fisheye wrist camera and trigger camera), the same gripper kinematics (two soft-fingered carriages driven symmetrically along a rail), and the same gripper fingers. Observations and actions therefore carry over without translation: the policy sees the scene through the cameras it was trained on and commands the finger width the handheld measured, so the camera and gripper gaps are removed by construction (Sec. E.1). The robot side exchanges poses in the handheld's wrist frame, which a per-robot hand-eye calibration maps onto the robot, and gripper widths in millimetres, the quantity the handheld measured. A policy trained on grasp-point poses and normalized widths (Sec. B.2) is wrapped so that its outputs are converted back to these quantities. On the deployment robot, the policy

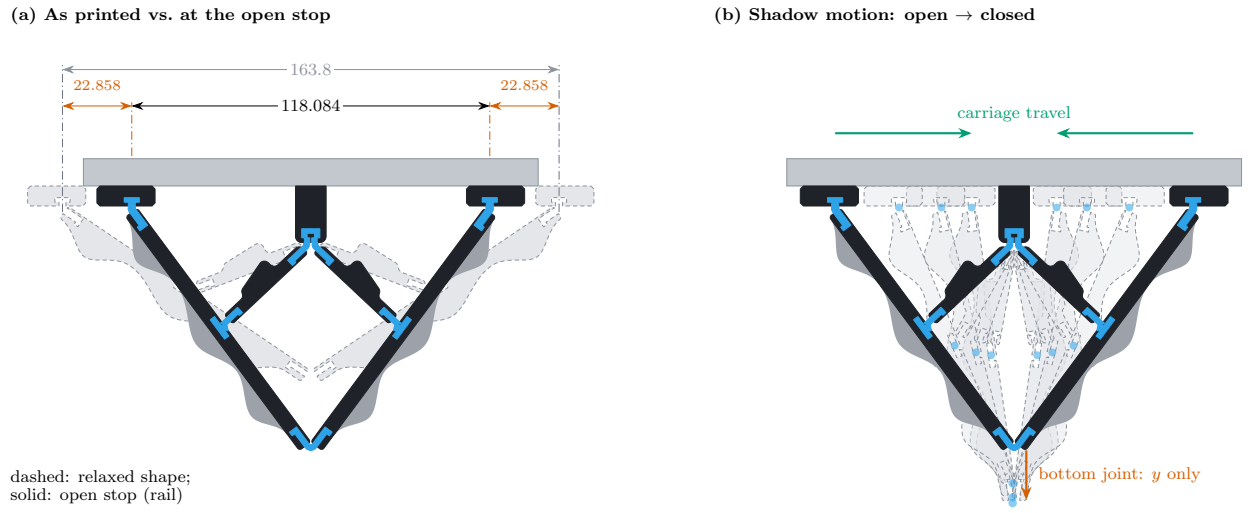


Figure 40: Pre-strain and motion of the flexure linkage. (a) Relaxed (printed) shape (dashed) and its pose at the open stop (solid). The 22.858 mm difference per side pre-strains the TPU joints. (b) Shadows of the linkage closing from the open stop to the closed pose: the finger mounts slide inward along the rail and the bottom joint moves along the symmetry plane.

predicts 48-step action chunks, of which 30 are executed, at 30 Hz inside a 60 Hz control loop. Fig. 41 shows rollouts of UMI-direct policies on four assembly tasks.

Training. We post-train GR00T 2 on the Gripper UMI demonstrations of each target task alone. UMI uses the grasp-point representation on the action-head row it shares with YAM (the shared-head design of Sec. B.2), even though no YAM data enter these mixtures, and it observes four views: the head camera, the two fisheye wrist views re-projected stereographically to 320×240 as in pretraining, and the bottom camera.

Ablation: UMI-10K in pretraining. To test whether UMI-10K data in pretraining help UMI direct, we post-train the same UMI-only recipe from two GR00T 2 pretraining checkpoints (Tab. 15): the default checkpoint, whose pretraining mixture includes UMI-10K, and a checkpoint pretrained without any UMI-10K data, the same GR00T 2 w/o UMI pretraining baseline as in Tab. 4. This checkpoint comes from an earlier pretraining run, with fewer pretraining samples, a different robot mixture, and small model differences, so part of the gap may come from these differences rather than from UMI data alone. It is not the GR00T 2 w/o cross-embodiment pretraining baseline of Sec. B.1.5, which is pretrained on target-robot data alone. The difference is large and consistent. With UMI-10K in pretraining, the four assembly tasks reach 30%, 50%, 10%, and 50% end-to-end success (average 35%) with progress scores of 79 to 94. Without it, the same post-training yields 0%, 10%, 0%, and 0% (average 2.5%) with progress scores of 60 to 75. The gap in progress is smaller than the gap in success: the policy without UMI pretraining still performs most coarse steps but fails the precision steps, screw insertion and part seating, that an assembly must chain to completion. We read this as the handheld observation domain being in-distribution for the pretrained backbone, so that post-training can spend its few hours of demonstrations on the task rather than on learning to see a handheld gripper through fisheye cameras. One caveat bounds the claim: because UMI-direct post-training places UMI on the action-head row it shares with YAM rather than on the row it was pretrained on, any benefit reaches the policy through the shared transformer, not through the pretrained action head.

E.3. UMI → YAM Transfer: Embodiment Gaps and Ablations

As described in Sec. 3.2 and Sec. B.2, GR00T 2 can be adapted to a bimanual gripper robot from handheld Gripper UMI demonstrations alone, with no teleoperation of the target robot. In this section, we present the details behind that result: how co-training lets the model absorb the appearance difference on its own (Fig. 14),



Figure 41: UMI direct rollouts. Each row shows one task with its instruction and six keyframes of a GR00T 2 policy post-trained on Gripper UMI demonstrations alone and deployed on the UMI robot gripper, numbered by stage: Toy Dumptruck Assembly, Toy Plane Assembly, Bluey Car Assembly, and Monster Truck Assembly. Each task requires placing parts, driving screws, and reorienting the body of the toy.

Table 15: UMI direct. Prog. is the average substage score and SR the end-to-end success rate on the UMI robot gripper, both on a 0 to 100 scale, of GR00T 2 post-trained on Gripper UMI demonstrations alone. GR00T 2 w/o UMI pretraining starts from an earlier pretraining checkpoint whose mixture contains no UMI-10K data, while GR00T 2 starts from the default checkpoint, whose pretraining mixture includes UMI-10K. Each task uses 10 trials, and the last row averages the four tasks. Bold marks the best Prog. and SR in each row.

		GR00T 2 w/o UMI pretraining		**GR00T 2**	
Task	UMI hours	Prog.	SR	Prog.	SR
Toy Dumptruck Assembly	5.07	67.0	0.0	**89.0**	**30.0**
Toy Plane Assembly	4.93	75.0	10.0	**85.0**	**50.0**
Bluey Car Assembly	3.70	67.0	0.0	**79.0**	**10.0**
Monster Truck Assembly	2.96	60.0	0.0	**94.0**	**50.0**
Average		67.3	2.5	**86.8**	**35.0**

the three gaps that remain in sensing, actuation, and environment and the quantities that define them (Tab. 16), and four ablations that isolate the design choices on which the transfer depends (Tab. 17).

E.3.1. Setup

The UMI head, wrist, and bottom cameras fill YAM's top, wrist, and bottom view slots (Fig. 42). The benchmark comprises Toy Dumptruck Assembly, NIST Board Disassembly, and Conveyor Belt Sorting and Tossing (Fig. 29 and Tab. 4). The UMI cohorts comprise 75 episodes for Toy Dumptruck Assembly, 74 for NIST Board Disassembly, and 41 for Conveyor Belt Sorting and Tossing. None of the YAM co-training sources contains a demonstration of any benchmark task. As in UMI direct, the policy predicts 48-step action chunks, of which 30 are executed.

E.3.2. What Co-Training Absorbs

The model imagines the robot. The UMI demonstrations show a human's arms and the blue-fingered handheld gripper, whereas the deployed policy sees YAM's own arms and gripper. GR00T 2 bridges this appearance difference natively rather than by editing the training videos. Because it predicts future frames jointly with actions, the task-agnostic YAM data teach it what YAM looks like and the UMI data teach it what the task

Table 16: Gripper UMI versus YAM. YAM differs from the handheld gripper in its cameras, its gripper interface, and its control frame.

	Handheld UMI	**YAM**
Wrist camera	Fisheye, $\approx 180°$ (OAK), 640×400	Standard FOV, $78 \times 63°$ (RealSense D405), 640×480
Other views	Head camera, wide FOV (OAK), trigger camera	Top and bottom cameras, standard FOV (RealSense)
Gripper	Trigger, gear train, soft fingers	Built-in gripper
Full opening	85.55 mm	96 mm
Gripper signal	Width (mm), encoder	Normalized $[0, 1]$
Control frame	Wrist frame at gripper motor center	Grasp site, 0.1347 m from the flange
Action label	Achieved pose and width	Leader command, follower lags $\approx$4 frames

looks like, and at deployment the two combine in the model's imagined future. Fig. 14 shows this directly: conditioned on the same YAM top-camera frame, a model post-trained on UMI data alone predicts human arms and UMI fingers entering the scene, whereas the co-trained model predicts YAM's arms performing the assembly. The co-training data therefore supply more than action statistics. They give the world model the robot's body, so that a task seen only in human hands is imagined, and then executed, on the robot.

E.3.3. What Remains: Three Embodiment Gaps

Co-training absorbs the visual appearance of the robot, but it does not by itself close the embodiment gap. Three gaps remain, in sensing, actuation, and environment, and each requires a deliberate design choice. We quantify them in turn (Tab. 16) and then ablate the choices that close them.

- *Sensing gap.* The two embodiments observe the scene through cameras with substantially different fields of view (Tab. 16 and Fig. 42). UMI pairs a wide-FOV head camera with fisheye wrist cameras of about $180°$ (OAK), whereas YAM uses a standard-FOV top camera and pinhole wrist cameras of $78 \times 63°$ (RealSense D405). The two wrist cameras also differ in orientation and in where they sit relative to the fingers. As a result, even the stereographic re-projection of the wrist fisheye used in pretraining places the scene 74 to 94 px from where YAM's camera shows it at 320×240 (Fig. 43). The lens centers are also 4.9 cm apart, so a re-projection can align the scene but not the fingers, which stay smaller in the UMI view than in YAM's. Beyond geometry, the UMI frames show the operator's arms and the handheld gripper, whereas YAM's frames show its own arms and gripper (Fig. 42).
- *Actuation gap.* Beyond sensing, the two embodiments differ in how actions relate to states and in the space in which actions are expressed (Tab. 16). First is action versus state. A UMI demonstration only records the pose the handheld gripper reached, so action and state coincide and the data assume zero tracking error. On YAM, the action is the command of a teleoperation leader arm and the state is where the follower arrives, and the two differ because the follower is a low-stiffness position-controlled arm (brushless motors with 40:1 shoulder and 10:1 wrist reductions under a PD controller with joint stiffness from 80 down to 10 N m/rad). It lags the leader by about four frames (133 ms at 30 Hz) and settles at a load-dependent offset from the command. YAM data thus teach the policy to command ahead of, and offset from, the observed state, whereas UMI data teach it to command the state itself. Second is the control space. UMI acts in end-effector space, with gripper widths as labels. YAM's action row holds the end-effector pose and width plus the six joint positions of each arm, 32 dimensions in all. Handheld data supply only the 20 end-effector dimensions, and the 12 joint dimensions have no counterpart. Even the shared dimensions use different conventions: UMI poses refer to a wrist frame at the gripper motor center

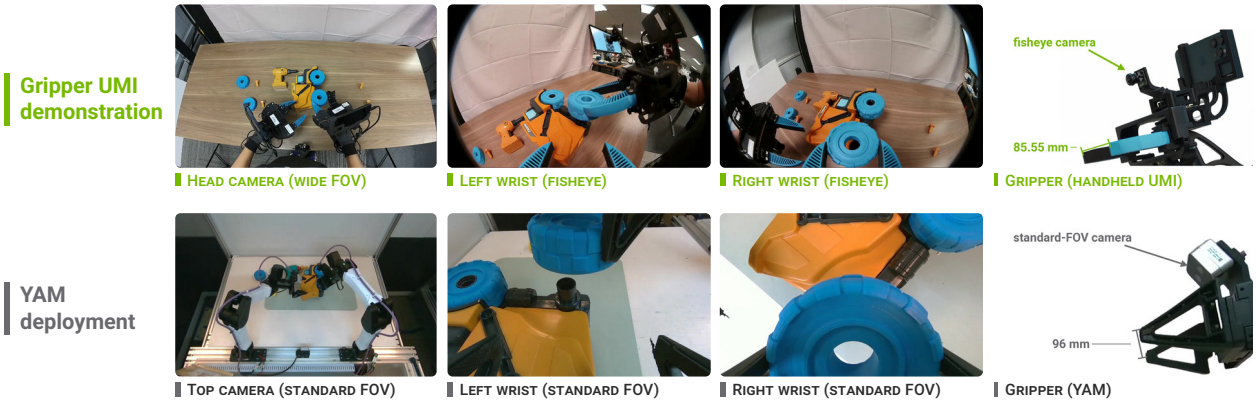


Figure 42: Gripper UMI versus YAM, same task, same camera slots. Top: a Gripper UMI demonstration of Toy Dumptruck Assembly seen through its wide-FOV head camera and two fisheye wrist cameras, recorded in the wild. Bottom: the YAM robot performing the task in its cell, seen through the standard-FOV top camera and two wrist RealSense cameras that fill the same observation slots. Right: the handheld UMI gripper and the YAM gripper, each with its wrist camera. The lines mark the full opening between the fingertips (85.55 mm and 96 mm). Both frames show the same step: the left gripper holds the truck while the right gripper brings the second wheel to the axle.

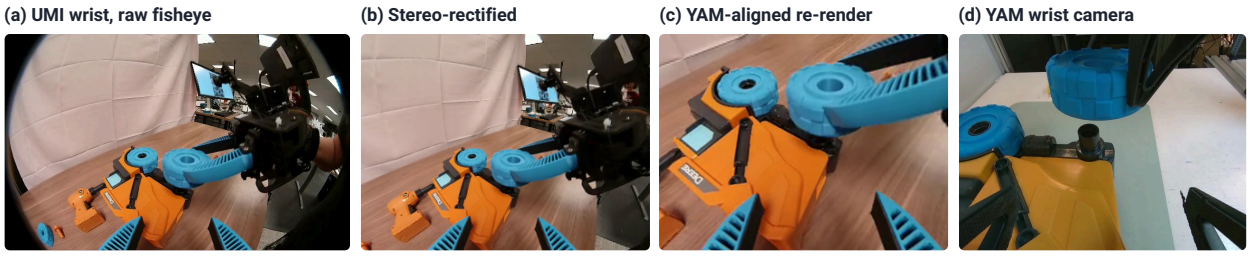


Figure 43: Wrist views for UMI → YAM transfer. The same UMI wrist frame as (a) the raw fisheye, (b) the stereographic view used in pretraining, and (c) the YAM-aligned re-render that imitates YAM's wrist camera. (d) The real YAM wrist camera at the same step.

and YAM poses to a grasp site 0.1347 m from the flange, and UMI records the width in millimetres against an 85.55 mm opening whereas YAM takes a normalized $[0, 1]$ command against 96 mm. In training, we move the UMI wrist frame 0.1347 m along its approach axis to the grasp site and normalize the widths to match YAM.

- *Environment gap.* Finally, the UMI demonstrations are recorded in the wild, whereas the policy is evaluated in a YAM cell whose workspace, lighting, and background never appear in the demonstrations (Fig. 42). The YAM co-training data show the cell but contain no demonstration of the target tasks, so they provide observations of the evaluation scene but no evidence of the target tasks.

E.3.4. Ablations

The default recipe uses a shared action head, YAM-aligned wrist views, all UMI episodes, and a pretrained checkpoint that includes UMI-10K data. Four ablations each change one of these choices and report the result in Tab. 17: whether UMI shares YAM's action head (actuation gap), whether the UMI wrist views are aligned to YAM's camera (sensing gap), how many UMI demonstrations are needed, and whether UMI data in pretraining matter.

Table 17: UMI → YAM ablations. Prog. is the average substage score and SR the end-to-end success rate on YAM, both on a 0 to 100 scale, as in Tab. 4. The shaded row in each block is the default configuration (shared action head, YAM-aligned wrist-view warping, all UMI episodes, pretraining with UMI-10K data), and the other rows change that one choice. For Conveyor Belt Sorting and Tossing, SR is the share of 50 items sorted, and there is no separate progress score. Bold marks the best value in each block.

		Toy Dumptruck Assembly		NIST Board Disassembly		Conveyor Belt Sorting and Tossing
Ablation	**Setting**	Prog.	SR	Prog.	SR	SR
Action head	Shared	**96.0**	**60.0**	86.0	50.0	**86.0**
	Separate	89.0	10.0	**93.0**	**60.0**	64.0
Wrist-view warping	YAM-aligned	**96.0**	**60.0**	**86.0**	**50.0**	**86.0**
	Random	86.0	0.0	78.0	0.0	60.0
	None	79.0	0.0	76.0	0.0	46.0
UMI episodes	Full	**96.0**	**60.0**	**86.0**	**50.0**	**86.0**
	Half	92.0	30.0	68.0	0.0	84.0
	Quarter	81.0	10.0	66.0	0.0	70.0
Pretraining	With UMI-10K	**96.0**	**60.0**	**86.0**	**50.0**	**86.0**
	Without	84.0	20.0	65.0	0.0	34.0

Ablation 1: shared versus separate action head. GR00T 2 shares its diffusion transformer across embodiments but keeps one row of action-encoder, state-encoder, and action-decoder weights per embodiment. The *shared-head* arm re-expresses UMI in YAM's semantics and places it on YAM's row: grasp-site poses, absolute widths normalized by the 85.55 mm full opening, the first 20 of YAM's 32 action dimensions, and YAM's normalization statistics. The *separate-head* arm keeps UMI on the row it was pretrained on, with wrist-frame relative poses, relative widths in millimetres, and its own statistics. The separate arm therefore changes the action representation as well as the head, so the comparison is not a single-variable change. The two arms trade embodiment grounding against skill coverage, and the results in Tab. 17 split along that line. The shared head is clearly stronger on Toy Dumptruck Assembly (60% versus 10%) and Conveyor Belt Sorting and Tossing (86% versus 64%), whereas the separate head is slightly better on NIST Board Disassembly (60% versus 50%, progress 93 versus 86). The NIST margin is concentrated in the hex-nut subtask, which the separate head completes in 7 of 10 trials and the shared head in 5 of 10. This is also the subtask on which the demonstrations are most specific: the UMI demonstrators slide the nut off its post without rotating the wrist, and the YAM co-training data do not contain this way of removing a nut. With the shared head, UMI actions are decoded by the same weights as the YAM data, so the decoder's prior follows YAM's motion statistics, and a strategy that

appears only in the demonstrations is reproduced less reliably. The separate head keeps UMI on the row it was pretrained on and preserves the demonstrators' strategy. The same mechanism explains the reverse outcome on the other two tasks: where the demonstrated motions are ones the YAM data already perform, placing UMI on YAM's row grounds the task evidence directly in YAM's action statistics and controller, whereas the separate head must rely on the transformer to map YAM observations onto a row it has never seen paired with them. The shared head is therefore the better default, while the separate head helps only where the demonstrations rely on a motion strategy that the co-training data do not share.

Ablation 2: wrist-view alignment. We re-render the native UMI fisheye ($\approx 180°$) as a virtual standard-FOV pinhole camera at the same optical center that imitates YAM's $78 \times 63°$ wrist view (Tab. 16 and Fig. 43): a 202.96 px focal length at 320×240 ($76.5 \times 61.2°$), pitched about $27.1°$ down, with rotation and focal length fitted in Isaac Sim on 112 same-scene UMI/YAM render pairs. On held-out pairs, the scene error is 8 px for objects closer than 30 cm and 11 px beyond, versus 74 and 94 px for the stereographic view, and the grasp zone is resolved at 12.5 rather than 6.9 px/cm for the raw fisheye. Identical maps back-fill the training data and run at deployment. We stress that this alignment is approximate. It matches the field of view and the viewing direction of YAM's wrist camera, but not the lens, the stereo baseline, the offset of the optical center from the gripper, or the appearance of the arm and fingers in the frame, and the residual discrepancy is visible in Fig. 43. We compare three training views for the UMI wrist cameras: the stereographic view used in pretraining, the YAM-aligned view, and a random choice between the two on every training sample. Only the aligned view transfers on the two long-horizon tasks (Tab. 17). With the stereographic view, end-to-end success is 0% on both long-horizon tasks, and 46% of the items are sorted on Conveyor Belt Sorting and Tossing. The random mixture recovers 60% on tossing but still 0% on the long-horizon tasks, while the aligned view reaches 60%, 50%, and 86%. The progress scores tell a more specific story. They fall only moderately without alignment (96 to 79 on Toy Dumptruck Assembly, 86 to 76 on NIST Board Disassembly), so the policy still performs most steps. What it loses is the precision steps, in which the gripper must meet a screw hole or a nut within a few millimetres. A view mismatch moves the grasp zone in the image and changes its pixel scale (12.5 versus 6.9 px/cm), which corrupts exactly the image-to-action mapping those steps depend on, while leaving coarse transport intact. The random mixture helps only on tossing (60% versus 46%), where coarse transport suffices, and not on the precision steps, because it asks the policy to be invariant to a viewpoint change that is informative for those steps. The policy averages over two geometries and is sharp in neither. We draw two conclusions. First, the video model absorbs the appearance gap, since human arms and a handheld gripper in training versus robot arms in deployment do not prevent transfer, but it does not absorb a geometric gap. Second, the geometry need only be roughly matched: coarse agreement in field of view and viewing direction is sufficient to recover most of the transfer, which makes the method practical for new target robots whose wrist cameras are known only from a datasheet.

Ablation 3: amount of UMI data. The UMI demonstrations are the only source of the target task, so their number bounds how much task evidence post-training can draw on. We shrink each cohort to about half and a quarter of its episodes at a fixed 0.6 : 0.4 UMI : YAM sampling ratio (Tab. 17). The three tasks degrade very differently. Conveyor Belt Sorting and Tossing is nearly flat (86%, 84%, 70%): it is a short-horizon dynamic skill with one decisive event, and 20 episodes already cover it. Toy Dumptruck Assembly degrades gradually in end-to-end success (60%, 30%, 10%) while its progress score stays high (96, 92, 81), the signature of a long horizon in which small per-step losses compound multiplicatively. Notably, the policy trained on only 20 demonstrations still completes the full four-stage assembly end to end in 1 of 10 trials and 81% of its individual steps, on a robot that has never been teleoperated for the task. NIST Board Disassembly collapses at the first halving (50% to 0%), and the substage table shows why: the nut subtask goes from 5 of 10 to 0 of 10 while the other four subtasks barely move. The data requirement of a task is therefore set not by its length but by its hardest subskill, the one with the least support in both the pretraining and the co-training data. That subskill is the first to disappear as demonstrations are removed, and in a sequential task its loss zeroes the end-to-end rate. In practice, 41 to 75 handheld demonstrations sufficed for every task here, and half of that sufficed whenever

the task contained no such rare subskill.

Ablation 4: UMI-10K in pretraining. The main result (Tab. 4) already shows that starting from a checkpoint pretrained without UMI data reduces transfer success to 20% and 0% on the long-horizon tasks and the share of conveyor items sorted to 34%. We repeat the row in Tab. 17 and examine what pretraining contributes. Both models see the same post-training data and the same number of steps, so the difference lies entirely in what the pretrained weights already encode. The checkpoint without UMI also differs in pretraining length, robot mixture, and small model details (Sec. E.2), so not all of the gap need come from UMI data. We identify two contributions. First, with UMI-10 data in pretraining, UMI's observation domain, a handheld gripper and human arms seen through fisheye wrist views, is already in-distribution for the video backbone, so post-training does not have to learn to see the demonstrations. Second, and more important for transfer, pretraining on UMI alongside YAM and the other robot corpora aligns UMI's actions with the other embodiments inside the shared transformer: the model has already learned that a UMI grasp and a YAM grasp are the same event. Post-training then only needs to attach task evidence to an alignment that exists, whereas without UMI in pretraining the same 41 to 75 episodes must simultaneously teach the observation domain, bind it to YAM's action semantics, and teach the task. The few-shot budget is spent on the embodiment rather than on the task. The progress scores are consistent with this reading. Without UMI pretraining the policy still performs most coarse steps (84 and 65), so it does learn the task structure. It fails at the precision steps, where an imprecise embodiment alignment turns into a missed hole or a dropped part. UMI data in cross-embodiment pretraining are thus not a convenience but the mechanism that makes a few dozen demonstrations sufficient.

E.4. DexUMI Exoskeleton Hardware

The GEAR DexUMI exoskeleton (NVIDIA GEAR Team, 2026) replicates the kinematics of the target robot hand (XHAND1 Pro-U) joint for joint and reproduces its fingertip and palm contact surfaces, which keeps the visual and physical gap to the robot small for policy training. It is also designed for wearability, so that operators can comfortably collect high-quality data across long, dexterous tasks. Compared with the original DexUMI, it offers the following improvements:

- **More degrees of freedom.** The exoskeleton captures 16 degrees of freedom, using 20 joints of which four pairs are coupled.
- **Higher precision and durability.** Machined parts give higher precision and durability.
- **No over-constraint, better tracking and wearability.** Removing non-essential contact surfaces and adopting a remote-center-of-motion (RCM) linkage resolves the over-constraint of the original structure. The structure flexes less, which improves tracking precision and wearability.
- **Daisy-chained encoders.** The joint encoders are daisy-chained, which simplifies cable management.
- **Left and right hands.** Both left- and right-hand versions are available, enabling bimanual tasks.

Remote-center-of-motion joints. Matching the robot hand's abduction/adduction (AB/AD) and flexion joints is a packaging problem, since those joint centers lie where the operator's finger is. We replicate them with an RCM linkage that creates a virtual joint at the robot hand's joint center without placing hardware in the operator's workspace (Fig. 44a). The RCM is a parallel four-bar linkage. To avoid over-constraining it, only two of the four holes at the top joint are used. Leaving the other two unused reduces binding from manufacturing tolerances and keeps motion smooth.

Thumb. The thumb linkages use a "kicked-out" geometry that preserves the robot hand's thumb kinematics while clearing space for the operator's own thumb, so its range of motion is unobstructed during dexterous manipulation (Fig. 44b).

Rigid, compact joints. Every joint is supported on both sides to prevent deflection and improve measurement precision. Custom pins keep joint bulk low while forming secure connections, and a variant with a direct keying interface mates to the joint encoders, reducing backlash and improving sensor accuracy (Fig. 45a).

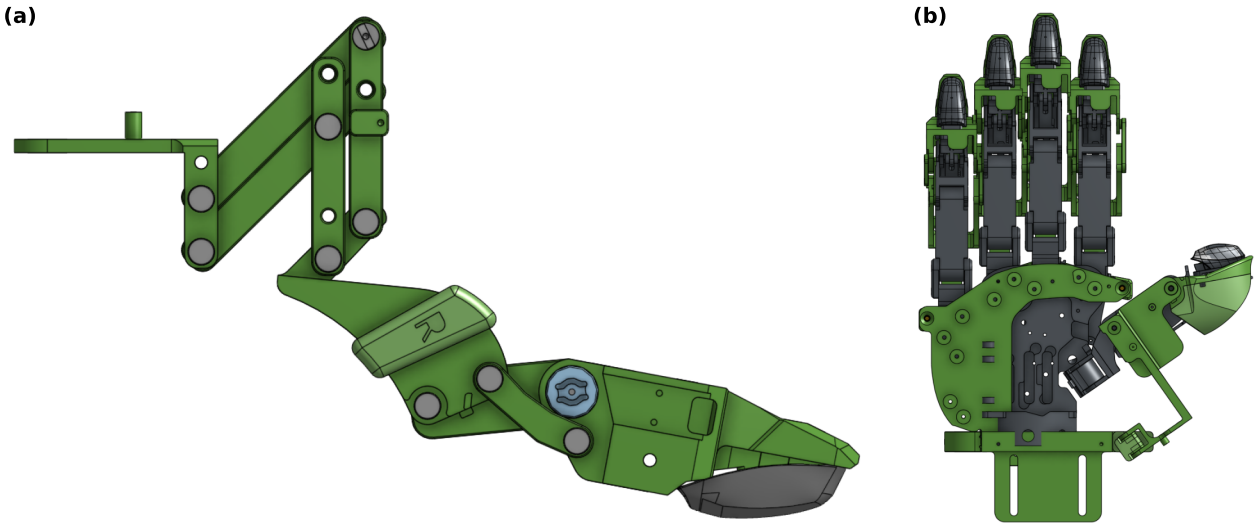


Figure 44: GEAR DexUMI kinematic design. (a) A remote-center-of-motion linkage places a virtual joint at the robot hand's AB/AD and flexion joint centers. (b) The kicked-out thumb linkage keeps the robot hand's thumb kinematics while clearing the operator's thumb.

Contact surfaces and camera. The fingertip and palm tactile-sensor contact surfaces match the robot hand's, and the fingertip geometry is reproduced on the exoskeleton to reduce the visual gap between human collection and robot execution (Fig. 45b). Non-essential contact surfaces are removed to improve wearability. The camera mounting angle and alignment are carried over to the robot hand's mount, so the collected images match the deployed field of view (Fig. 45c). Finally, the chassis is anodized green so that a color threshold can segment it for the masked-video augmentation (Sec. E.6).

Sensing. The exoskeleton fingertips report force in the robot hand's own tactile format: five fingertip sensors with two pads and three force axes each, 30 channels per hand. Per hand, an iPhone tracks a wrist frame 12.6 cm from its lens, which a per-arm hand-eye calibration maps onto the robot, and the same 640×400, 45 fps wrist and head cameras serve collection and deployment.

E.5. DexUMI Post-Training Recipe

For the DexUMI results of Sec. B.3, we post-train the 2B GR00T 2 model from the GR00T 2 pretraining checkpoint for 60,000 steps at a learning rate of 3×10^{-5}, with 224×320 images from the wrist cameras and, optionally, the head camera. The policy predicts 40-step action chunks at stride 2 on the 45 fps data: relative end-effector poses for each arm and, for each hand, either absolute joint positions with joint torques or offset joint positions in their place (Eqs. (1) and (3)).

E.6. DexUMI WAM-Native Inpainting

GR00T 2 addresses the visual gap between the exoskeleton and the robot hand inside training rather than by editing video: it corrupts the exoskeleton region so that the model cannot rely on exoskeleton-specific appearance. By WAM-native inpainting we mean this in-model treatment, in place of offline video editing. Our earlier pipeline, following the original DexUMI (Xu et al., 2025), filmed each demonstration replayed on a real robot hand, segmented the exoskeleton from manually picked keypoints, inpainted it with ProPainter, and composited the robot hand back in, at the cost of robot time, manual input, and GPU-heavy segmentation and inpainting.

Exoskeleton masks. We segment the green regions in every camera video with HSV color bands, morphological cleanup, and a ± 2-frame temporal majority vote, and store each mask as a lossless binary video beside its RGB stream.

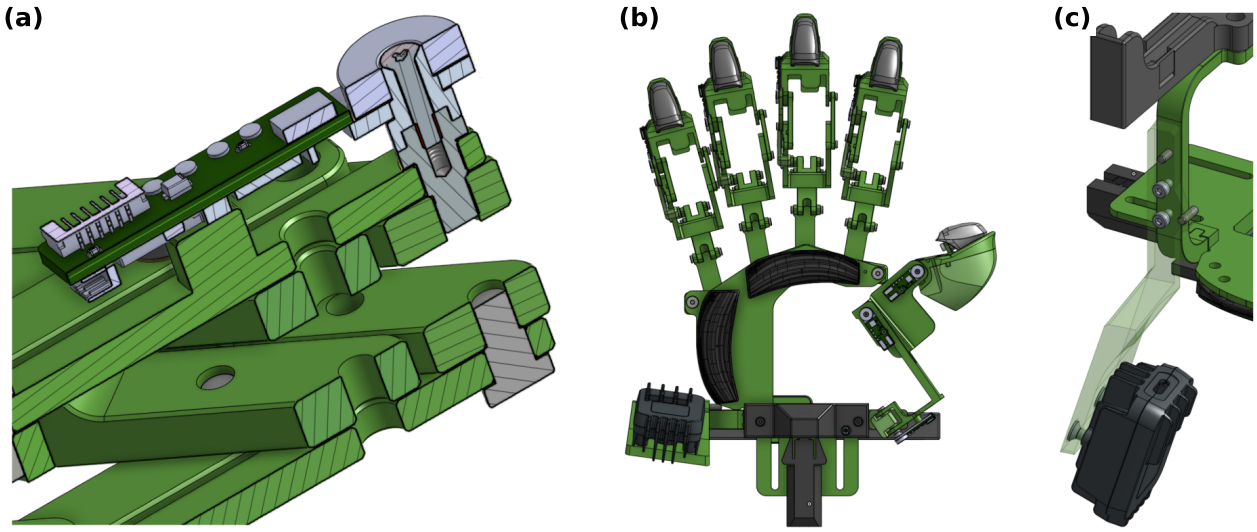


Figure 45: GEAR DexUMI joints, contact surfaces, and camera. (a) Double-sided joint support and a custom pin with a direct keying interface to the joint encoder. (b) Fingertip and palm contact surfaces replicate the robot hand's tactile sensors. (c) The camera mount matches the robot hand's mounting angle and alignment.

Masked-video augmentation. During training only, GR00T 2 corrupts the masked region before cropping and resizing, with one draw of corruption parameters per sample shared across frames and views, blended in through a feathered mask:

$$\tilde{x} = x + \alpha \odot \big(c(x) - x\big), \qquad \alpha = G_\sigma * m, \tag{9}$$

where m is the binary mask and G_σ a Gaussian blur with $\sigma = 6$ px at 640 px width, so the corruption fades out over a narrow band around the mask and pixels farther from it stay bit-identical. The default corruption c rotates hue over the full color wheel, scales saturation by $U[0.1, 1.2]$ and value by $U[0.2, 1.15]$, and adds luminance noise. The saturation and value ranges deliberately reach dark, desaturated colors because the deployed robot hand is largely black. Masks are dropped from the deployed model's inputs, so a checkpoint needs only RGB and inference frames pass through unchanged.

F. RL Post-Training

Pretraining equips GR00T 2 with broad visual and motor priors, but reliable real-world manipulation often requires adaptation to precise alignment, contact dynamics, and physical interaction. Reinforcement learning (RL) provides a complementary approach to supervised fine-tuning (SFT), enabling policies to improve through robot experience and task-level feedback.

Applying RL to GR00T 2 introduces unique challenges. Unlike action-only policies, GR00T 2 jointly generates actions and predicted visual futures through iterative denoising. Policy optimization must therefore accommodate joint video-action computation while enabling effective exploration. Furthermore, physical interaction is expensive, requiring efficient algorithms and infrastructure to make iterative post-training practical.

We address these challenges through three complementary components. First, we develop direct GR00T 2 post-training methods, including GRPO-based optimization (Shao et al., 2024) and self-imitation from successful RL exploration, alongside residual RL for localized corrections (Sec. F.1). Second, we leverage *AutoRL*, an autonomous research system that discovers and improves RL post-training algorithms through closed-loop experimentation in simulation (Sec. F.2). Third, we develop scalable infrastructure that coordinates robot interaction, training, and evaluation while preserving experience and minimizing iteration overhead (Sec. F.3). We demonstrate these capabilities on four real-world precision manipulation tasks (Sec. F.4). We also investigate RL post-training through interaction with a learned neural simulator. We adapt DreamDojo (Gao et al., 2026) to recorded policy rollouts, then demonstrate that GR00T 2 can improve through world-model-based RL (Sec. F.5).

F.1. RL-Based Adaptation of World-Action Models

We investigate two complementary strategies for adapting GR00T 2. *Direct GR00T 2 post-training* updates the pretrained model through RL or self-imitation, while *residual RL* freezes GR00T 2 and learns a lightweight corrective policy. The former improves the underlying generative policy, whereas the latter targets precision-critical stages where GR00T 2 already produces approximately correct behaviors.

F.1.1. Direct GR00T 2 Post-Training

Stochastic Exploration with CPS. To enable exploration and likelihood-based policy optimization (Liu et al., 2025; Lei et al., 2026), we replace GR00T 2's deterministic Euler action updates with Coefficients-Preserving Sampling (CPS) (Wang and Yu, 2025) during rollout. Noise is injected only into the action denoising process. Video denoising remains deterministic given its initial latent, while video features continue to condition action generation.

Following DPPO (Ren et al., 2025) and RL-100 (Lei et al., 2026), we optimize GR00T 2 through its action denoising process. Let t index environment decisions and k index action-denoising events. GR00T 2 uses four such events: joint passes ($k = 0, 2$) update video and action latents through a shared DiT, while cached passes ($k = 1, 3$) update action tokens using video and state keys and values from the preceding joint pass (Fig. 46). After denoising, the first H_e actions are executed, where H_e is the executed action horizon.

Let θ denote the GR00T 2 parameters, $\mathbf{a}_t^{(k)}$ the intermediate action latent at environment decision t and denoising event k, and $\mathbf{v}_{\theta,t}^{(k)}$ the corresponding action velocity in the noise-minus-data convention. The noise schedule σ_k decreases from $\sigma_0 = 1$ to $\sigma_4 = 0$. The clean-action and noise estimates are

$$\begin{aligned} \hat{\mathbf{a}}_{\mathrm{data},t}^{(k)} &= \mathbf{a}_t^{(k)} - \sigma_k \mathbf{v}_{\theta,t}^{(k)}, \\ \hat{\boldsymbol{\epsilon}}_t^{(k)} &= \mathbf{a}_t^{(k)} + (1 - \sigma_k)\mathbf{v}_{\theta,t}^{(k)}. \end{aligned} \tag{10}$$

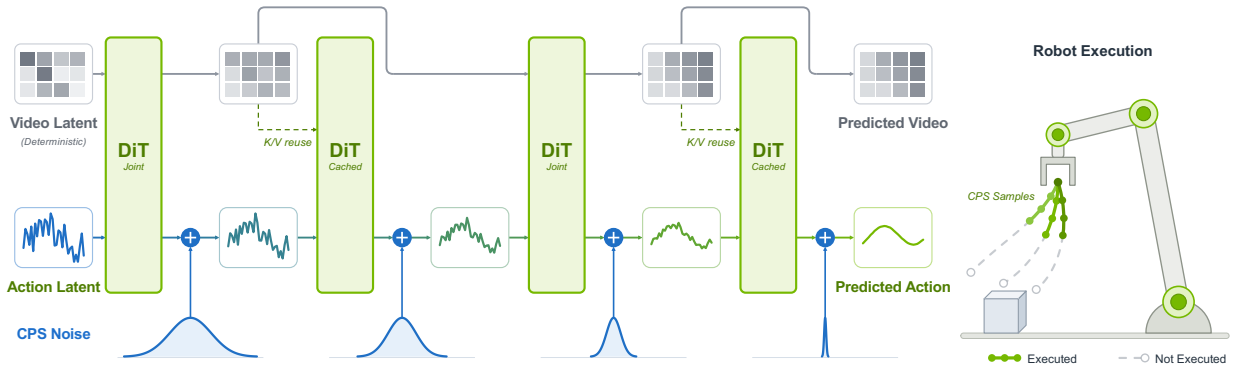


Figure 46: Joint video-action denoising with CPS exploration. Four action-denoising events alternate between joint and cached passes. CPS adds fresh noise only to executed action coordinates. Video denoising is deterministic given its initial latent. GRPO scores joint action transitions and backpropagates through their joint forward passes, with carried latents detached.

Using clipped transition standard deviations (Lei et al., 2026), we sample

$$\begin{aligned} \boldsymbol{\mu}_{\theta,t,k} &= (1-\sigma_{k+1})\hat{\mathbf{a}}^{(k)}_{\mathrm{data},t} + \sigma_{k+1}\cos(\eta\pi/2)\hat{\boldsymbol{\epsilon}}^{(k)}_{t}, \\ s_k &= \mathrm{clip}(\sigma_{k+1}\sin(\eta\pi/2), s_{\min}, s_{\max}), \\ \mathbf{a}^{(k+1)}_{t,\mathrm{exec}} &= \boldsymbol{\mu}_{\theta,t,k,\mathrm{exec}} + s_k\boldsymbol{\xi}_{t,k}, \\ \boldsymbol{\xi}_{t,k} &\sim \mathcal{N}(\mathbf{0},\mathbf{I}). \end{aligned} \tag{11}$$

Here η controls stochastic exploration, $\boldsymbol{\mu}_{\theta,t,k}$ is the CPS transition mean, and s_k is its standard deviation, bounded by $s_{\min} > 0$ and $s_{\max}$. The subscript exec denotes active coordinates in the first H_e action rows. Noise injection and likelihood evaluation are restricted to these coordinates, excluding padding. CPS provides explicit Gaussian transition densities for subsequent policy optimization.

Online RL with GRPO. We optimize GR00T 2 from its own successful and failed rollouts with a GRPO-style algorithm developed through AutoRL (Sec. F.2). A training round collects N_g attempts for each selected reset context g (a task paired with a scene layout) under a fixed behavior policy. Attempt i of context g receives a binary return $R_{g,i} \in \{0,1\}$.

Leave-one-out advantages with a context memory. With a baseline computed solely from the current group, uniform outcomes yield zero reward-based advantages. We incorporate a historical per-context mean to retain a learning signal when current outcomes differ from past performance. The baseline for attempt i pools the other $N_g - 1$ returns of its context with h_g pseudo-returns fixed at the context's historical mean m_g:

$$b_{g,i} = \frac{h_g\, m_g + \sum_{j\neq i} R_{g,j}}{h_g + N_g - 1}, \qquad A_{g,i} = R_{g,i} - b_{g,i}. \tag{12}$$

With $h_g = 0$ this is the leave-one-out baseline of RLOO (Ahmadian et al., 2024). We use $h_g = 0$ only on a context's first visit and $h_g = h = 2$ afterwards. A uniform failure in a context that used to succeed at rate m_g now yields a negative advantage of magnitude $h\, m_g/(h + N_g - 1)$, and a uniform success in a historically weak context yields the mirror image.

Let $T_{g,i}$ be the number of environment decisions in an attempt. The attempt's advantage is spread evenly over

its decisions and rescaled once per batch,

$$\hat{A}_{g,i,t} = c\,\frac{A_{g,i}}{T_{g,i}}, \qquad c = \frac{\sum_{g,i} T_{g,i}}{\sum_{g,i} |A_{g,i}|}, \tag{13}$$

so that long attempts do not dominate merely by containing more decisions, and c is applied only when the batch has nonzero advantage mass. The sums defining c run over the collected attempts.

Persistent context table. The historical means m_g live in a global table indexed by context. The table persists across rounds and, through checkpointing, across training segments, so it records how well the policy has done in every context so far. Once a round's advantages have been computed, each visited context is updated by an exponential moving average of its round mean,

$$m_g \leftarrow \alpha_{\mathrm{EMA}}\,\mu_g + (1-\alpha_{\mathrm{EMA}})\,m_g, \qquad \mu_g = \frac{1}{N_g}\sum_i R_{g,i}, \tag{14}$$

with $\alpha_{\mathrm{EMA}} = 0.25$ and $m_g = \mu_g$ on the first visit. Unvisited contexts are left untouched. Updating the table after advantage computation ensures that the historical term uses only preceding rounds.

Clipped transition-level objective for a denoising policy. GR00T 2 produces actions by iterative denoising, so the exact action log-likelihood that GRPO would compare between policies is not available. CPS, however, gives a Gaussian density for every action-denoising transition, and we score the policy on those transitions instead. On the selected joint denoising events $\mathcal{K} = \{0, 2\}$ we maximize the clipped surrogate (Liu et al., 2025; Lei et al., 2026; Schulman et al., 2017)

$$\mathcal{J}(\theta) = \mathbb{E}_{(g,i,t)\sim\mathcal{B},\; k\sim\mathcal{K}}\Big[\min\left(r_{g,i,t,k}\,\hat{A}_{g,i,t},\;\mathrm{clip}(r_{g,i,t,k}, 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\,\hat{A}_{g,i,t}\right)\Big], \tag{15}$$

where $\mathcal{B}$ is the batch of recorded decisions, ϵ_{clip} is the clipping threshold, and $r = \exp(\ell_\theta - \ell_{\mathrm{old}})$ is the current-to-behavior likelihood ratio for one recorded transition. Gaussian log densities condition on the observation context and input action latent at event k and sum over executed coordinates. Ratios are not multiplied across events or decisions. In practice, rather than sampling k, each minibatch scores one joint event, alternating between $k = 0$ and $k = 2$. Recorded action latents are teacher-forced, while video computation is reconstructed under the current parameters. Gradients flow through the selected joint pass, including the shared DiT, state/action encoders, and action decoder. Carried latents are detached, yielding a local surrogate without backpropagation through the full denoising chain.

GRPO-Induced Self-Imitation. Online RL enables GR00T 2 to discover successful behaviors through exploration, while supervised learning provides a potentially more stable optimization approach for consolidating these behaviors. We investigate GRPO-induced self-imitation, combining the exploration capabilities of RL with the optimization stability of supervised fine-tuning.

We retain successful episodes from the full set of rollouts used to train the best-performing GRPO checkpoint, forming $\mathcal{D}_{\mathrm{succ}}$. Starting from the same SFT initialization as GRPO, we perform one round of fine-tuning using GR00T 2's original joint video-action objective:

$$\mathcal{L}_{\mathrm{SI}}(\theta) = \lambda_{\mathrm{v}}\mathcal{L}_{\mathrm{video}}(\theta; \mathcal{D}_{\mathrm{succ}}) + \lambda_{\mathrm{a}}\mathcal{L}_{\mathrm{action}}(\theta; \mathcal{D}_{\mathrm{succ}}). \tag{16}$$

Here $\mathcal{L}_{\mathrm{video}}$ and $\mathcal{L}_{\mathrm{action}}$ denote GR00T 2's original video and action denoising objectives, and λ_{v} and λ_{a} control their relative weights.

Task success determines which trajectories are selected, while the learning signal comes from the original video and action supervision. This enables GR00T 2 to benefit from RL-discovered successful behaviors while optimizing a standard supervised objective, without additional robot rollouts, demonstrations, or human

corrective actions.

F.1.2. Residual Reinforcement Learning

Direct GR00T 2 post-training updates the generative policy itself. A complementary approach is to retain GR00T 2's pretrained capabilities and learn targeted corrections for precision-critical stages, where approximately correct trajectories can still fail because of small alignment or contact errors.

We keep GR00T 2 frozen and train a lightweight residual policy to correct its actions online. Unlike direct GR00T 2 training, residual RL does not use CPS. Instead, it relies on the stochastic residual actor for exploration.

The residual operates within designated task stages. At each decision j, a tanh-squashed Gaussian actor π_ϕ receives observations $\mathbf{z}_j$ comprising camera images, proprioception, GR00T 2's forecast end-effector targets, and the current correction offset. The actor produces a bounded residual action $\mathbf{a}_j$ over the next control horizon.

Per-tick corrections are accumulated into bounded offsets and composed with GR00T 2's targets:

$$\begin{aligned} \mathbf{o}_{j+1} &= \mathrm{clip}(\mathbf{o}_j + \gamma\delta\mathbf{a}_j, -\mathbf{B}, \mathbf{B}), \\ \mathbf{x}_j^{\mathrm{cmd}} &= \mathbf{x}_j^{\text{GR00T 2}} \oplus \mathbf{o}_j. \end{aligned} \tag{17}$$

Here $\mathbf{o}_j$ is the accumulated correction offset, δ is the per-tick correction scale, γ is the correction gain, and $\mathbf{B}$ denotes axis-wise offset bounds. The operator $\oplus$ composes the correction offset with GR00T 2's end-effector target. Zero residual action leaves the current offset unchanged, while zero offset recovers the base policy.

Following HIL-SERL (Luo et al., 2025), we train the residual actor using Soft Actor-Critic (SAC) (Haarnoja et al., 2018) with prior demonstrations and online interaction, including occasional human corrections. Evaluation episodes are manually scored using binary task-specific success criteria.

We draw half of each critic batch from online data and half from demonstrations, and train an ensemble of critics using bootstrapped returns. The actor objective combines entropy-regularized policy improvement with a zero-residual prior and imitation of successful residual actions:

$$\begin{aligned} \mathcal{L}_\pi(\phi) = \mathbb{E}_{\mathbf{z}\sim\mathcal{D}_{\mathrm{RL}},\,\tilde{\mathbf{a}}\sim\pi_\phi(\cdot|\mathbf{z})}\left[\alpha\log\pi_\phi(\tilde{\mathbf{a}}\mid\mathbf{z}) - \lambda_Q\bar{Q}_\psi(\mathbf{z},\tilde{\mathbf{a}})\right] \\ + \lambda_0\mathbb{E}_{\mathbf{z}\sim\mathcal{D}_{\mathrm{RL}}}\left\|\boldsymbol{\mu}_\phi(\mathbf{z})\right\|^2 \\ + \lambda_{\mathrm{bc}}\mathbb{E}_{(\mathbf{z},\mathbf{a})\sim\mathcal{D}^+}\left\|\boldsymbol{\mu}_\phi(\mathbf{z}) - \mathbf{a}\right\|^2. \end{aligned} \tag{18}$$

Here ϕ and ψ denote the residual actor and critic parameters, respectively. The distribution $\mathcal{D}_{\mathrm{RL}}$ represents observations sampled from the training replay buffer. The function $\bar{Q}_\psi$ is the ensemble critic mean, $\boldsymbol{\mu}_\phi$ is the actor mean, and $\mathcal{D}^+$ contains successful residual experience. The entropy coefficient α controls exploration, while λ_Q, λ_0, and λ_{bc} weight the critic, zero-residual regularization, and behavior-cloning terms, respectively.

The zero-residual prior discourages unnecessary corrections, while successful experience reinforces effective behavior. Residual RL thus enables targeted adaptation without modifying pretrained GR00T 2 weights.

F.2. Automated Discovery of RL Post-Training Algorithms

Developing effective RL post-training algorithms for robot foundation models involves choices across objectives, exploration, credit assignment, reward design, and rollout horizons. These decisions interact with task characteristics and model architectures, making algorithm development an iterative process of experimentation and refinement.

We leverage *AutoRL*, an autonomous research framework for closed-loop experimentation, to systematically improve RL post-training algorithms for GR00T 2. The research agent analyzes training failures, proposes hypotheses, modifies code, and evaluates candidate algorithms under fixed protocols. We ran AutoRL for

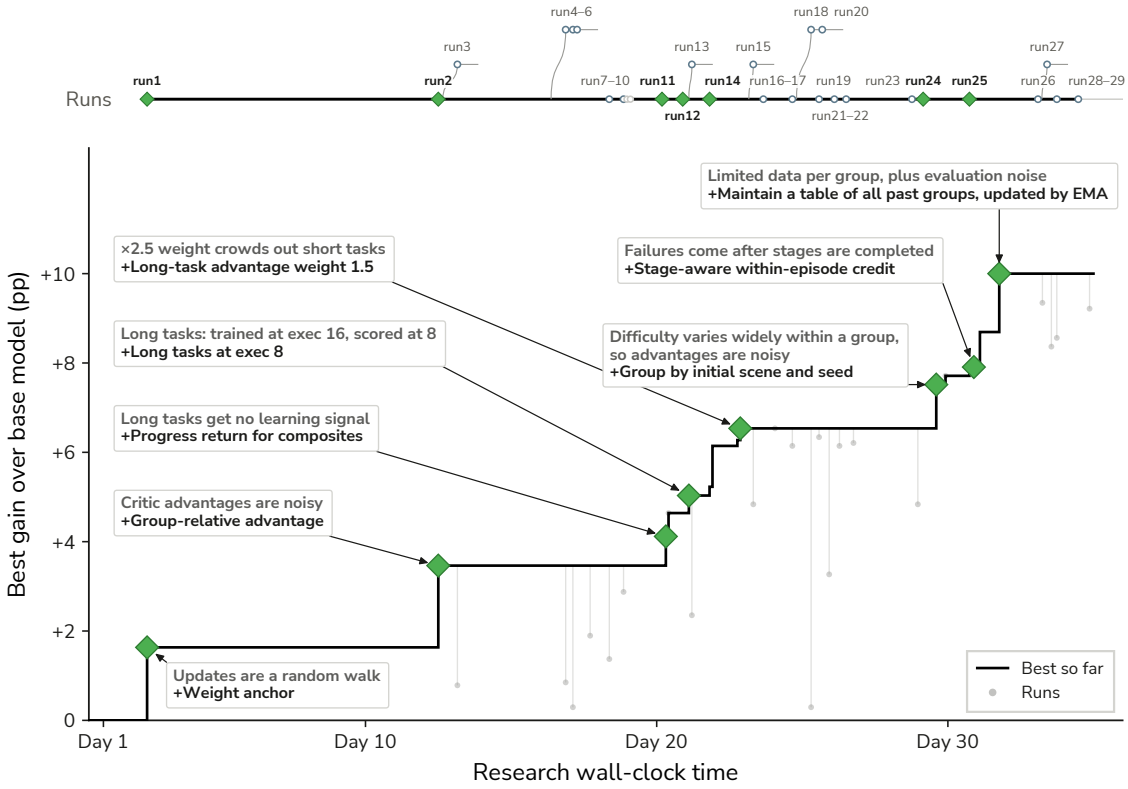


Figure 47: Autonomous RL recipe discovery with AutoRL. Hill-climbing performance on RoboCasa365 across successive agent-driven experiments. The curve shows the best evaluation performance achieved.

approximately 35 days on RoboCasa365 (Nasiriany et al., 2026), using an early developmental GR00T 2 checkpoint and the Atomic-Seen and Composite-Seen tasks.

F.2.1. Autonomous Experimentation Harness

Each AutoRL discovery round follows four stages. First, the agent analyzes previous experiments, diagnostic metrics, and failure modes to identify bottlenecks. Second, it proposes up to four mutually orthogonal hypotheses, each specifying an isolated mechanism, expected outcome, and falsification criterion. A critic filters out redundant or insufficiently testable proposals. Third, valid hypotheses are implemented in independent branches, locally tested, and submitted for cluster training and evaluation. Finally, candidates are compared with the current best baseline and promoted only when improvements exceed a predefined margin. At most one candidate is promoted per round, while rejected hypotheses remain available as negative evidence.

This harness enables systematic exploration of algorithmic modifications beyond conventional hyperparameter tuning.

F.2.2. Discovering RL Recipes

AutoRL starts from a standard PPO implementation and iteratively improves the post-training recipe through successive experiments (Fig. 47). Below, we summarize the eight changes marked in Fig. 47 by the component they affect, each with the best gain over the base model reached after it.

Weight anchoring and longer rollouts. Early runs showed the actor drifting rapidly from the pretrained weights, losing existing skills before any useful RL gradient appeared, and rollouts on the long-horizon RoboCasa365 tasks ending before the task could be completed. The agent added an L_2 penalty toward the pretrained weights and extended the rollout horizon. Together these give 1.6 percentage points.

Replacing the critic (group-relative baseline). The critic's explained variance stayed near zero, so its advantages were noise. More capacity and warm-up did not help. The agent removed it in favor of a group-relative baseline in the style of GRPO (Shao et al., 2024), the $h_g = 0$ case of Eq. (12), and extended the horizon further. Cumulative gain: 3.5 percentage points.

Shaping the return (stage progress). Six long-horizon composite tasks still did not move: with binary success every attempt in a group failed identically, leaving no relative signal. For these simulation experiments, the agent introduced stage-progress returns in $[0, 1]$, assigning partial credit according to the furthest subgoal milestone reached. The real-robot experiments instead use binary task-success returns, as defined in Sec. F.1.1. Cumulative gain: 4.1 percentage points.

Balancing long and short tasks. Long tasks were trained with an execution horizon of 16 but scored with 8, and training them at 8 as well raised the cumulative gain to 5.0 percentage points. An advantage weight of 2.5 on long tasks crowded out the short ones, and lowering it to 1.5 gave 6.5 percentage points.

Less noisy advantages (grouping and stage credit). Task difficulty varied widely within a group, which made the group-relative advantages noisy. Grouping rollouts by initial scene and seed raised the cumulative gain to 7.5 percentage points. Failures often came after some stages had already been completed, so the agent added stage-aware credit within each episode, which gave 7.9 percentage points.

Remembering each context (persistent baseline). Progress rewards do not help contexts the policy already solves every time or never advances at all. Their groups stay uniform and a current-group baseline gives them zero advantage, and on RoboCasa365 such contexts are most of the data. The agent gave every context a memory of its past returns, an exponential moving average across rounds (Eq. (14)), folded into the leave-one-out baseline as a small number of pseudo-observations (Eq. (12)). This baseline retains a learning signal for uniform-outcome groups when their current returns differ from past performance. The same context statistics also guide rollout allocation away from consistently solved contexts. Cumulative gain: 10.0 percentage points.

These experiments demonstrate how AutoRL systematically discovers effective post-training strategies: stabilizing policy optimization, replacing an uninformative critic, balancing long and short tasks, improving credit assignment, and keeping a learning signal alive on saturated and unsolved tasks. The resulting algorithm forms the foundation of our direct GR00T 2 post-training approach, which we further adapt to GR00T 2's joint video-action architecture and real-world robot interaction. As demonstrated in Sec. F.4, this approach achieves substantial improvements on contact-rich manipulation tasks.

F.3. Online RL Post-Training Infrastructure

Real-robot RL requires repeated coordination among data collection, reward annotation, policy optimization, and evaluation. These stages have different resource requirements and execution rates, while physical interaction and scene resets constrain collection throughput. We develop shared infrastructure that supports heterogeneous learning algorithms, reduces iteration overhead, and preserves experience across failures (Fig. 48).

Shared Experience Buffer. Our infrastructure uses a shared experience buffer backed by Lance (Pace et al., 2025). Experience records are constructed incrementally as collection, scoring, and batch preparation produce their outputs. Workers can selectively access relevant fields, while training updates use pinned table versions to ensure stable inputs (Lance Contributors, 2026). Common record identities and policy provenance associate observations, rewards, and updates, and algorithm-defined schemas support different post-training methods within the same data infrastructure.

Resource-Efficient Execution. Collection, training, and evaluation use independently configurable resource pools. Compatible GPU resources can be reassigned according to workload dependencies, accommodating both on-policy and off-policy learning. Incremental uploads and prefetching overlap collection, data transfer, scoring, decoding, and training preparation, reducing idle time between iterations.

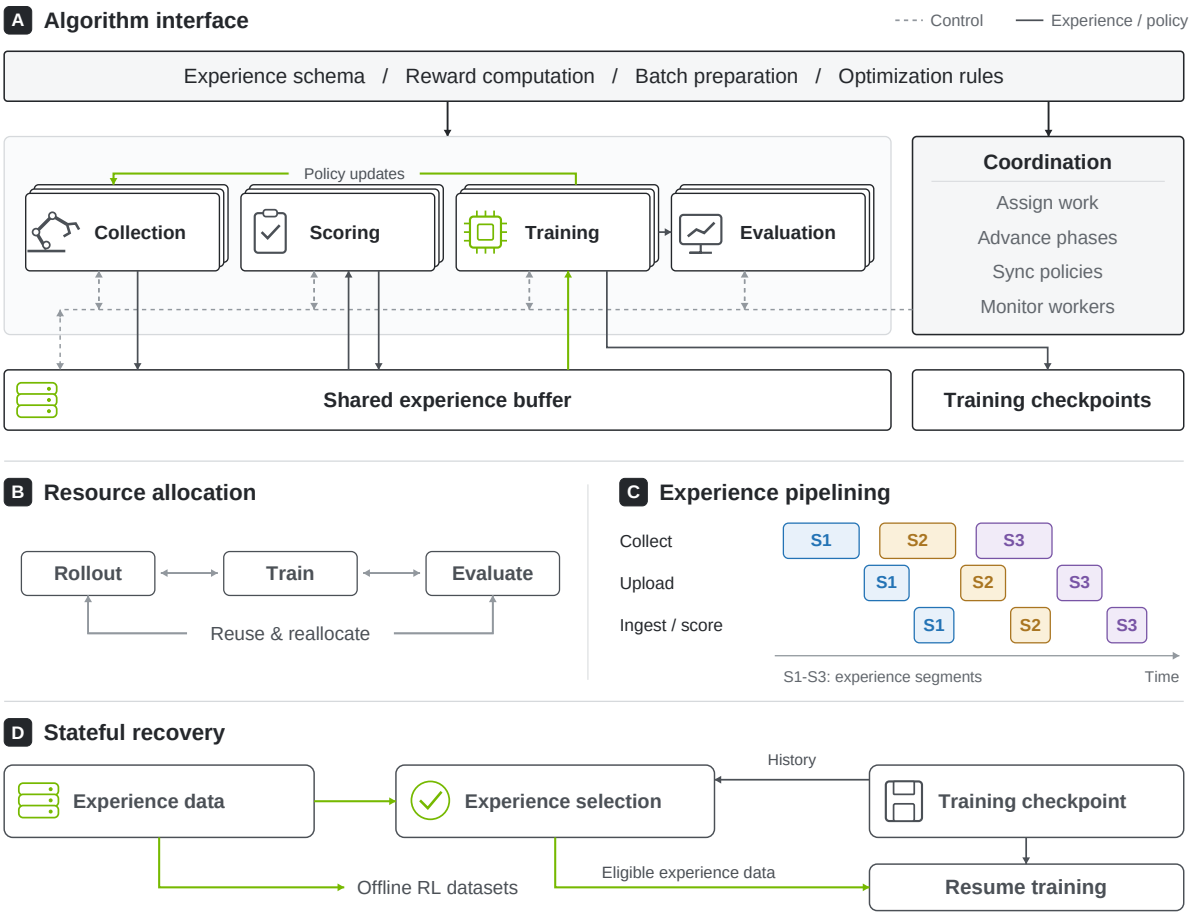


Figure 48: Online post-training infrastructure. (A) Algorithm interfaces, worker groups, and shared services. (B) Resource allocation across execution roles. (C) Experience pipelining. (D) Stateful recovery. Execution patterns are illustrative.

Stateful Recovery. Physical experience is expensive to reproduce, making robustness to interrupted training essential. Our infrastructure retains experience data and feedback alongside model and optimizer checkpoints. Recovery distinguishes experience already reflected in a restored checkpoint, data associated with discarded updates, and unused data. Compatible experience can subsequently be reused for resumed training or offline learning, reducing redundant physical interaction. This complements prior work on distributed RL execution and durable robot experience data (Yu et al., 2026a; Zang et al., 2026a,b; Wang et al., 2026; Chen et al., 2026c).

Execution Efficiency. We evaluate infrastructure efficiency in simulation and real-robot training.

In RoboCasa365 GRPO, rollout and training alternate on a shared 40-GPU pool while evaluation retains 40 GPUs. This reduces total allocation from 120 to 80 GPUs and enables evaluation to overlap training. Across eight rounds and nine evaluations, execution time decreases from 14.1 to 8.8 hours, corresponding to approximately $1.6\times$ improvement in update throughput.

For single-robot GRPO (Tab. 18), experience pipelining and trainer-ready records reduce upload and conversion time from approximately 50 minutes to one minute, while transferring only tuned weights reduces policy download time from approximately 30 minutes to one minute. Total iteration time decreases from between 140 and 160 minutes to between 62 and 72 minutes, yielding approximately $2.2\times$ iteration throughput. Incremental uploads overlap data transfer with physical collection, leaving only a short data-handling delay at the end of each round. These mechanisms reduce overhead without requiring faster robot execution, allowing more

Table 18: Per-iteration time breakdown for single-robot GRPO. All times are reported in minutes.

Stage	Baseline	Ours	Notes
Collection	50–60	50–60	Bounded by physical robot execution
Upload and conversion	50	$\sim$1	Asynchronous upload yields trainer-ready data
Training	10–20	10	Fixed compute budget per iteration
Policy download	30	$\sim$1	Only tuned weights are transferred
Total	**140–160**	**62–72**	$\approx 2.2\times$ higher iteration throughput

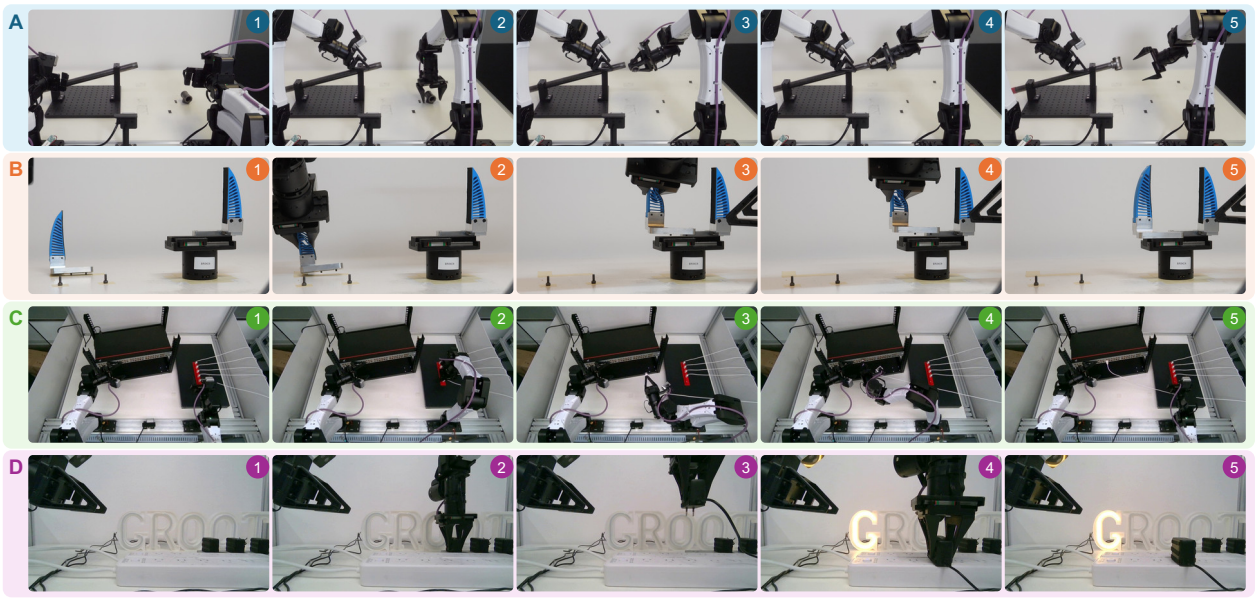


Figure 49: Real-world RL tasks. (A) Fixed-pipe fittings: grasp, align, engage threads, turn, and release. (B) Finger placement: grasp, align two pins with the base holes, insert, and release. (C) Cable insertion: plug an Ethernet cable into a network switch. (D) Charger insertion: plug a charger into an outlet.

efficient use of limited physical interaction resources.

F.4. Precision Manipulation through RL Post-Training

We evaluate three complementary post-training approaches on four real-world contact-rich manipulation tasks, demonstrating how GR00T 2 improves through RL-based learning and targeted policy adaptation.

Tasks. Our evaluation includes four precision manipulation tasks (Fig. 49). In *fixed-pipe fittings*, the robot grasps a threaded tee fitting, aligns it with a horizontally mounted pipe, engages the threads, and completes one full turn. This is the Fixed-Pipe Fittings task of Tab. 1a, but each RL trial starts from a fitting pose that the post-trained policy struggles with. In *finger placement*, the robot inserts two 3 mm pins on a gripper finger into matching holes on a fixed base, requiring accurate position and yaw alignment.

In *cable insertion*, the robot inserts an Ethernet connector into a designated network-switch port, with success determined by network-link connectivity. In *charger insertion*, the robot inserts a two-prong charger into a specified outlet within a 2×6 grid. These tasks involve tight geometric tolerances, partial occlusion, and contact-sensitive execution.

Evaluation Protocol. All evaluated methods build on the same SFT-initialized GR00T 2 checkpoint, either as an initialization or as a frozen base. GRPO directly updates GR00T 2 using both successful and failed

trajectories. Self-imitation reuses successful GRPO rollouts without additional collection. Residual RL instead learns a separate controller on top of the frozen base, using human-assisted training experience.

For each task, evaluations use the same target distribution, reset conditions, and time limits, with no human intervention during evaluation. Trials are independent of training rollouts, and we report per-task success rates. An episode is considered a failure if misalignment prevents the policy from recovering or if the insertion exceeds the time limit.

Real-World Results. *Direct RL on GR00T 2 improves precision manipulation.* GRPO increases success rates on all four tasks (Fig. 5), raising fixed-pipe fittings from 66% to 86%, finger placement from 58% to 72%, cable insertion from 76% to 92%, and charger insertion from 40% to 94%. With 50 trials per task, the gains on charger insertion and fixed-pipe fittings are significant at the 5% level (two-sided Fisher exact test), whereas those on cable insertion (38 to 46 successes) and finger placement (29 to 36) are not conclusive at this sample size.

Self-imitation consolidates successful RL exploration. Self-imitation reaches 88% on fixed-pipe fittings, 98% on finger placement, 94% on cable insertion, and 96% on charger insertion, matching or exceeding GRPO on all four tasks. The gain is most pronounced on finger placement, where success rises from 72% to 98%.

These results highlight the potential of combining RL exploration with supervised policy optimization. RL discovers successful behaviors through interaction, while self-imitation consolidates them under a stable supervised objective. This offers a complementary route to improving GR00T 2 beyond direct policy-gradient optimization.

Residual RL enables targeted adaptation without updating GR00T 2. Residual RL reaches 76% on fixed-pipe fittings, 84% on finger placement, 82% on cable insertion, and 70% on charger insertion, with clear gains over supervised post-training on finger placement and charger insertion. These results show that lightweight corrective controllers can improve precision execution while leaving the pretrained GR00T 2 intact.

These experiments show that direct RL, self-imitation, and residual RL can improve GR00T 2's precision manipulation through complementary forms of post-training.

The preceding experiments improve policies through physical interaction. We next investigate a complementary source of training experience: interaction with a learned environment constructed from recorded policy rollouts.

F.5. Reinforcement Learning with Neural Simulation

In neural-simulation RL, the policy interacts with a learned world model in place of the physical environment. This changes how training experience is generated while retaining reward-based optimization of the pretrained world-action policy.

We use DreamDojo (Gao et al., 2026), initialized from pretrained video priors, to turn recorded robot interactions into a learning environment. Successful and failed attempts provide supervision for task-specific adaptation. A pretrained GR00T 2 policy can then collect further training experience through interaction with the adapted model. Task rollouts are still required to construct this environment, while subsequent policy-learning rollouts are generated in neural simulation.

To support policy learning, the model's predictions must provide usable inputs to GR00T 2 throughout an episode. We provide multi-view camera predictions, align them with executed action chunks, and approximate proprioceptive inputs using action targets. We also maintain visual history across successive world-model calls. We first describe world-model construction and the closed-loop policy interface, including support for long-horizon interaction. We then present the experimental setup, reward modeling and policy optimization, and results in RoboLab and on real-world tasks.

Training Data and Task Adaptation. We first mid-train DreamDojo on multi-view episodes, then adapt the resulting model to downstream tasks. The bimanual YAM corpus contains 8,310 four-view episodes from policy

rollouts and DAgger collection, including failed attempts and recovery. These episodes total approximately 226.8 hours at 30 Hz. The DROID source corpus contains 57,773 episodes, approximately 273 hours at 15 Hz, with three synchronized camera views and recorded joint and gripper actions.

We tile synchronized camera views into one video canvas so that the model predicts them jointly using DreamDojo's existing video backbone and causal variational autoencoder (VAE). YAM uses a 2×2 layout at 480×640 pixels. DROID concatenates three 180×320 views horizontally and pads the canvas vertically to 192×960 pixels.

Video and action sampling are aligned separately. For YAM, we downsample video from 30 to 15 Hz while retaining fourteen-dimensional actions at 30 Hz. Each window contains five conditioning frames and twelve future frames, paired with 24 future actions. DROID video and eight-dimensional actions both remain at 15 Hz, and each window contains five conditioning frames and sixteen future frames. We precompute and cache VAE latents, which are the compressed video representations used by the world model. We sample temporally aligned training windows from this cache, avoiding repeated encoding of the same recorded videos.

Task adaptation. We adapt the mid-trained world model to the scenes and action distributions encountered by the target policy. Successful and failed rollouts provide supervision on the states reached during execution, allowing the model to learn task-relevant visual changes and action effects. We fine-tune the model on these rollouts using the multi-view representation and temporal alignment used during policy interaction. Recorded robot actions and world-model inputs use consistent action coordinates and normalization.

Closed-Loop Simulation Interface. To collect experience for policy optimization, we alternate GR00T 2 action prediction with world-model prediction throughout a task episode. A rollout starts from recorded multi-view observations. GR00T 2 predicts an action sequence, and the world model predicts the camera observations resulting from its executed prefix. These predicted images become the policy's next visual input, allowing GR00T 2 to respond to the consequences of its own actions. Repeating this interaction produces a trajectory for reward evaluation and policy learning.

At interaction round r, GR00T 2 samples an action plan from the current observation and task instruction. Let A_r denote its executed prefix of H actions and C_r the world model's conditioning latents. Initially, C_0 is obtained by encoding five recorded video frames into two latent frames. The mapping g converts actions into the world model's action coordinates and normalization. Future latents and the next conditioning history are then given by

$$\begin{aligned} F_r &\sim p_\phi(\cdot \mid C_r, g(A_r)), \\ C_{r+1} &= \mathrm{Tail}_2([C_r, F_r]). \end{aligned} \tag{19}$$

Here, Tail_2 retains the final two latent frames of the concatenated conditioning and predicted sequence. After decoding, we remove the overlapping conditioning prefix and append the newly predicted images to the policy's visual history. The views and historical frames required by GR00T 2 form its next observation, closing the interaction loop.

The rollout preserves the video-action alignment used during training. For YAM, the world model predicts twelve new video frames at 15 Hz for 24 native actions at 30 Hz. For DROID/RoboLab, it predicts sixteen frames for sixteen actions, both at 15 Hz. We query the policy again after executing this action prefix and truncate the final round at the task horizon.

We exclude the five-frame conditioning prefix from the images returned to the policy. RoboLab decodes the complete latent window at each round. YAM retains the causal decoder cache and decodes only new latents after the first round. This decoder cache is separate from the latent history passed to the world model. For policies requiring proprioception, we initialize the state input from the recorded reset state. At each later policy query, we use the last executed action target as an approximation of the current robot state.

Long-Horizon Prediction. Policy learning requires multiple rounds of interaction, whereas the world model is trained on short video windows. We therefore generate successive segments autoregressively. After the first segment, the model conditions on its own predictions rather than recorded observations. Errors in this history can affect subsequent predictions and policy decisions.

For the DROID-derived model used in RoboLab, we perturb recorded conditioning latents during training to expose the model to imperfect visual histories, while keeping recorded future latents as prediction targets. For a clean conditioning tensor C, we use

$$\widetilde{C} = (1 - \sigma)C + \sigma\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I). \tag{20}$$

The noise strength σ is sampled from 0, 0.0245, 0.0480, and 0.0925 with probabilities 0.2, 0.2, 0.4, and 0.2, respectively. We also explored replacing recorded history with a model-generated preceding segment, with gradients stopped through that history. These preliminary trials showed no clear additional benefit over noise augmentation, which we retain in the final training recipe.

At rollout time, we pass the final two predicted latent frames to the next world-model segment as its conditioning history. This avoids decoding and re-encoding images to construct that history. We still decode predicted latents into images for GR00T 2: the world model carries its history in latent space, while the policy receives camera observations in pixel space.

Experimental Setup. Starting from SFT, GRPO updates GR00T 2 through interaction with the learned environment, while the world model and learned success detector remain fixed. Native RoboLab evaluation and real-world execution test whether these updates improve behavior outside the learned environment.

We collect rollouts from the initial GR00T 2 policy, retaining successful and failed attempts for world-model adaptation and reward supervision. The study covers ten RoboLab tasks and four families of physical manipulation: peg insertion, test-tube insertion into a specified rack hole, bun placement on yellow and white cups, and hardware sorting with nine instructions. Hardware sorting tests whether the policy selects the requested target object and destination from three target object categories and three colored trays in the presence of distractors. Fig. 52 illustrates the physical task setups.

World-model adaptation uses 300 physical rollouts for each insertion task and each of the two bun-placement tasks, and approximately 300 rollouts in total for hardware sorting. The insertion and hardware tasks share a jointly adapted world model, while a separate model is adapted for bun placement. For RoboLab, we adapt one shared world model using 450 training trajectories per task across 64 tasks, with 50 trajectories per task held out. Joint policy RL uses ten tasks from this collection. Each RL run starts from its SFT baseline and keeps the world model and reward model fixed. RoboLab reward supervision uses 450 training and 50 validation trajectories per task. The test-tube success detector uses a separate set of 600 training and 120 validation videos labeled for insertion and release in the designated rack hole.

Reward Modeling and Policy Optimization. *Sparse rewards.* Dense progress rewards are difficult to obtain consistently across manipulation tasks and expensive to annotate. They can also introduce reward hacking when a policy increases a proxy score without completing the requested goal. We therefore use sparse task-completion rewards, learning a binary success detector rather than a continuous progress score.

We train compact success detectors based on VideoMAE (Tong et al., 2022) to provide rewards during RL. For physical videos, a vision-language model (VLM) proposes outcome labels and supporting visual evidence, and targeted human review resolves ambiguous cases. These offline labels supervise the compact detectors, avoiding VLM queries during policy learning. For hardware sorting only, annotated world-model rollouts augment the recorded videos to cover observations encountered in neural simulation. The other reward models use recorded task data.

Success is defined by the requested instruction: choosing the wrong object or destination is a negative outcome.

We use task-specific detectors, including instruction-specific outputs for joint hardware sorting. RoboLab provides native success labels, while physical-task labels use recorded outcomes and reviewed visual evidence. The detector and its task-specific decision rule remain fixed during RL. Final policy evaluation uses native RoboLab outcomes and physical execution, independently of the learned reward model.

Policy optimization. We apply the clipped transition-level policy optimization principle of Sec. F.1.1 to interaction with a fixed learned environment. We collect groups of attempts from the same recorded reset context and score them with the learned success detector. The neural-simulation experiments use Flow-SDE (Liu et al., 2025) for stochastic action denoising, while the physical-RL recipe uses CPS. Both provide tractable transition densities for likelihood-ratio updates.

We optimize a clipped likelihood-ratio objective on selected action-denoising transitions. For each selected transition, we store the sampled latent input and output, denoising index, conditioning inputs, and behavior-policy probability. During optimization, we replay the same transition and recompute its probability under the current parameters with gradients enabled. The sampled transition and noise schedule remain fixed, so the likelihood ratio compares the two policies on the same sampled event.

Consistent replay is necessary because numerical differences can produce an apparent policy change before any update. We disable dropout, preserve positional-encoding precision, and check rollout and replay log-probabilities under unchanged weights before the first optimizer update. Activation checkpointing reduces memory use during backpropagation through the joint world-action model described in Sec. F.1.1. The likelihood ratio scores the selected action transition, with no separate video-likelihood objective or learned critic. In the physical-task recipe, we compute the actor loss using actions up to and including the first chunk in which success is confirmed. Later actions are excluded.

Policy Improvement in RoboLab. We evaluate task success using native RoboLab outcomes, independently of the learned world model and reward model. Fig. 50 compares the SFT policy with the policy obtained by joint RL from that initialization. Each policy is evaluated for 500 episodes per task, or 5,000 episodes in total, using matched observation and action settings and full task horizons. Mean success increases from 51.32% to 70.88%, a gain of 19.56 percentage points.

Batch and group size. Sparse rewards connect exploration to optimization: a larger group can reveal a return difference that a smaller group misses. In RoboLab, we compare $(B, G) = (512, 8)$ and $(1536, 24)$, where B is the number of trajectories collected per round and G the number of attempts per reset group. Both recipes collect 64 groups and perform four optimizer updates per round.

Fig. 51(a) tracks DreamDojo rollout success and native RoboLab success at twelve policy checkpoints, including SFT. Native RoboLab success reaches 71% in evaluations of 100 episodes per policy. The separate final evaluation uses 5,000 episodes per policy and shows an increase from 51.32% to 70.88%. DreamDojo success is measured by the learned reward on training rollouts, whereas RoboLab evaluation uses native task outcomes. Because the sampling procedures and horizons differ, these curves track learning within each setting rather than measure simulator calibration.

Fig. 51(b) compares eleven matched collection budgets through 67,584 trajectories. The larger combined batch/group configuration has higher native RoboLab success at ten budgets, and its success rate is 11.1 percentage points higher on average across the eleven budgets. At the final budget, success is 71% versus 54%, a 17-point difference. At 24,576 trajectories, the rates are 68% and 47%, respectively.

These single-run results favor the larger combined recipe. They also differ in optimization and reset layout: at the final budget, the smaller recipe has made 528 optimizer updates and the larger one 176. Late continuation of the smaller recipe additionally removed a numerical lower log-ratio bound. These differences limit attribution to either size parameter, and one run per configuration does not establish stability across independent runs.

Policy Improvement on Real-World Tasks. We deploy fixed policies on peg insertion, test-tube insertion into a specified rack hole, bun placement on yellow and white cups, and nine-instruction hardware sorting. For

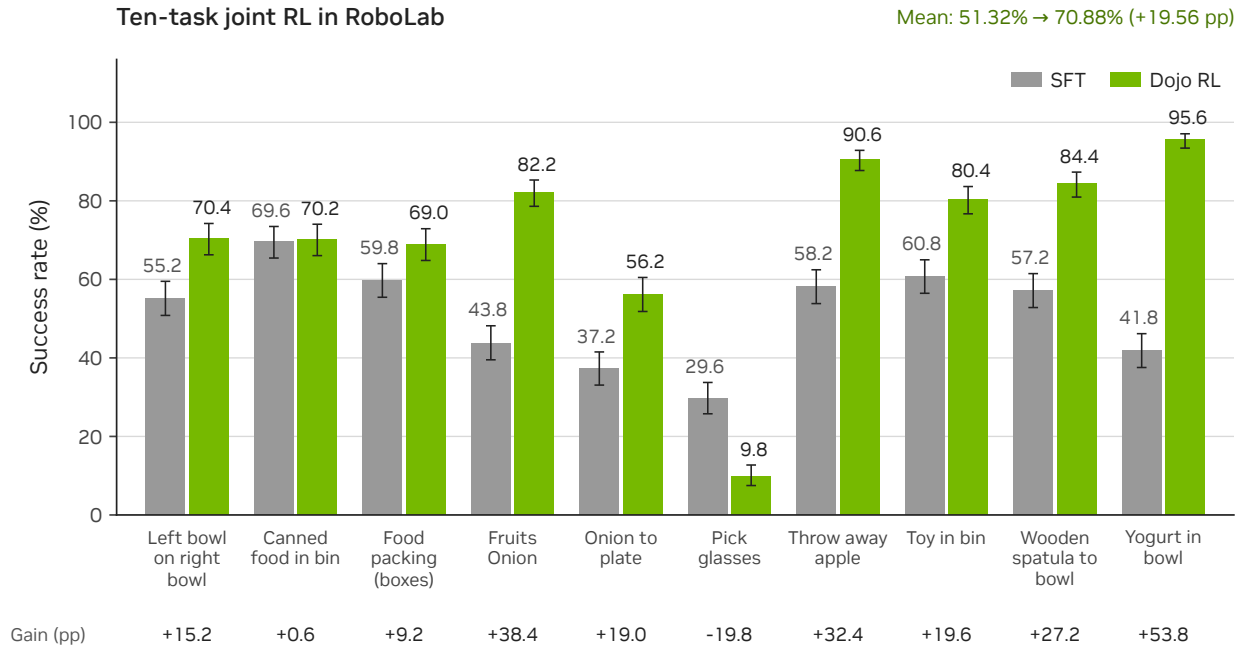


Figure 50: Joint RL on ten RoboLab tasks. Training GR00T 2 in DreamDojo increases mean task success from 51.32% to 70.88%, evaluated over 500 episodes per task and policy. Error bars denote 95% Wilson confidence intervals.

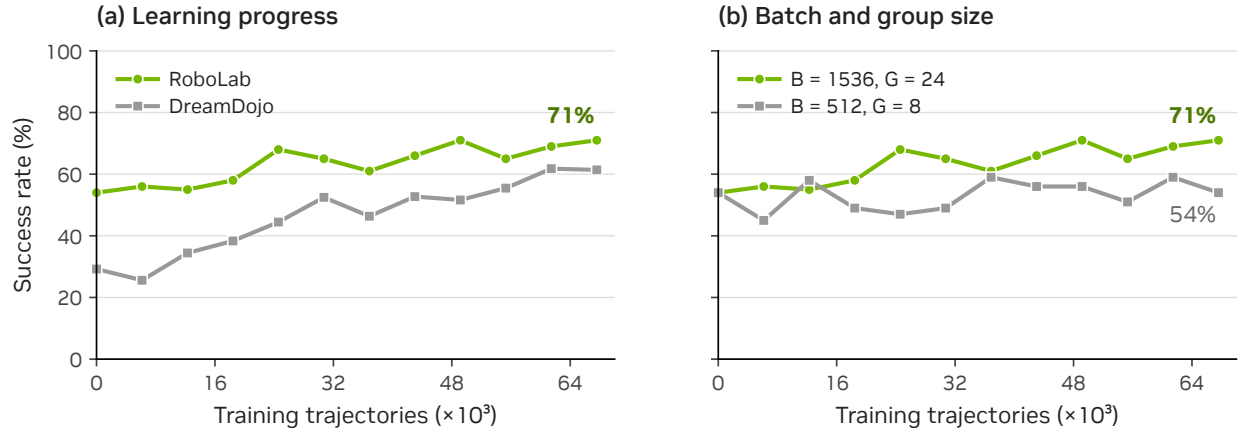


Figure 51: Policy improvement through neural simulation. (a) DreamDojo rollout success and native RoboLab success during RL. (b) Native RoboLab success at matched collection budgets for two batch/group configurations, reaching 71% and 54%. Each configuration has one training run. Batch and group sizes vary together.

insertion tasks, success requires insertion into the requested hole followed by release. For bun placement, the bun must be released onto the requested cup. For hardware sorting, the requested object must be released into the specified tray. Outcomes use operator records and reconciled human-review labels where available. Each RL policy is compared with its SFT initialization. Hardware-sorting trials are included only when they are confirmed to use the same reference scene.

Fig. 52 compares four real-world tasks using 60 trials per policy and task. Peg insertion increases from 43.3% to 81.7%, and insertion into the specified rack hole from 51.7% to 73.3%. Bun placement increases from 63.3% to 76.7% on the yellow cup and from 11.7% to 55.0% on the white cup. The equally weighted four-task

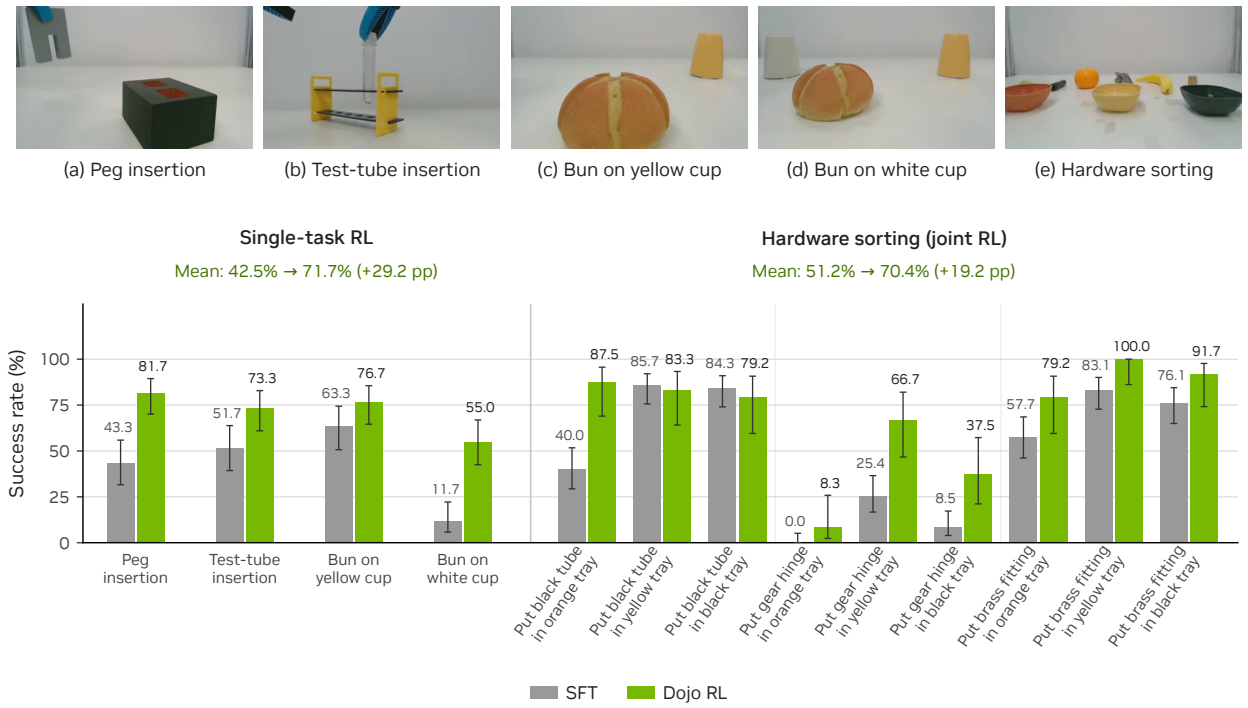


Figure 52: Real-world policy improvement with DreamDojo. Top: two-legged peg insertion, test-tube insertion into the fifth rack hole, bun placement on yellow and white cups, and hardware sorting. Bottom: task success before and after RL, and the nine hardware-sorting instructions are trained jointly with one GR00T 2 policy. Error bars denote 95% Wilson confidence intervals.

mean increases from 42.5% to 71.7%, a gain of 29.2 percentage points.

For hardware sorting, one GR00T 2 policy is trained jointly with RL on nine instructions, formed by pairing three target objects (black tube, gear hinge, and brass fitting) with three destination trays (Fig. 52). Averaging the nine instruction success rates equally, success increases from 51.2% to 70.4%, a gain of 19.2 percentage points.

Each hardware instruction has 70 or 71 SFT trials and 24 RL trials. The RL cohort includes four checkpoint-selection trials and 20 subsequent repeat-evaluation trials per instruction. SFT and RL were collected in separate batches using the same reference scene.

Discussion. These results demonstrate policy improvement using experience generated by a world model adapted to an initial policy's rollouts. GR00T 2 learns in this environment and is evaluated in RoboLab and on real-world robots. The current pipeline still requires task-rollout collection and task-specific world-model adaptation. Reusing the learned environment across new tasks and repeating the full world-model and policy improvement cycle remain directions for future work.

Our experiments focus on individual manipulation skills that serve as building blocks for more complex robot behavior. These skills can be integrated into agentic execution (Sec. G.1), where high-level orchestration, progress monitoring, and recovery coordinate actions across longer task horizons. RL-based adaptation and agentic execution therefore provide complementary mechanisms for improving robot reliability, from fine-grained physical interactions to long-horizon task completion.

Our longer-term goal is to develop scalable post-training methods that enable world-action models to improve consistently through real-world experience, extending their capabilities to increasingly challenging manipulation tasks.

G. Agentic GR00T 2

Pretraining and post-training equip GR00T 2 with increasingly capable manipulation skills, but robust long-horizon execution requires more than a strong underlying policy. Complex tasks involve multiple stages, task-level decisions, and unexpected failures that are difficult to resolve through continuous policy execution alone. Language instructions can also leave important aspects of task intent underspecified, such as which object to manipulate or where to place it.

We explore two complementary approaches to agentic manipulation. First, *agentic skill composition and recovery* integrates GR00T 2 with CaP, enabling higher-level agents to compose manipulation skills, monitor execution, and recover from unexpected failures (Sec. G.1). Second, *steerable world-action generation* exposes GR00T 2's learned capabilities through the WAM API, allowing agents to guide and compose robot behaviors using visual targets (Sec. G.2). Together, these approaches enable higher-level reasoning agents to leverage GR00T 2 for complex, long-horizon manipulation.

G.1. Agentic Skill Composition and Recovery with Code-as-Policy

Even after post-training, long-horizon manipulation remains vulnerable to compounding errors and unforeseen states (Ross et al., 2011). We introduce an agentic execution framework in which a frontier multimodal coding agent serves as System 2 (S2), composing GR00T 2 with CaP tools while monitoring task progress and recovering from failures. The agent verifies each intervention before resuming execution, without requiring GR00T 2 to learn every contingency.

The same framework supports complementary execution strategies. In simulation, we use ASPIRE (Lu et al., 2026) to discover task-level programs that call GR00T 2 as a manipulation tool, and each task runs either its program or GR00T 2 alone as the default executor, whichever succeeds more often on development seeds. On physical robots, where GR00T 2 is already capable of handling most manipulation, it remains the default controller. In both settings, S2 provides monitoring and CaP-based recovery. The key distinction is what executes between interventions.

Agentic Runtime Architecture. The runtime exposes GR00T 2 execution, robot observations, and CaP primitives through the Model Context Protocol (MCP). S2 uses these tools in an observe-reason-act-verify loop (Fig. 6).

The `wam_act` tool executes GR00T 2 with a task prompt and returns execution feedback. The agent can access camera observations and robot state through dedicated observation tools and adjust GR00T 2's language prompt during execution if needed. The `stop_wam` tool terminates GR00T 2 execution when a recovery action requires control. CaP action primitives issue end-effector and gripper commands through inverse kinematics and motion planning, while observation primitives use vision foundation models for information such as object localization and pose estimation. CaP programs compose these primitives into executable skills that can be added at test time without retraining GR00T 2. A single control worker coordinates actuation so that GR00T 2 and CaP do not issue conflicting robot commands.

The agent receives an operating guide (`AGENTS.md`), skill documentation, and task context describing stages, success criteria, and known failure modes. It checks observations and tool feedback for stalled progress or errors, then invokes a recovery action when needed. Recovery may involve regrasping a dropped part, reorienting a component, or returning the robot to a valid configuration before execution resumes.

Program and Skill Acquisition. ASPIRE learns the executable behaviors used alongside GR00T 2, but the scope differs by setting. In simulation, it discovers task-level CaP programs that compose GR00T 2 calls with perception, geometric reasoning, and manipulation primitives. During online execution, S2 may reuse or adapt code from these programs for recovery. On physical robots, ASPIRE instead acquires targeted CaP skills, either through direct robot interaction or through *CaP sim-to-real transfer*, in which skills are learned in simulation and adapted to real-world conditions (Sec. J.2.4).

Table 19: Comparison with the top seven entries of the RoboCasa365 and RoboDojo leaderboards, including agentic submissions. We report success rate (%) overall and per category. The RoboDojo average is the official mean over the five capability dimensions. Leaderboard numbers are as of November 6, 2026. Our RoboDojo results use layout bank 0, so the GR00T 2 SFT row differs slightly from the official three-bank evaluation of the same benchmark in Tab. 31 (35.5% average success). In the agentic rows, each task's default executor is selected on development seeds of the same benchmark, disjoint from the test seeds. The best and second-best results are highlighted in bold and underline, respectively.

RoboCasa365

Method	Type	Avg.	Atomic-Seen	Comp.-Seen	Comp.-Unseen
HexaShell 1.5	Agent	<u>85.6</u>	91.1	85.0	**80.0**
Awomo-0.5I (Awomo-WAM Team, 2026)	WAM	64.4	86.5	58.8	45.2
minionx	VLA	62.6	81.8	60.0	43.6
Awomo-0.5V (Awomo-WAM Team, 2026)	WAM	62.5	85.0	60.4	39.4
Axiom-0 (Shu et al., 2026)	VLA	61.6	86.5	59.4	35.8
Paimon-0	VLA	58.1	79.3	55.8	36.7
Xiaomi-Robotics-1	VLA	57.4	80.2	57.1	32.1
GR00T 2 - SFT (Ours)	WAM	67.2	87.4	73.6	38.2
GR00T 2 - SFT + RL (Ours)	WAM	72.4	<u>92.5</u>	81.5	40.6
GR00T 2 - SFT + Agent (Ours)	Agent + WAM	81.8	88.2	<u>86.9</u>	69.5
GR00T 2 - SFT + RL + Agent (Ours)	Agent + WAM	**88.0**	**95.6**	**92.0**	<u>75.5</u>

RoboDojo

Method	Type	Avg.	Gen.	Precision	Long-Hor.	Memory	Open
Q0-Preview	Agent	68.9	58.7	58.5	58.5	<u>93.7</u>	75.0
GO-3 Nano Preview	Agent + WAM	56.4	33.7	**76.0**	53.5	63.7	55.3
PyRUA-Lean	Agent	50.4	43.2	38.3	43.8	68.3	58.3
HexaShell-Preview	Agent	46.9	34.5	32.0	31.5	81.0	55.3
WorldScape-R^{2}SI	Agent + WAM	39.4	35.8	15.5	34.3	63.7	47.5
EmbodiedEvo	Agent	39.0	30.8	16.3	21.8	76.7	49.3
WALL-OSS-1.0-lite	VLA	35.0	37.9	29.1	32.3	56.0	19.6
GR00T 2 - SFT (Ours)	WAM	35.2	40.0	40.5	53.8	36.0	5.8
GR00T 2 - SFT + RL (Ours)	WAM	40.8	45.0	53.7	55.0	39.3	11.0
GR00T 2 - SFT + Agent (Ours)	Agent + WAM	<u>72.3</u>	<u>64.0</u>	62.5	<u>62.3</u>	<u>93.7</u>	<u>79.2</u>
GR00T 2 - SFT + RL + Agent (Ours)	Agent + WAM	**75.8**	**67.7**	<u>68.8</u>	**68.0**	**95.0**	**79.5**

Simulation Evaluation. We evaluate on RoboCasa365 (Nasiriany et al., 2026) and RoboDojo (Chen et al., 2026a) using four configurations: SFT, SFT + RL, SFT + Agent, and SFT + RL + Agent. For agentic execution, ASPIRE discovers programs using 15 development seeds per task. On 30 development seeds, including those 15, we then run each task's program and GR00T 2 alone, and make whichever succeeds more often the task's default executor, keeping the program on ties. With the SFT checkpoint, this selects GR00T 2 for 23 of the 50 RoboCasa365 tasks and 12 of the 54 RoboDojo task configurations. S2 starts every episode with the default executor, monitors it, and invokes online CaP recovery when needed. We test on 50 held-out seeds per task on both benchmarks (25 each for the standard and randomized variants of RoboDojo's generalization tasks), disjoint from the development seeds.

With SFT alone, GR00T 2 achieves the highest success rate of any non-agentic policy on both the RoboCasa365 and RoboDojo leaderboards (Sec. J.3 and Tab. 19). Memory- and reasoning-based tasks and high-precision manipulation still leave room for improvement.

RL post-training with AutoRL (Sec. F.2) mainly improves precision, raising RoboDojo Precision success from 40.5% to 53.7%, and lifts average success from 67.2% to 72.4% on RoboCasa365 and from 35.2% to 40.8% on RoboDojo layout bank 0 (Tab. 19). Agentic execution helps most on memory- and reasoning-focused tasks: ASPIRE programs and S2 raise success on RoboCasa365 Composite-Unseen tasks from 38.2% to 69.5% and on RoboDojo Memory tasks from 36.0% to 93.7%. Overall, it brings average success to 81.8% on RoboCasa365 and 72.3% on RoboDojo, and combining it with RL gives the best results on both benchmarks. Compared with the top entries of both public leaderboards, including agentic submissions, SFT + RL + Agent ranks first on RoboCasa365 (88.0% vs. 85.6% for HexaShell 1.5) and on RoboDojo, where we evaluate layout bank 0 only (75.8% vs. 68.9% for Q0-Preview) (Tab. 19).

Table 20: Episode outcomes by whether S2 intervenes (SFT + Agent). Each cell gives the number of episodes and how many of them succeed, over all episodes of the SFT + Agent runs (2,093 on RoboDojo and 2,500 on RoboCasa365).

	S2 leaves it alone	S2 intervenes
RoboDojo	1,320 (1,291 succeed)	773 (188 succeed)
RoboCasa365	1,512 (1,492 succeed)	988 (553 succeed)

To characterize online recovery in the SFT + Agent setting, we split episodes by whether S2 intervened (Tab. 20). S2 rarely lets a failure through: of the episodes it leaves untouched, only 29 of 1,320 fail on RoboDojo and 20 of 1,512 on RoboCasa365. Of the episodes it intervenes in, 24% succeed on RoboDojo (188 of 773) and 56% on RoboCasa365 (553 of 988). Fig. 53 shows one rescue and one failed recovery.

A manual review of failed SFT + Agent episodes on both benchmarks (Fig. 54) shows that most failures fall into two groups: the recovery requires manipulation too precise for S2 to perform with motion-planning tools (38% on RoboCasa365, 39% on RoboDojo), or the episode reaches a state that can no longer be recovered, such as an object on the floor or too little time left (24% and 38%). Most of the rest look complete but do not satisfy the benchmark's success check.

Real-World Long-Horizon Manipulation. We evaluate agentic execution with GR00T 2 as the default controller on multi-stage assembly and repeated precision insertion, examining how CaP recovery complements policy post-training.

Multi-stage assembly. We consider Toy Plane, Toy Tractor, and Hot Wheels assembly (Fig. 55). To stress-test robustness, each episode begins by dumping parts onto the table, producing clutter and varied object poses. The robot must assemble the parts in order, use a toy drill, and recover from execution errors. We compare SFT, SFT + DAgger (Ross et al., 2011), and SFT + DAgger + Agent.

The Toy Plane task assembles a fuselage, two wings, and two screws using a toy drill. It requires seating and fastening the bottom wing, turning the airplane over, and fastening the top wing, with bimanual handovers and accurate screw-hole alignment. Success requires both wings fastened and the drill set down. The Toy Tractor task mounts and fastens two small front wheels and two large rear wheels, turning the body over between sides and returning it upright at the end. The Hot Wheels task mounts the correct front and rear wheels and installs a correctly oriented hood, fastening the components with a drill.

Fig. 7a reports the results. DAgger improves Toy Plane Assembly, Toy Tractor Assembly, and Hot Wheels Assembly success from 69%, 51%, and 23% to 84%, 71%, and 78%, respectively. Adding agentic execution further increases success to 98%, 97%, and 97%. This improvement reflects the ability of S2 to detect and repair local execution failures while preserving GR00T 2's learned manipulation capabilities. Because every trial starts from parts dumped at random, these success rates measure robustness to unseen starting configurations and are not directly comparable with the post-training benchmark (Tab. 1a), whose trials start from arranged layouts. For Hot Wheels Assembly, for example, the benchmark reports 85% success from its fixed layout.

Repeated precision insertion. We test five consecutive cable and charger insertions using the precision skills in

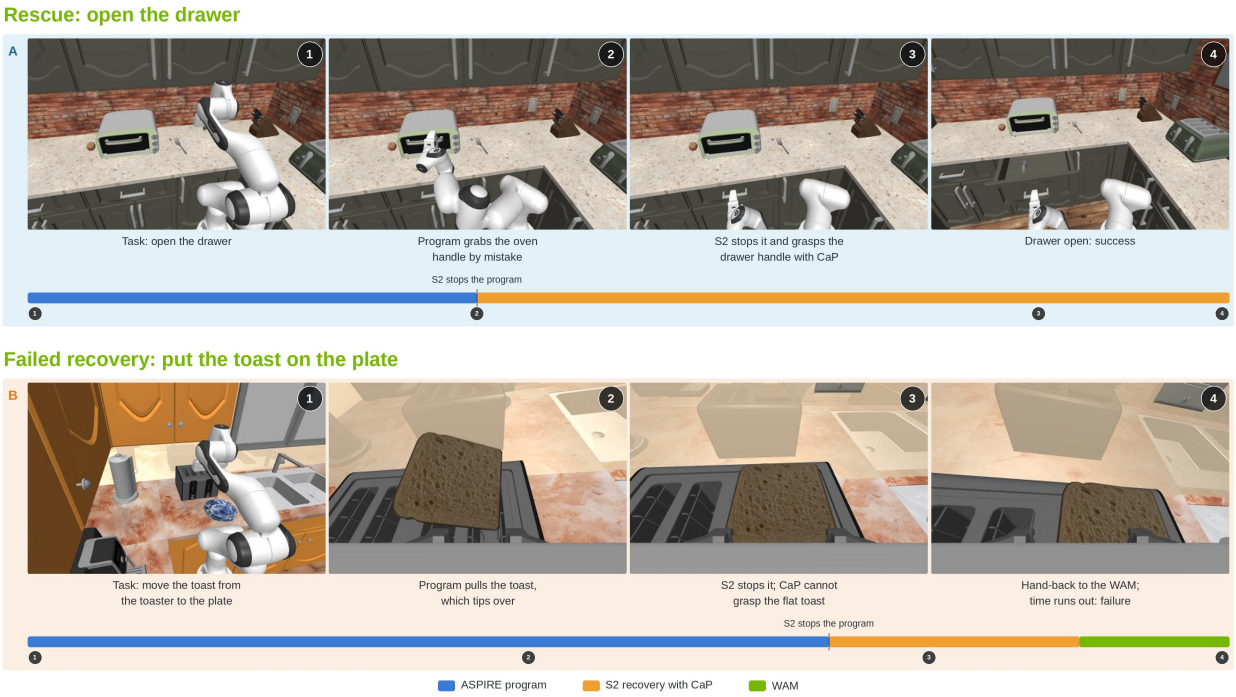


Figure 53: Online recovery on RoboCasa365 (SFT + Agent). Keyframes from two episodes. The bar below each row shows who controls the robot over the episode. (A) The ASPIRE program grabs the oven handle instead of the drawer handle. S2 stops it and grasps the drawer handle with CaP, and the drawer opens. (B) The program pulls the toast out of the toaster but it lands flat on top. S2 stops the program, its CaP grasp of the flat toast fails, and the hand-back to GR00T 2 runs out of time.

Sec. F.4, comparing SFT and RL-post-trained GR00T 2 within the same agentic system (Fig. 55).

For multi-cable insertion, five Ethernet (RJ45) cables stand upright beside a network switch. The robot must connect them in a fixed order to five designated ports. CaP first localizes all twelve candidate port faces using SAM 3 (Carion et al., 2025) and camera depth, then grasps each connector and transports it along a collision-free path to a pre-insertion pose. GR00T 2 performs the insertion, and S2 checks for an Ethernet link and verifies that the connector occupies the assigned port. If insertion stalls or exceeds 60 s, CaP retracts the connector and prepares another attempt, allowing up to three attempts per port. S2 can stop retries early if the connector has rotated in the gripper. The episode succeeds only if all five connectors are inserted, a setting that compounds insertion errors and introduces occlusion from previously inserted cables.

For multi-charger insertion, a human specifies five target outlets by pointing to them in order among twelve visually identical outlets arranged as a 2×6 grid. S2 grounds the gestures to individual outlets, with SAM 3 providing an independent interpretation. CaP grasps and transports each charger from its holder to the selected pre-insertion pose, and GR00T 2 performs the insertion. S2 checks whether the corresponding letter of an illuminated sign turns on and verifies the final target. Stalled or timed-out attempts can trigger CaP recovery, with at most three attempts per outlet and a 90 s limit per attempt. The task requires both accurate grounding of nonverbal instructions and precise insertion as the scene grows more cluttered.

Fig. 7b shows that RL post-training improves long-horizon insertion reliability within the same agentic system. For multi-cable insertion, success rises from 12/20 to 19/20, and for multi-charger insertion, it rises from 2/20 to 20/20. The gains highlight the value of RL-trained precision skills within long-horizon agentic execution.

Failure Analysis. Of the 300 SFT + DAgger + Agent assembly episodes, eight fail in three categories. *Incomplete fastening* accounts for three failures (two airplane, one Hot Wheels): a screw remains partially driven, but both

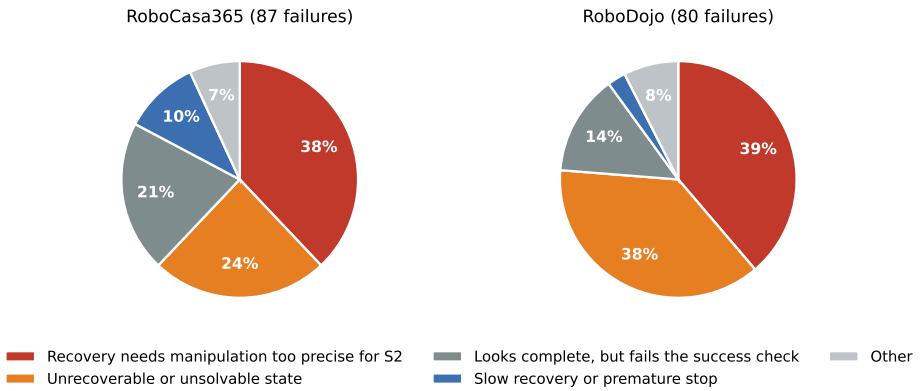


Figure 54: Why SFT + Agent episodes fail. Primary cause of manually reviewed failed SFT + Agent episodes: 87 on RoboCasa365 and 80 on RoboDojo.

GR00T 2 and S2 judge the step complete because the visual difference is subtle. *Late intervention* accounts for three failures (one tractor, two Hot Wheels): GR00T 2 mounts a small wheel on an incorrect rear hub before S2 stops it, and the available CaP skills cannot reliably reverse the operation. *Physical limitations* account for two tractor failures: one screw falls off the table, while another becomes mechanically jammed. Both of these episodes can be completed after brief human intervention, but are counted as failures under the autonomous evaluation protocol.

On repeated insertion, SFT + RL + Agent fails in only one of the 40 evaluated episodes, due to a timeout during multi-cable insertion. SAM 3 yields an imprecise connector grasp, leading to an offset pre-insertion pose. Although the RL-trained policy moves toward the port, alignment recovery requires small exploratory motions while maintaining forward pressure and supporting the connector against gravity, and the attempt exceeds the 60 s limit. This case illustrates how residual pose errors can still make recovery too slow under a fixed execution budget.

Together, these evaluations demonstrate two complementary forms of agentic execution. In simulation, ASPIRE-generated programs provide the task structure that direct GR00T 2 execution may lack, with online CaP recovery addressing unexpected states. In the real world, GR00T 2 performs most manipulation directly, while S2 detects failures and invokes targeted CaP skills, including skills acquired through CaP sim-to-real transfer. Both strategies extend GR00T 2's capabilities through programmable recovery without requiring every contingency to be absorbed into the foundation policy itself.

G.2. Agentic Steering with Visual Guidance

Agentic skill composition and recovery (Sec. G.1) integrates GR00T 2 with external CaP programs and skills, whereas agentic steering keeps GR00T 2 as the only controller and lets the agent shape its behavior through the WAM API. We first describe the three parts of visual steering, in which the agent adds visual guidance to its language instructions: the interface, the agent loop, and the mid-training of GR00T 2. We then evaluate visual steering on short- and long-horizon benchmarks on YAM and Dexter.

Visual Guidance Interface. The interface separates *what and where* from *how*. For each step, the agent names the acting arm and gives a grasp point and a release point as pixel coordinates in the image of one scene camera (Tab. 21), while GR00T 2 determines the grasp pose, the approach, and all motion in between. We observed that GR00T 2 conditions strongly on fine visual details in its conditioning frames, so we pass the points by drawing them as markers directly on the conditioning frame rather than through a separate input: a filled disc for the grasp point and a plus-shaped cross for the release point, in the acting arm's color (magenta for the left arm, cyan for the right), as in Fig. 8a. Only the acting arm is marked, and only on the current frame of the scene camera. Alongside the markers, GR00T 2 receives the agent's per-step instruction, which names the



Figure 55: Real-world tasks. (A) Toy Plane assembly: dump the parts, attach and fasten the bottom wing, turn the airplane over, and fasten the top wing. (B) Toy Tractor assembly: dump the parts, fasten the wheels on one side, turn the body over, and complete the other side. (C) Hot Wheels assembly: dump the parts, mount and fasten four wheels, and install and fasten the hood. (D) Multi-cable insertion: plug five Ethernet cables into a network switch. (E) Multi-charger insertion: plug five chargers into a power strip.

acting arm, in place of the task instruction.

Agent Loop. The agent loop has three layers. *System 2* is the agent, which plans the steps and judges whether each step is done and what comes next. *System 1* is GR00T 2, which executes the current step from the agent's instruction and points. The *harness* is the rule-based program between them, which passes the agent's instruction and points to GR00T 2 and decides when to query the agent. We use GPT-6 Astra as the agent, zero-shot and with one conversation per episode. At the start of an episode, it receives the task instruction and the images from all of the policy's cameras, plans the steps, and returns the acting arm and the two points for the first step. The harness queries the agent again when an arm releases an object, when a fast vision-language model, used as a completion checker reports the step done, or after 45 s without a step end. The agent then decides whether to advance to the next step, redo the current one, replan the remaining steps, or finish.

Mid-Training for Visual Guidance. Following the points requires mid-training, because pretraining never shows GR00T 2 markers on its observations. We generate the point labels automatically from existing demonstrations by detecting grasp and release events for each arm (Tab. 21). If an arm's next event is a grasp, its grasp marker is placed at that grasp and its release marker at the following release. If the arm already holds an object, the grasp marker stays at the preceding grasp and the release marker is placed at the next release. Each marker is the end-effector position at the event, projected into the scene camera. Markers are drawn for the arm that acts next, and an arm is never marked alone while the other arm is the one that acts next, so an unmarked arm next to a marked one should not act, as at evaluation. Each sample is conditioned on its annotated sub-task instruction, or on the task instruction where none exists.

Each point-conditioned checkpoint is mid-trained from the pretrained GR00T 2 checkpoint for 20k steps with AdamW (learning rate 5×10^{-5}, warmup-stable-decay schedule). For each, we also train a language-only checkpoint on the same data, cameras, schedule, and compute, without markers, which serves as the language-only baseline in the evaluation on YAM and on Dexter trash sorting.

Table 21: Embodiment-specific choices for visual steering. Everything not listed follows the shared rules in the text. The Dexter column describes the joint YAM and Dexter recipe used for Dexter trash sorting.

	YAM	**Dexter**
End effectors	Two parallel grippers	Two Sharpa five-fingered hands
Policy cameras	Top, two wrists, bottom	Head, two wrists
Marked camera	Top	Head
Grasp and release events	Gripper opening	Fingertip contact forces
Marker projection	Per-station calibration (in-lab data), nominal camera model (additional teleoperation data)	Nominal camera model
Mid-training data	Additional teleoperation (about 14k h, weight 0.7) and in-lab demonstrations (about 340 h, weight 0.3)	Joint with YAM: the YAM mixture (weight 0.71) and in-lab Dexter demonstrations (about 11.3k episodes, weight 0.29)
Global batch, GPUs	320, 32 GB300	448 (on average 320 YAM and 128 Dexter), 64 GB300
Action chunk	48 predicted actions (1.6 s at 30 Hz), of which the first 30 are executed per policy call	
Per-step instruction	"use the <arm> arm to …"	"… with the <arm> hand"

Evaluation Settings. Each real-robot benchmark compares three settings, named as in Fig. 8. *GR00T 2* is a language-only checkpoint given the original task instruction. *GR00T 2 + Agent (language)* is the same checkpoint given the agent's per-step instructions, from the same agent loop with a prompt that omits the points. *GR00T 2 + Agent (language + visual)* is a point-conditioned checkpoint given the agent's per-step instructions and markers. We refer to the last two settings as language guidance and visual guidance.

Short-Horizon Instruction Following. The short-horizon benchmarks test whether the guidance steers the policy. The YAM benchmark uses counterfactual instruction pairs: both instructions in a pair start from the same scene, which includes distractor objects, and differ only in the requested behavior, so a policy that repeats one plausible behavior regardless of the instruction can succeed on at most one instruction of each pair. It spans eight reference categories with three pairs each, giving 24 pairs and 48 instructions (Fig. 56). Six categories test grounding: *nominal* references identify objects or receptacles by name, category, or color; *spatial* references specify a position, relation, or distance; *ordinal* references select within an ordering, including relative size; *cardinal* references specify a number of objects or operations; *arm* references specify the acting arm; and *exclusion* references name an object that must be left untouched. Two categories test execution: *action* references distinguish operations such as opening versus closing, and *grasp* references constrain the approach or contact region. Each instruction is evaluated in three trials per setting (144 trials per setting). We report pass@1, the mean trial success, and pass@3, the fraction of instructions solved in at least one of their three trials.

The Dexter benchmark likewise separates reference types, without counterfactual pairs, in dexterous pick-and-place with colored cubes. It contains 32 tasks, each requiring the robot to move a specified cube into a specified basket with a permitted hand, grouped into four overlapping categories: attribute references (color, the Dexter counterpart of nominal references, 6 tasks), spatial references (12), ordinal references (7), and compositions of these references and constraints (9). A trial succeeds if the correct cube is released inside the specified basket by a permitted hand and remains stable for 2 s. Each task is evaluated in two trials per setting (64 trials per setting).

Long-Horizon Tasks. The long-horizon benchmarks require completing an entire multi-step task from a single instruction: trash sorting on both robots, and board disassembly on YAM as a stress test. In *trash sorting*, the robot places five items (on YAM: a glove, a crushed can, a plastic bottle, a sandwich, and a pepper; on Dexter:

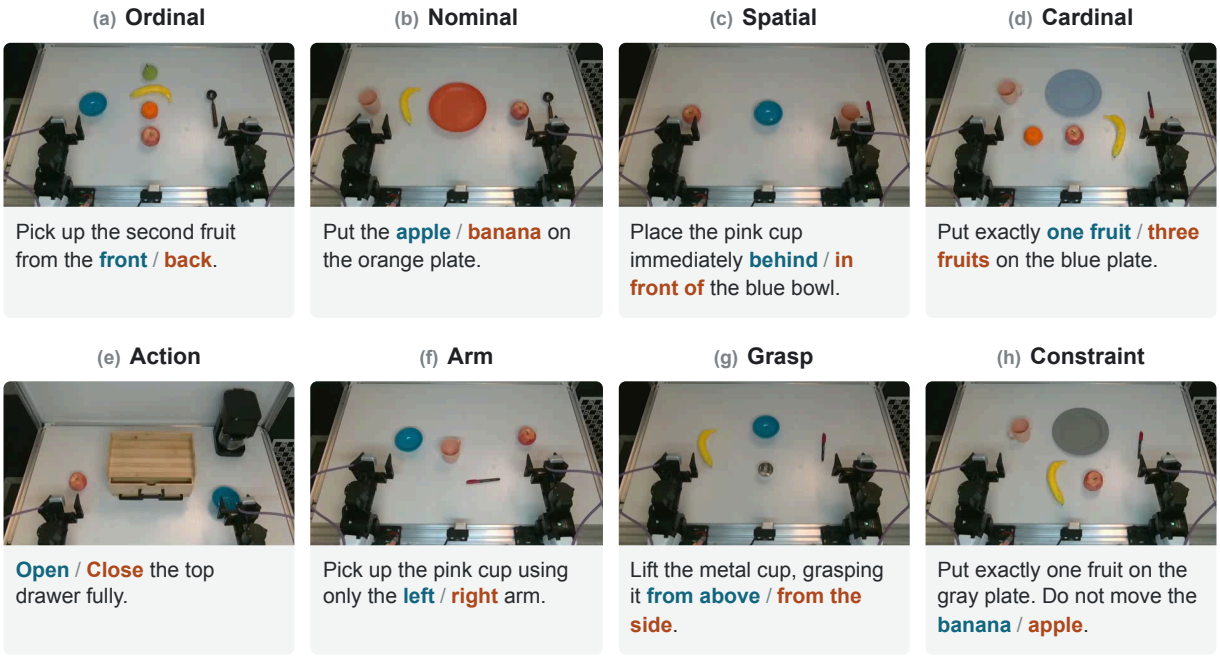


Figure 56: Short-horizon instruction following benchmark on YAM. Eight reference categories, each illustrated by a shared starting scene and two contrasting instructions. Blue and orange text denote the alternatives, and all remaining wording is identical. The panel labeled Constraint shows an exclusion reference.

a glove, a crushed can, a drink bottle, a sandwich, and a food scrap) into labeled compost, recycling, and trash bins whose order varies across 20 layouts per robot. Each item scores 0.2 if it ends in its correct bin. In *board disassembly* (Kimble et al., 2020), the robot removes the USB cable, two plugs, and two gears from a NIST task board and places the cable on a tray, the plugs in the right bin, and the gears in the left bin. Each item scores 0.1 for removal and 0.1 for correct placement. We report full-task success, in which all five items are completed, together with the mean rubric score. On YAM, the long-horizon runs were split between two stations with different operators, so the three settings were not all evaluated under the same station and operator.

Results. Tab. 22 lists every benchmark behind Fig. 8. On short-horizon tasks, visual guidance gives the highest success on both robots. On YAM, pass@1 rises from 41.7% for GR00T 2 and 55.6% with language guidance to 83.3% with visual guidance, and the gap narrows but persists at pass@3 (64.6%, 79.2%, and 89.6%). On Dexter, success rises from 29.7% and 32.8% to 70.3%. On long-horizon tasks, visual guidance completes 17 of 20 YAM trash-sorting episodes and 3 of 10 episodes of the board-disassembly stress test, whereas neither GR00T 2 nor language guidance completes any episode of either task. On Dexter trash sorting, visual guidance completes 9 of 20 episodes, against none for GR00T 2 and 1 with language guidance. Paired by layout, it sorts more items correctly than language guidance on 15 of the 20 layouts and fewer on 3 (two-sided sign test, $p = 0.008$).

The YAM category breakdown (Tab. 23) shows where visual guidance helps. It reaches 100% pass@1 on nominal, spatial, and exclusion references and 94.4% on ordinal references. Language guidance alone already helps on nominal and ordinal references (88.9% each) but barely on spatial references (from 50.0% to 55.6%), where the marked grasp or release point resolves the relation directly. Gains are smaller on the two execution categories: relative to GR00T 2, success rises from 55.6% to 72.2% on action references and from 27.8% to 55.6% on grasp references. The markers do not specify the grasp approach, object orientation, or the motion required for operations such as flipping, pouring, and scooping. On Dexter, visual guidance gains most over GR00T 2 on spatial, ordinal, and compositional tasks (50.0, 57.1, and 50.0 points) and little on attribute tasks

Table 22: Visual steering results on every benchmark. Short horizon: success rate (%), with successful trials in parentheses (pass@1 on YAM). Long horizon: full-task successes over episodes, with the mean rubric score (0 to 1) in parentheses. n counts trials (short horizon) or episodes (long horizon). The success rates and full-task counts are the components averaged in Fig. 8. †Obtained with an earlier implementation of the model.

Benchmark	n	GR00T 2	GR00T 2 + Agent (language)	GR00T 2 + Agent (language + visual)
YAM instruction following	144	41.7 (60)	55.6 (80)	**83.3** (120)
Dexter instruction following†	64	29.7 (19)	32.8 (21)	**70.3** (45)
YAM trash sorting	20	0/20 (0.20)	0/20 (0.46)	**17/20** (0.94)
YAM board disassembly (stress test)	10	0/10 (0.12)	0/10 (0.07)	**3/10** (0.46)
Dexter trash sorting	20	0/20 (0.40)	1/20 (0.46)	**9/20** (0.77)

Table 23: YAM short-horizon pass@1 (%) by reference category. Each category has six instructions and 18 scored trials per setting.

Reference category	GR00T 2	GR00T 2 + Agent (language)	GR00T 2 + Agent (language + visual)
Nominal	61.1	88.9	**100.0**
Spatial	50.0	55.6	**100.0**
Ordinal	33.3	88.9	**94.4**
Cardinal	27.8	38.9	**66.7**
Arm	50.0	50.0	**77.8**
Exclusion	27.8	55.6	**100.0**
Action	55.6	44.4	**72.2**
Grasp	27.8	22.2	**55.6**

(8.3 points), where GR00T 2 already reaches 75.0%.

Guidance Following. Success rates alone do not show that GR00T 2 follows the guidance, because a steerable policy should change its behavior when only the guidance changes (Chen et al., 2026d). The counterfactual pairs of the YAM short-horizon benchmark are designed to test a system-level version of this: within a pair, the scene is fixed and only the requested behavior changes. Counting a pair as solved if both of its instructions succeed in at least two of three trials, GR00T 2 solves 4 of the 24 pairs (16.7%), language guidance 10 (41.7%), and visual guidance 19 (79.2%). Conversely, 24 of the 60 successful trials of GR00T 2 (40.0%) come from instructions whose counterpart fails all three trials, as expected from a policy that repeats one behavior regardless of the instruction. This share falls to 12.5% (10 of 80) with language guidance and 5.8% (7 of 120) with visual guidance. On YAM trash sorting, GR00T 2 also follows the arm assignment: with visual guidance, the marked (and named) arm made 169 of 172 gripper closures (98.3%), against 164 of 181 (90.6%) for the named arm with language guidance, so wrong-arm closures fall from 9.4% to 1.7%. Its releases also follow the markers: with visual guidance, every release aimed at a bin landed in the marked bin, with one case unclear, whereas with language guidance, 21 items went into a wrong bin, or a different item was moved, after the agent had named the correct bin (see Failure Analysis).

Agent Cost and Latency. Steering adds the agent's cost and latency to every episode, which we measure on trash sorting on both robots (Tab. 24). With visual guidance, an episode takes a median of 9 agent turns and costs $0.26 on YAM and $0.14 on Dexter, and the robot waits for the agent during 38% and 32% of the episode. GR00T 2 was served without the accelerations of Sec. 4.6.1 on YAM (about 0.6 s per policy call) and with TensorRT on Dexter (about 0.14 s).

Detecting Step Completion. The agent's cost and the robot's waiting also depend on when the harness queries the agent at the end of a step. In the YAM long-horizon runs, gripper events, which need no model call,

Table 24: Agent cost and latency on trash sorting. Medians over the 20 episodes per setting. Latency is the per-episode median agent-turn time, agent cost is the charge for the agent's turns (excluding the completion checks), and waiting is the share of the episode during which the robot holds still for the agent.

Setting	Agent turns	Latency (s)	Agent cost (USD)	Waiting (%)	Episode (s)
YAM					
GR00T 2 + Agent (language)	10.5	7.4	0.35	29	306
GR00T 2 + Agent (language + visual)	9	7.5	0.26	38	186
Dexter					
GR00T 2 + Agent (language)	9	6.1	0.15	35	170
GR00T 2 + Agent (language + visual)	9	5.5	0.14	32	178

triggered 418 of the 435 step-end queries, and the 45 s limit triggered the remaining 17. The completion check is meant for steps that end without a release, such as closing a drawer, but replaying the logged requests shows that even an always-correct check would confirm a step a median of 3.9 to 6.7 s after it ends, against about 1.5 s for a gripper event. Polled throughout every step, the checks also account for 26 to 36% of the combined agent and check charge on YAM, so we keep the check as a backstop.

Failure Analysis. Failures on YAM trash sorting differ in kind between the two agent-guided settings. With visual guidance, 6 of 100 items are lost, in 3 episodes, and each of these episodes contains a failed hand-over, in which the agent asked one arm to stage an item at the center of the table for the other arm. By the agent's step-end observations, every release aimed at a bin landed in the marked bin, with one case unclear. Of the 54 items lost with language guidance, 21 were released into a wrong bin, or a different item was moved, after the agent had named the correct bin, and 20 were never attempted because the agent spent the remaining steps retrieving misplaced items. On Dexter trash sorting, visual guidance leaves 11 items in a wrong bin, against 43 with language guidance and 56 for GR00T 2, while a similar share of pick-and-place attempts ends in some bin in every setting (72%, 65%, and 71%), so the settings differ mainly in choosing the right bin.

Simulation Test of Steerability on LIBERO. Simulation lets us test steerability without the agent by computing the markers from the simulator state. We evaluate 45 tasks from LIBERO-PRO (Zhou et al., 2025), which perturbs the scenes and instructions of LIBERO (Liu et al., 2023), with 10 episodes per task (448 valid episodes), including spatial-instruction variants of a two-bowl scene on which an earlier language-only checkpoint failed, to test steering where language is weak. We compare a single-arm language-only checkpoint with a single-arm point-conditioned checkpoint, both mid-trained for 20k steps on RoboCasa365 (Nasiriany et al., 2026), LIBERO, and DROID (Khazatsky et al., 2024). This simulation checkpoint uses an earlier, richer form of the interface: a time-colored end-effector path with a disc at the next grasp or release point. The markers raise success from 69.6% (312 of 448) for the language-only checkpoint to 85.0% (381), while the same point-conditioned checkpoint with nothing drawn reaches 72.5% (325). On the two-bowl variants, the markers raise success from 18 of 70 for the language-only checkpoint (20 with nothing drawn) to 41 of 70. Across tasks, the marked checkpoint does better than the language-only checkpoint on 13, worse on 8, and ties on the other 24.

Limitations. Visual steering also has three limitations. The points specify spatial targets and the acting arm, not a manipulation strategy, which limits the gains on grasp and action references. Every step moves one arm, so coordinated two-arm motion is excluded, and a hand-over becomes two single-arm steps. On the real robots, generalization to new scenes and objects is not tested.

G.3. Toward Robotic Self-Improvement

Agentic learning, execution, and steering address complementary challenges in building autonomous robotic systems. AutoResearch investigates how learning algorithms can be improved through autonomous experimentation. CaP enables structured orchestration and specialized intervention during long-horizon execution.

Visual guidance allows a high-level reasoning model to steer the behavior of a foundation policy through richer interfaces beyond language.

A promising direction for future work is to integrate these capabilities into a unified self-improvement process. During deployment, agentic execution and steering could identify difficult states, generate corrective behaviors, and collect interaction experiences that expose limitations of the current foundation policy.

These experiences could be incorporated into AutoResearch to discover better post-training strategies, while successful agentic behaviors could be distilled into the foundation policy to improve its generality and reduce the need for future intervention.

Such integration would create a feedback loop in which agentic systems improve not only how robots execute tasks, but also how their underlying policies are trained and adapted.

Although realizing this fully integrated loop remains a direction for future work, these three approaches illustrate complementary opportunities for building robotic systems that can autonomously learn, orchestrate, and adapt their behavior.

H. Humanoid Loco-Manipulation

Loco-manipulation for bipedal humanoids poses additional challenges for robot foundation models, as humanoids must achieve precise end-effector control while maintaining balance under complex whole-body dynamics. Given these stringent requirements for reactive control, we turn to GR00T 2 Turbo, the lightweight variant of GR00T 2 suitable for deployment on edge devices (*e.g.* NVIDIA Jetson AGX Thor (Maheshwari et al., 2025)). We extend GR00T 2's two-level architecture with SONIC (Luo et al., 2026) to form a three-level hierarchy (Fig. 9). A reasoning model, *System 2* (S2), selects the task. Conditioned on this task and the current observations, GR00T 2 Turbo, *System 1* (S1), generates short-horizon whole-body motion commands in SONIC's compact latent action space. SONIC, *System 0* (S0), translates these motion commands into joint-level commands while maintaining balance. Both S1 and S0 run onboard the robot, coupling task-directed motion generation with reactive whole-body control.

Our investigation shows that GR00T 2 Turbo achieves high success rates across four task settings that highlight complementary aspects of whole-body loco-manipulation: scooping and serving popcorn, which requires precise bimanual tool use while standing; clearing diverse objects from a table into bins, which requires long-horizon coordination of walking, reaching, grasping, and carrying; and floor-level tasks, such as wiping spills with the feet and picking up objects while squatting or kneeling, which require coordinated upper- and lower-body motion.

All post-training data for whole-body loco-manipulation comes from whole-body teleoperation, which is expensive to collect and hard to scale. We therefore augment it with a large amount of synthetic data from a real-to-sim pipeline: teleoperated pick-and-place motions are fitted to the S0 controller and rolled out in simulation, so the synthetic demonstrations share the action space of the real data. GR00T 2 Turbo makes better use of this simulated data than the GR00T 1.7 VLA baseline, especially on out-of-distribution tabletop pick-and-place: starting from the same 6.7% success with real data alone, co-training raises GR00T 2 Turbo to 63.3%, compared with 40.0% for GR00T 1.7 (Tab. 26).

H.1. GR00T 2 Turbo on Whole-Body Latent Actions

The architecture of GR00T 2 Turbo is described in Sec. 2. To use GR00T 2 Turbo on a humanoid, which requires high-frequency control to remain balanced, we use the latent token interface of SONIC (Luo et al., 2026) as its action space. At a frequency of 1 Hz, GR00T 2 Turbo predicts action chunks containing the desired SONIC motion tokens and gripper width commands for the next 1.6 s. Each chunk is sampled at 50 Hz, so that at each 20 ms control step the SONIC controller receives the corresponding low-dimensional motion token together with the current robot state and outputs joint position targets for the humanoid. Gripper widths are not passed through the SONIC controller. The same token space is used for data collection, simulation, and deployment. In teleoperation, the operator's motion is encoded into tokens and executed by SONIC, so demonstrations are recorded directly in the action space of the policy. Simulated demonstrations are generated by running the same controller in simulation.

H.2. Real-World Whole-Body Loco-Manipulation

Tasks. We evaluate four task families that focus on different aspects of whole-body loco-manipulation (Fig. 57). In *popcorn serving*, the robot stands in front of a popcorn machine, takes a cup from a dispenser, scoops the popcorn, fills the cup, and places it for a customer. The task requires bimanual tool use and precise placement while standing. In *table clean-up*, objects such as bottles, food containers, fruit, and wrappers are spread over a table, and the robot picks them up one at a time and puts them into trash bins, walking between the table and the bins. This task tests walking, reaching across a table, grasping diverse objects, and carrying. The *ground pick-up* task asks the humanoid to pick up diverse objects from the ground by bending down. The *whole-body cleaning* task combines floor pick-up with cleaning a spill: the robot takes a towel from a counter, picks up a cup, drops the towel on the spill, wipes with its feet, and discards the towel and the cup into the bins. These

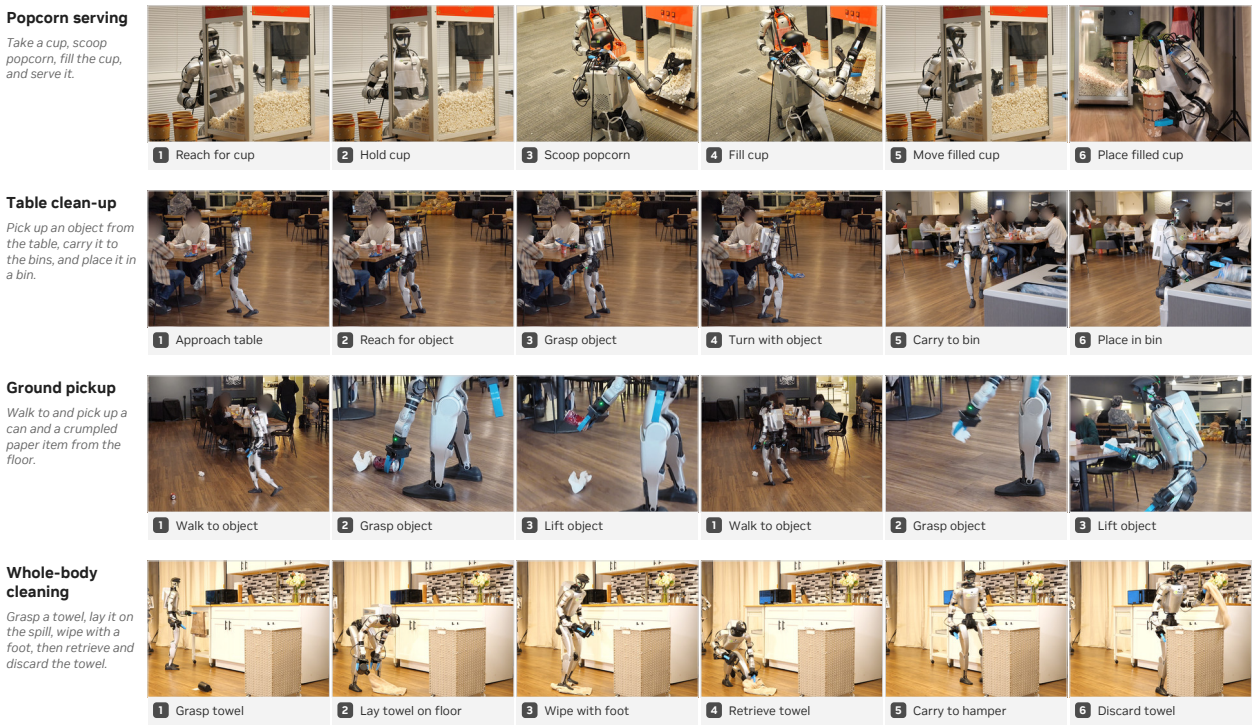


Figure 57: Autonomous real-world whole-body loco-manipulation. Representative task sequences executed autonomously by post-trained GR00T 2 Turbo policies, with time progressing from left to right. From top to bottom: *popcorn serving* requires precise bimanual tool use to scoop, fill, and place a cup; *table clean-up* combines reaching and grasping with walking and carrying objects to bins; *ground pick-up* requires reaching to the floor and standing up with an object before carrying it away; and *whole-body cleaning* coordinates towel handling, foot-based wiping, and kneeling to retrieve the towel for disposal.

tasks require deep reaches and *coordinated use of the feet and arms*, and they exercise the full capabilities of a bipedal humanoid. In addition, to study real and sim co-training, we use a *tabletop pick-and-place* task in which the humanoid starts at different positions relative to the table and picks up one of 30 diverse objects. The objects used for the table clean-up and tabletop pick-and-place tasks are shown in Fig. 58.

Data collection. We train our policies on demonstrations collected via whole-body teleoperation and on synthetic demonstrations generated in simulation. During teleoperation, an operator wears a headset and body trackers. The operator's motion is encoded into SONIC tokens, executed on the robot, and recorded together with the camera images. For the popcorn task, we collected 1,500 episodes for a total of 17 hours of data. Table clean-up was recorded as continuous episodes in which the operator clears several objects in a row. The robot starts facing the table at a random heading, picks up a piece of trash from the table, and then puts the object into the trash bin. For this task, we collected 3,995 episodes: 1,105 general, 375 room, and 2,515 ground episodes. For ground pick-up and tabletop pick-up, we collected 3,402 and 3,292 episodes, respectively. For the spill clean-up task, we used 292 episodes, totaling 3 hours of data. Every episode is reviewed and scored, and only episodes that pass review are used.

Training. Popcorn and spill clean-up are post-trained as single-task policies on their own demonstrations. Pick-and-place is post-trained as a multi-task policy that mixes tabletop pick-up, ground pick-up, placing into bins, and the clean-up segments, with a crosshair indicating the target. Object targets for the crosshair are labeled once per segment with a vision-language model and propagated through the video with a point tracker. For the comparison with GR00T 1.7, $\pi_{0.5}$, and a diffusion policy (DP), we post-train each model on the same demonstrations with the same token action space and the same cameras.



Figure 58: Object sets for the tabletop pick-and-place and the table clean-up tasks.

Table 25: Real-world whole-body loco-manipulation on the Unitree G1. Success rate (%) using each listed model as S1, with the SONIC controller as S0. No partial credit is given (the reported success rate is the sum of the binary scores over all trials divided by the total number of trials). All policies are post-trained on the same demonstrations per task. DP: diffusion policy on latent tokens; $\pi_{0.5}$ (VLA); GR00T 1.7 (VLA); GR00T 2 Turbo (WAM).

Task	Trials	DP	$\pi_{0.5}$	GR00T 1.7	GR00T 2 Turbo
Popcorn scooping and serving	48	27.1	0.0	83.3	**100.0**
Table clean-up (diverse objects)	30	0.0	0.0	73.3	**93.3**
Ground pick-up (diverse objects)	30	36.7	30.0	50.0	**83.3**
Spill clean-up (whole body)	30	10.0	0.0	43.3	**46.7**
Average		18.5	7.5	62.5	**80.8**

Evaluation and results. Popcorn serving is scored on a fixed suite of 48 runs, each requiring two full cups to be scooped and placed. The table clean-up task is scored on 30 trials; each trial involves picking up 6 diverse objects and putting all of them into the bin successfully. Failing to pick up an object or to drop it into the trash bin results in failure. Tabletop pick-and-place is scored on 30 trials with 30 household objects, 25 rigid and 5 soft, placed at ten positions on tables of three heights, with a 30 s limit per trial. Ground pick-up is scored on the same 30 household objects placed on the ground. For the spill clean-up task, we measure success over 30 trials of the full sequence: grabbing the towel, picking up the cup, wiping the spill with the feet, and dropping the towel and the cup into the bins. Tab. 25 reports the results. GR00T 2 Turbo completes 48 of 48 popcorn runs at about 41 s per two cups, while GR00T 1.7 trained on the same data completes 83% at about 48 s per two cups and is the best-performing baseline on this task. On table clean-up, GR00T 2 Turbo succeeds in 28 of 30 trials. Failures are flat objects such as a napkin or a chips bag that the gripper misses on, and objects knocked out of reach. Ground pick-up succeeds in 25 of 30 trials with the same failure modes. On the whole-body spill clean-up task, GR00T 2 Turbo succeeds in 14 of 30 trials, and GR00T 1.7 is again the second-best model with 13 of 30.

H.3. WAMs Consume Simulation Data Better than VLAs

Motion-centric Real-to-Sim Demonstrations. Real-world teleoperation data is costly to collect and limited in scene and object diversity. Our key insight is to reuse the whole-body motions recorded on the G1 by finding simulated scenes in which those motions successfully perform the task. Since teleoperated episodes contain motion and camera observations but not exact table and object poses, we initialize their simulated placements from the head-camera images and refine them using CMA-ES (Hansen et al., 2003). In MuJoCo (Todorov et al., 2012) or Isaac Lab (Mittal et al., 2025), we replay each motion through the same SONIC controller used on the

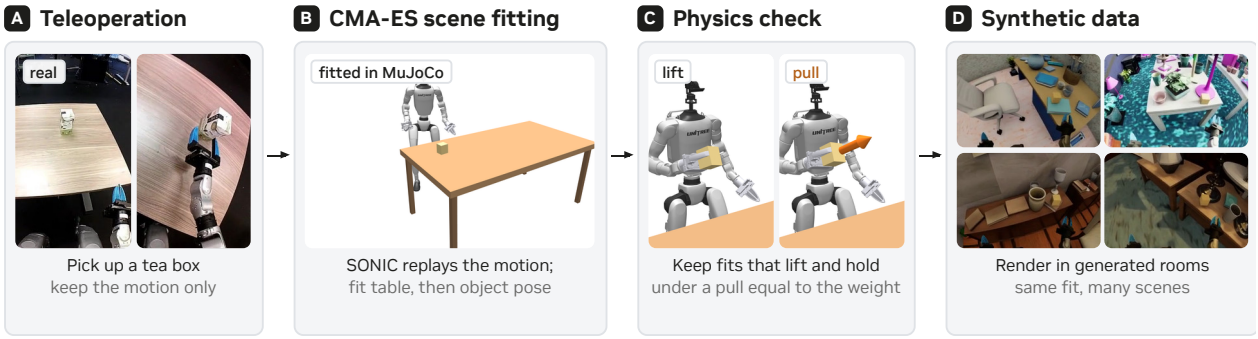


Figure 59: Motion-centric simulation data generation. (A) We use the recorded whole-body motion of a teleoperated episode. (B) SONIC replays the motion in simulation while CMA-ES fits the table and object placement. (C) A fit is kept only if the robot lifts the object and holds it under a pull equal to its weight. (D) The same accepted fit is rendered in many generated rooms.

Table 26: WAMs consume simulation data better than VLAs. Tabletop pick-and-place success rate (%) over 30 trials on the G1 when simulated demonstrations are mixed into the same real dataset.

	In-Distribution		Out-of-Distribution	
Policy	Real	Real + Sim	Real	Real + Sim
$\pi_{0.5}$ (VLA)	40.0	16.7	0.0	13.3
GR00T 1.7 (VLA)	46.7	66.7	6.7	40.0
GR00T 2 Turbo (WAM)	73.3	**83.3**	6.7	**63.3**

real robot, keeping the recorded motion fixed while optimizing the scene. The search evaluates 48 candidate scenes in parallel per generation and proceeds in two stages: fitting the table position, height, and yaw; then fitting the object type, position, and orientation on the table. A fit is accepted only if the final rollout grasps the object with both fingers, lifts it, and holds it through a 30-frame disturbance force equal to the object's weight, followed by a 100-frame hold. We then render these fitted trajectories in procedurally generated environments with randomized lighting and camera placement, recording token sequences and rendered images in the same format as the real demonstrations. The resulting synthetic demonstrations preserve the real-data action space while expanding visual and object diversity. We use this procedure to generate 250,000 synthetic episodes totaling 1,000 hours. The generation procedure and examples are shown in Fig. 59.

Results. In Tab. 26, we test each policy in distribution, in the environment where the real data were collected, and out of distribution, in unseen scenes and locations. The synthetic data improve the pick-and-place success rate of GR00T 1.7 and GR00T 2 Turbo in both settings, while $\pi_{0.5}$ degrades in distribution. The gain is largest out of distribution, where both GR00T 1.7 and GR00T 2 Turbo start from 6.7% with real data alone. The WAM benefits more from the simulated data in this setting: co-training removes 61% of its failures, compared with 36% for GR00T 1.7, a difference of 25 percentage points. With 30 trials per cell, however, the difference between the two co-trained models (19 against 12 successes) is suggestive rather than conclusive (two-sided Fisher exact test, $p \approx 0.12$).

Why. We hypothesize that this comes from the video prediction objective: the WAM's more general video encoder maps rendered and camera frames into a shared latent space in which the model predicts how the scene evolves under its actions, a correspondence that is identical in simulation and reality, whereas a VLA maps appearance features tuned to real images directly to actions and treats rendered frames as out-of-distribution.

I. Accelerating GR00T 2

Practical deployment of a generalist robotics foundation model requires timely predictions, continuous motion, and efficient task completion across tasks and embodiments. Deployment on edge hardware also imposes compute, memory, power, and thermal limits. We address these requirements through inference acceleration, training-time real-time chunking (RTC) (Black et al., 2025), and phase-conditioned action chunk retiming. We evaluate latency, task outcomes, and motion quality together.

We first reduce inference latency within the deployment resource budget. WAMs in our setting incur higher inference latency than their VLA counterparts, motivating policy inference acceleration and system-level optimizations. Visual encoding and iterative video-action denoising contribute to model prediction latency, while observation processing and communication add delay around the model. We optimize both paths and distinguish model prediction latency from observation-to-applied-action latency. However, some delay remains even after optimization. Waiting for each prediction can interrupt execution, while naively executing predictions asynchronously can introduce discontinuities between successive action chunks. We therefore use training-time real-time chunking (RTC) to account for actions committed during inference and train the policy to produce compatible continuations.

Continuity must also preserve the policy's ability to correct its behavior using new observations. In our setting, standard training-time RTC improves continuity but may degrade task performance. We address this tradeoff with noisy-prefix, action-only RTC. Perturbing the conditioning prefix is intended to reduce reliance on the previous plan while retaining context for a smooth continuation. We further find that a small-scale fine-tuning stage applied to the post-trained base model captures most of the action-continuity benefits, making RTC adaptation practical without repeating the full training pipeline.

Finally, continuous execution and low inference latency do not necessarily yield shorter task completion times. We introduce phase-dependent action-chunk acceleration to increase execution speed where the task permits it while retaining finer timing for precise interactions. Accelerated execution also changes the temporal relationship between successive chunks, requiring the continuity mechanism to account for the revised execution timeline. Temporal chunk alignment addresses this timing mismatch, while cross-chunk jerk optimization smooths the transition between successive commands. We evaluate the combined approach through inference latency, motion quality, task completion time, and success rate to establish whether faster execution preserves reliable task performance.

I.1. Accelerating Model Inference

Inference uses a client-server architecture: the robot client sends observations and executes returned action chunks through its controller, while the server runs the policy (Sec. K.5).

GR00T 2 predicts action chunks by encoding camera observations, robot state, and task instructions, then iteratively denoising video and action latents. This computation adds delay to the control loop, while observation processing and communication introduce further overhead. We reduce these costs through computation reuse, optimized GPU execution, and a serving path that supports different embodiments and deployment hardware.

Timing boundaries. The environment continues to change while the policy computes, so prediction time is only part of the delay before a robot responds. We distinguish two intervals (Fig. 60). *Model prediction latency* covers encoding and iterative denoising performed within a prediction request. *Observation-to-applied-action latency* extends from observation preparation to controller application, including serialization, transport, and response handling. Streaming and prefetch can move encoding before the prediction request. We identify these configurations explicitly because their foreground prediction time excludes that encoding work.

Optimizing model prediction. We first remove repeated computation. The policy reuses text features while the task instruction and model remain unchanged, and refreshes visual and state representations for new observations. For a fixed checkpoint and denoising schedule, timestep-dependent conditioning values are

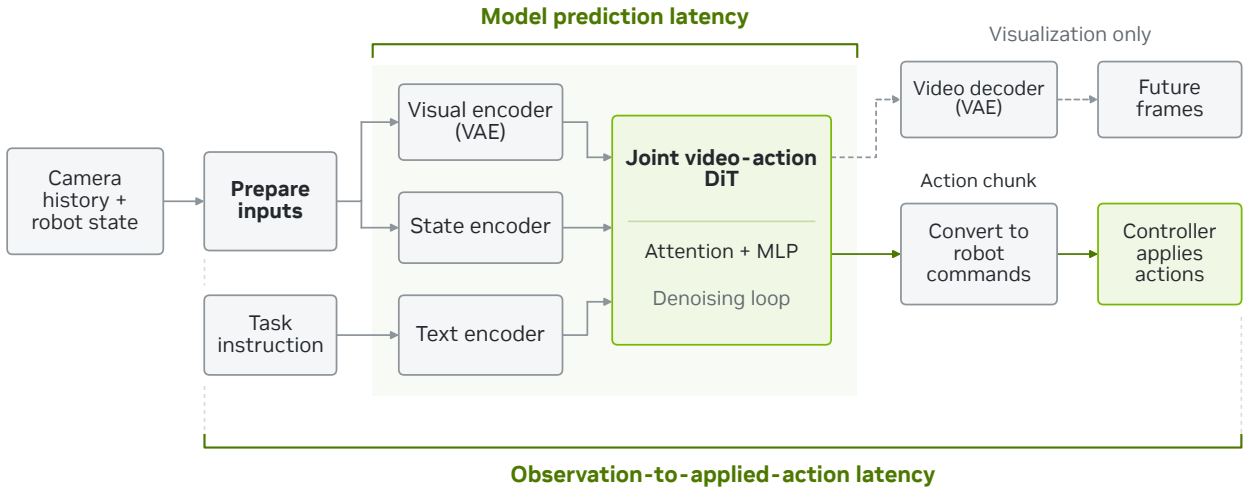


Figure 60: Timing boundaries for GR00T 2 policy inference. Model prediction covers request-time encoding and iterative denoising. Observation-to-applied-action latency additionally includes observation preparation, communication, response handling, and controller scheduling. The diagram is schematic, and stage widths do not represent measured durations.

computed in advance. Within the denoising loop, action-only updates reuse attention keys and values from the most recent video update. Persistent buffers reduce repeated cache preparation and copying.

We then accelerate the remaining computation. TensorRT compiles the video VAE encoder and transformer into specialized GPU engines, and CUDA graph replay reduces CPU dispatch and kernel-launch overhead. For supported checkpoints, we calibrate the BF16-trained model on representative robot observations and apply FP8 to selected projection and feed-forward layers in the joint transformer. Normalization layers and the action-only engine retain higher precision. Supported recipes also use FP8 for VAE encoding. Reducing denoising steps provides an additional tradeoff between latency, task success, and motion smoothness, which we assess for each checkpoint and task.

Fig. 19 compares measured model-prediction latency for a GR00T 2 Turbo checkpoint deployed on G1 using RTX 5090 and Jetson Thor.

Tab. 27 reports median prediction latency for GR00T 2 and GR00T 2 Turbo across three embodiments and the evaluated hardware. Each baseline-optimized pair uses the same underlying checkpoint and input observations. The optimized configurations provide $2.94\times$ to $7.15\times$ speedup relative to eager BF16 execution. Each speedup applies to the complete configuration: the G1 comparison includes fewer denoising steps, while optimized Dexter and YAM use prefetched VAE representations. The timing boundaries and execution recipes are specified alongside the table.

We use one checkpoint per embodiment across hardware comparisons. All three use the 2B model configuration, with five history frames per camera at 224×320 pixels (height $\times$ width). Dexter uses three camera views in four model slots and a 48-action prediction horizon, YAM uses four views and a 48-action horizon, and the GR00T 2 Turbo checkpoint for G1 uses three views and an 80-action horizon. The notation 2v4a denotes two video and four action denoising steps. Dexter uses 2v2a and YAM uses 2v4a in both baseline and optimized execution. G1 changes from 2v4a to 1v2a, with the action horizon unchanged.

All optimized configurations use TensorRT and CUDA graphs. Dexter uses FP16-mixed transformer engines with BF16 streaming VAE and prefetch, without codec roundtrip. G1 uses FP8 mixed precision, with VAE encoding included in prediction latency. YAM uses FP8 joint-transformer and VAE engines, a BF16 action engine, and whole-window VAE prefetch with MP4V. For Dexter and YAM, eager timings include VAE encoding,

while optimized timings exclude encoding completed before the request.

Table 27: Median model-prediction latency (p50) for GR00T 2 and GR00T 2 Turbo across embodiments and hardware. Speedup is baseline latency divided by optimized latency.

Embodiment	Model	Hardware	Prediction baseline (ms)	Prediction optimized (ms)	Speedup
Dexter	GR00T 2	GeForce RTX 5090	449.20	**152.78**	**2.94×**
	GR00T 2	DGX Station GB300	219.02	**48.47**	**4.52×**
G1	GR00T 2 Turbo	GeForce RTX 5090	224.73	**38.40**	**5.85×**
	GR00T 2 Turbo	Jetson Thor (120 W)	720.41	**123.97**	**5.81×**
YAM	GR00T 2	GeForce RTX 5090	550.96	**179.79**	**3.06×**
	GR00T 2	DGX Station GB300	386.06	**54.00**	**7.15×**

We evaluate embodiment-specific checkpoints. Each baseline-optimized pair uses the same underlying checkpoint and input profile. Baselines use unquantized BF16 PyTorch execution with checkpoint-default attention kernels. Baseline revisions are selected per checkpoint to preserve its required model behavior.

Optimizing the serving path. Multi-camera observations must be encoded, transferred, and prepared before the model can use them. We reduce this overhead with bounded CPU worker pools for camera-history processing, faster JPEG encoding and decoding, and fewer memory copies. Where supported, incremental streaming VAE reuses cached encoder state as camera frames arrive. VAE prefetch prepares the required visual latents before the next prediction request, moving encoding off the foreground prediction path. The benefit depends on the observation format, client implementation, and hardware. Deployment-level evaluation must also account for the age of the observations and the encoding work performed before the request.

Latency and action fidelity. Approximation-based inference acceleration can reduce latency at the cost of task success, our primary measure of robot policy quality. We therefore assess the effects of acceleration through numerical validation, simulation, and physical-robot evaluation. Numerical comparisons use the same checkpoint's eager BF16 execution as a reference, holding observations, conditioning, denoising schedules, and initial noise fixed. We compute mean squared error over valid, newly generated actions, excluding padding and committed RTC prefixes, and inspect decoded errors by action group. Simulation and physical-robot tests evaluate whether accelerated execution preserves task success, alongside completion time and motion smoothness.

During the remaining prediction delay, the robot must continue executing actions. The following subsection describes how training-time RTC conditions new predictions on actions already committed for execution, supporting continuity between action chunks.

I.2. Policy Smoothness

To account for inference latency, we build on training-time real-time chunking (RTC) (Black et al., 2025), which conditions each prediction on the committed action prefix from the previous prediction. We propose a simple extension of training-time RTC that we term *noisy-prefix RTC*. During fine-tuning, we perturb the conditioning action prefix with Gaussian noise. At inference, we condition on clean committed commands.

Simulation experiments on training-time RTC schemes. We evaluate task performance and action smoothness for GR00T 2 under fixed simulated inference delays, comparing RTC-conditioned execution with naive asynchronous execution.

A simple noisy-prefix RTC adaptation. We initialize from a GR00T 2 policy post-trained for 30k steps on RoboDojo (Chen et al., 2026a) demonstrations without RTC. Our proposed adaptation applies 5k additional fine-tuning steps on 800 demonstrations spanning eight precision tasks, adding Gaussian noise with $\sigma = 0.1$ to the conditioning action prefix during training. This trains the policy to generate continuations from perturbed prefixes. We compare against clean-prefix training-time RTC, which uses no prefix noise during fine-tuning.

Table 28: Evaluation of a simple noisy-prefix RTC extension under simulated inference latency. Success and progress use 1,200 matched episodes per condition, while jerk uses 120 matched diagnostic layouts. Jerk changes are relative to async execution of the noisy-prefix fine-tuned checkpoint at the same delay. Negative changes indicate lower jerk. Bold values indicate the best result within each delay setting.

Method	Success ↑	Progress /100 ↑	Δ p95 jerk ↓	Δ peak jerk ↓
Four-command delay: 160 ms				
Naive async	32.83%	42.03	0.0%	0.0%
Clean-prefix RTC fine-tuning	30.67%	39.48	**−31.1%**	**−80.2%**
Noisy-prefix RTC (ours)	**34.75%**	**43.16**	−6.5%	−69.2%
Eight-command delay: 320 ms				
Naive async	15.00%	22.06	0.0%	0.0%
Clean-prefix RTC fine-tuning	**21.42%**	**29.81**	**−87.8%**	−16.3%
Noisy-prefix RTC (ours)	**21.42%**	29.64	−87.2%	**−22.5%**

Both variants use clean committed prefixes during RTC inference. The naive async reference uses the *same noisy-prefix fine-tuned checkpoint* as the noisy RTC method, with prefix conditioning disabled at inference.

Evaluation protocol. All conditions replan every eight commands at 25 Hz. Fixed delays of four and eight commands correspond to 160 ms and 320 ms, respectively. Delay is modeled through the action queue, without adding wall-clock sleeps to the simulator. Success rate and mean task progress are measured over 1,200 matched episodes per condition, comprising 150 layouts for each task. Jerk is measured in a separate diagnostic using 120 matched layouts per condition, with 15 layouts per task distributed across three layout banks.

We compute commanded joint jerk using third finite differences at $\Delta t = 0.04\,\text{s}$, excluding grippers and startup. Cross-chunk jerk includes all three four-sample stencils crossing each old-to-new prediction handoff. The p95 statistic gives equal weight to each episode. Peak jerk is the largest observed absolute sample. At an eight-command delay, the prediction handoff coincides with the next replanning boundary.

Results. Table 28 reports performance and commanded-action smoothness relative to naive async execution. With identical noisy-prefix fine-tuned weights, RTC increases observed success from 32.83% to 34.75% at a four-command delay, while reducing p95 cross-chunk commanded jerk by 6.5% and peak jerk by 69.2%. At an eight-command delay, success increases from 15.00% to 21.42%, with reductions of 87.2% in p95 jerk and 22.5% in peak jerk.

Clean-prefix fine-tuning produces smoother commands at the four-command delay, but achieves lower task success than noisy-prefix fine-tuning. At the eight-command delay, the two RTC variants achieve the same observed success rate. The clean-prefix variant has lower p95 commanded jerk, whereas the noisy-prefix variant has lower peak commanded jerk. For simulated joint states at the eight-command delay, noisy RTC reduces p95 jerk by 40.1% but increases the observed peak by 13.4% relative to async execution.

I.3. Accelerating Task Throughput Beyond Demonstrations

We leverage an agentic task-throughput acceleration algorithm to enable GR00T 2 to complete tasks with higher task-completion throughput than the demonstration dataset. We decompose overall task execution into semantically meaningful phases, such as reaching, grasping, transporting, and screwing, which differ in their sensitivity to acceleration. An agent searches for a speed schedule over these phases by evaluating accelerated rollouts and refining the speed schedule based on observed failures. At deployment, a lightweight observation-conditioned future subtask predictor predicts a sequence of future phases aligned with GR00T 2's action-chunk horizon. Given the action chunk, predicted phases and a speed schedule, a trajectory transformation function produces a new action chunk for execution (see Fig 20). We train the predictor using temporally aligned phase annotations generated by an agent.

Evaluation Tasks. We showcase GR00T 2-Accelerated across three tasks on the Tianji Sharpa robot from the Dexter post-training benchmark task (see Section A.1). We report task progress scores, success rates and mean completion times of successful episodes in Table 29. Our experiments indicate that GR00T 2-Accelerated consistently decreases task completion times while preserving success rates across the evaluated long-horizon dexterous tasks.

Table 29: Task success rate and completion time on Dexter post-training tasks. Completion time is averaged over successful trials only.

Metric	Method	Make Shipping Label	Flashlight Battery Replacement	Crude Salt Filtering
Task progress Success rate (%)	GR00T 2	0.78 | 60%	0.86 | 55%	0.98 | 85%
	GR00T 2-Accelerated	0.81 | 65%	0.93 | 60%	0.94 | 80%
Mean completion time (s)	GR00T 2	95	80	184
	GR00T 2-Accelerated	81	66	137

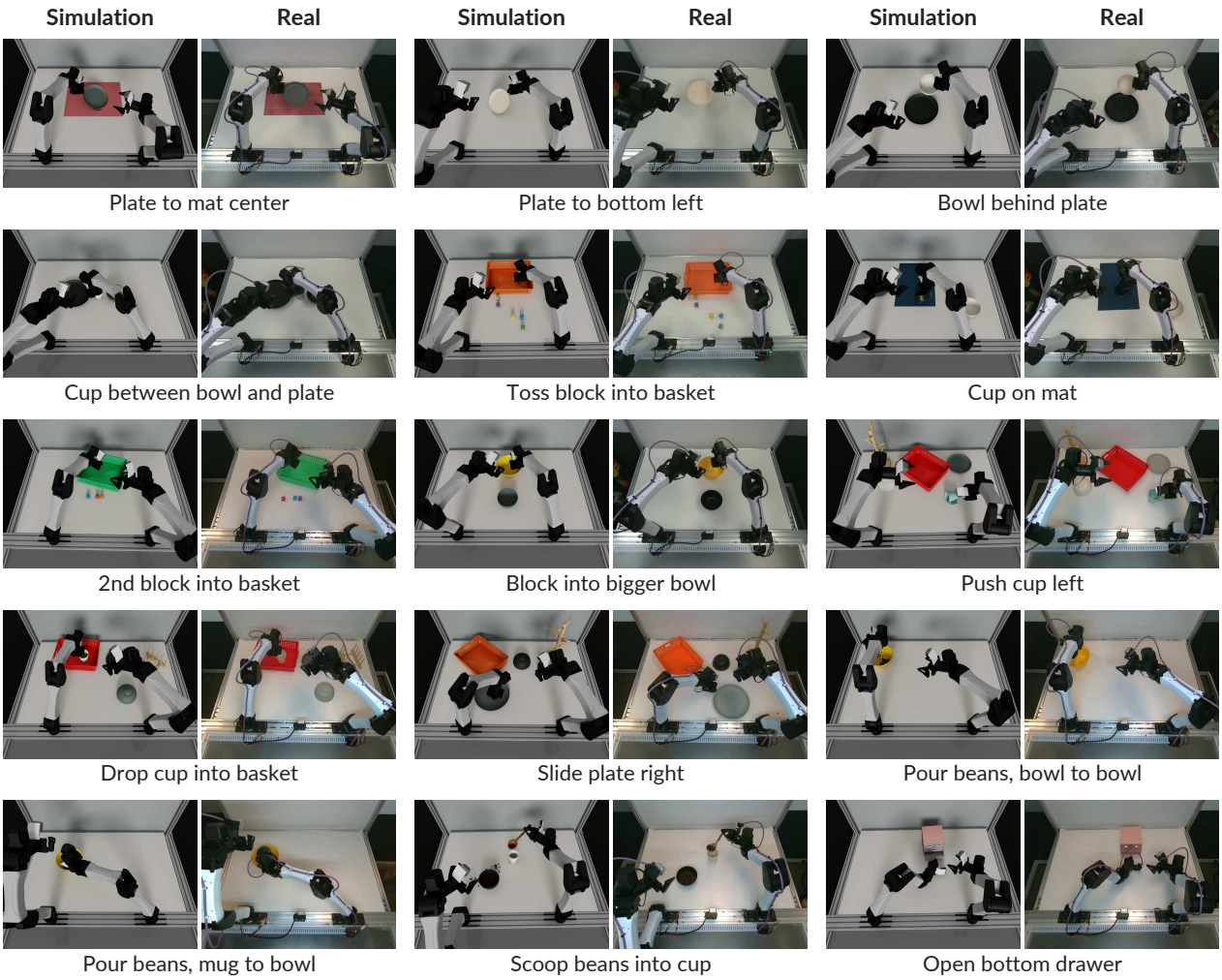


Figure 61: Subset of environments for Real-to-Sim Evaluation. Side-by-side of real-world tasks that are used to evaluate generalist capabilities of GR00T 2 and their simulation counterparts. Odd columns are the simulation environments and even columns are frames from the real-world robot workcell. Each task is evaluated five times in the real-world and ten times in simulation per checkpoint, with the results used to calculate the correlations between performance in simulation and the real-world.

J. Simulations

In this section, we outline how simulation contributed to GR00T 2. We developed a real-to-sim workflow that recreates high-fidelity digital twins of real-world tasks used for evaluating GR00T 2 (Sec. J.1). Policy evaluations in these high-fidelity digital twins proved to have strong correlations with real-world evaluations, allowing them to accelerate GR00T 2 development (Sec. J.2). Finally, we showcase GR00T 2's capabilities on standardized simulation environments, establishing new state-of-the-art results on several of the latest benchmarks (Sec. J.3).

J.1. Real-to-Sim Generation of High-Fidelity Tasks

J.1.1. GR00T 2 Simulation Task Suite

We seek to recreate the real-world tasks used in GR00T 2 development, including general manipulation tasks that are evaluated zero-shot and multi-stage assembly tasks. The task set spans rigid and articulated objects, deformable materials, and particles with diverse interactions including, but not limited to, sliding, pushing and scooping. We aim to reproduce both the visual setup and the physical interactions that determine whether

each task succeeds, so that the resulting environments are useful for policy evaluation and skill development. This modeling can be challenging for tasks involving complex interactions with particles or deformables, since physics stability and fidelity must be maintained while running fast enough for large-scale evaluation. To enable this, we agentically integrate and tune different solvers such as the Finite Element Method (FEM) (Sifakis and Barbic, 2012) for deformable bodies, Position-Based Dynamics (PBD) (Müller et al., 2007) for granular media, in addition to rigid-body and articulation dynamics. Contact-rich assembly provides another demanding example of real-to-sim reconstruction. Parts must fit together, remain attached during manipulation, and interact with robot grippers and tools. We reconstruct toy plane, Hot Wheels, and toy tractor assembly assets with separate components, fasteners, and handheld drills (Fig. 62). Their initial and assembled configurations illustrate the component structure and geometric relationships that these environments must represent.

J.1.2. Agent-Guided Simulation Creation

We use complementary agent-assisted workflows to construct these environments. Tools from SimFoundry (Ranawaka et al., 2026) use real-world robot videos to guide scene reconstruction and refinement. We also use scans, photographs, and available physical measurements to reconstruct individual assets and their functional interfaces. Agents assist with geometry, appearance, collision modeling, and simulation setup, while human experts guide modeling decisions and review the resulting interactions.

Visual reconstruction is paired with collision geometry tailored to the intended contacts. Fig. 63 illustrates this process for the tractor body, from the physical object and input scan to the reconstructed visual asset and its collision representations. Coarse handling geometry captures the overall shape, while detailed fitting geometry retains features such as bores and reconstructed threaded interfaces. The wheel-shaft close-ups show the two levels of detail: a simplified shape is sufficient for grasping and large-scale handling, whereas high-precision operations such as threading and insertion require the full fitting geometry.

The toy plane and Hot Wheels assembly environments run in Isaac Lab Arena (NVIDIA Isaac Lab-Arena Contributors, 2025), using Newton (The Newton Contributors, 2025) with the MuJoCo Warp solver (Google DeepMind and NVIDIA, 2025). Each object has two sets of colliders: a coarse set for interactions such as grasping and contact with the worktable, and a detailed set for tight-fit insertion and threading bolts into nuts with a powered drill. We use convex and signed distance field (SDF) colliders for these representations. Collision masks determine which sets can interact, allowing larger-scale and fine-scale contacts to be simulated simultaneously. To support long-horizon, multi-stage assembly tasks, we developed an Isaac Lab extension for defining attachment points on individual parts. When a part reaches its intended assembly position within a predefined tolerance, the extension creates a joint between the corresponding rigid bodies, keeping the assembled parts connected during subsequent manipulation.

We validate the environments and assets in several stages. First, we test assembly without a robot, applying forces directly to the parts and the drill. These tests follow the real-world sequence of inserting parts and driving bolts. We also perform grasping and drop tests to check the stability of individual assets and the overall scene. Finally, we replay real-world robot demonstrations in simulation (Fig. 64) to assess asset stability and interactions such as grasping, insertion, and fastening. The validated environments then support the GR00T 2 applications described in Sec. J.2.

J.2. Accelerating GR00T 2 Development with Simulation

Once generated, the simulation environments can be used for many different applications. We first quantify the utility of our simulation suite by comparing the outcomes of GR00T 2 policy evaluations in sim and in real (Sec. J.2.1). We then highlight how these high-fidelity environments contributed to GR00T 2 development, including: optimization of GR00T 2 inference for deployment (Sec. J.2.2), faster iteration on the GR00T 2 pretraining process (Sec. J.2.3), and the development of agentic recovery skills in simulation that assist GR00T 2 during real-world deployment (Sec. J.2.4).



Figure 62: Reconstructed assembly tasks. Toy plane, Hot Wheels car, and toy tractor (left to right), shown in reset configurations (top) and assembled configurations (bottom). Camera and scale are held fixed within each column. The panels illustrate asset configurations rather than robot-policy success.

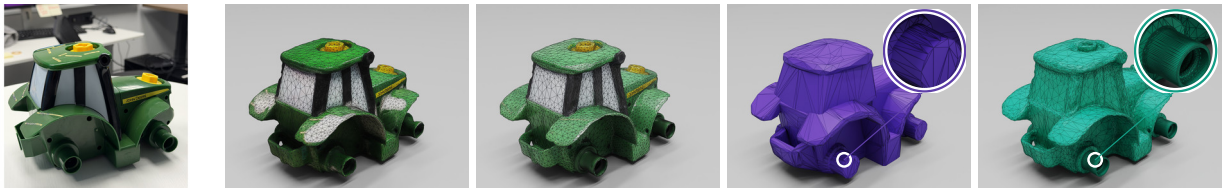


Figure 63: From physical objects to sim-ready assets. Source photograph, input scan, reconstructed visual asset, handling collision hulls (purple), and fitting signed-distance-field (SDF) source surfaces (teal) for the same tractor body, from left to right. Circular close-ups compare the same wheel shaft: a coarse solid handling shape and the fitting bore with reconstructed thread geometry.

J.2.1. Sim and Real Policy Evaluation Correlation Analysis

In order for our simulation suite to be useful, it is important for simulation to be calibrated with the real world. In this section, we gauge our simulation quality by seeing if it can be used as convincing surrogate for real world evaluation (Ranawaka et al., 2026; Jain et al., 2025). We test our policies zero-shot in simulation, without additional fine-tuning or co-training with simulation data, and find that the simulation evaluations strongly correlate with real-world results. This enables the testing of policies at a scale, cadence, and level of reproducibility that would be intractable for real-world evals.

Setup. We evaluate 14 checkpoints in both simulation and the real world on matched setups across 25 tasks on the YAM embodiment. Additionally, we evaluate 10 tasks on the DROID (Khazatsky et al., 2024) embodiment with 5 generalist checkpoints. Each task is evaluated for 5 trials on the real robots, and 10 trials in simulation.

Metrics. We measure sim and real correlation using Pearson Correlation Coefficient (r), Spearman Rank Correlation Coefficient (ρ) and the Mean Maximum Rank Violation (MMRV) proposed by Li et al. (2024). For a simulation that perfectly reflects real-world conditions, we would observe $r \to 1$, indicating that policy performance in simulation matches that in the real world, $\rho \to 1$, indicating that the relative rankings of policies are maintained between simulation and the real world, and MMRV $\to 0$, indicating that the number of policy ranking violations are minimized. These correlations are calculated for task progress, success rate and the

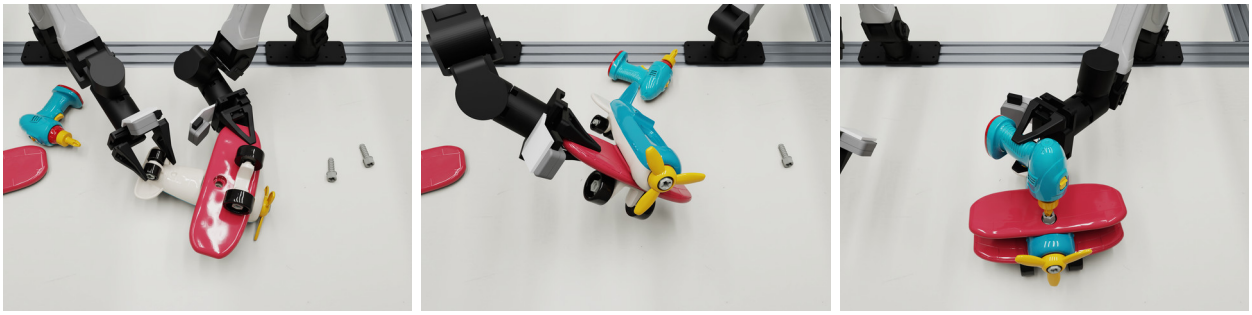


Figure 64: Robot assembly of the toy plane. Wing insertion, reorientation, and fastening with a power drill, shown from left to right during a single continuous demonstration-guided execution.

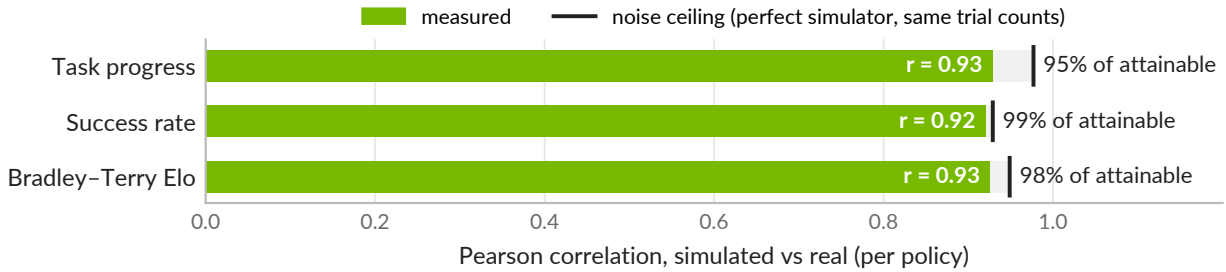


Figure 65: Sim-real Pearson correlation per policy for three computed metrics, compared with the noise ceiling: the correlation a perfect simulator would reach given the same number of real and simulated trials.

Bradley-Terry (BT) (Bradley and Terry, 1952) Elo ratings of the checkpoints. Task progress is the rubric score of a trial, a weighted sum of the completed sub-tasks between 0 and 1, and a trial is successful when the score is 1. Both are averaged within each task and then across tasks. MMRV is reported for task progress and success rate.

Bradley-Terry Elo. Real-world evaluations use blinded A/B tests: in each trial, two or three checkpoints run in randomized order from the same initial setup (based on a corresponding simulation initialization), and every pair of checkpoints in the trial forms one comparison, won by the higher score (equal scores count as a tie). We fit BT ratings to these pairwise outcomes. In simulation, every checkpoint is evaluated on the same fixed initial layouts (10 per task), so any two checkpoints can be compared layout by layout without re-running them.

Results on YAM embodiment. We evaluate 14 GR00T 2 variants across 25 YAM tasks in the same simulator. We completed real-world trials with the A/B evaluation procedure defined earlier. For each trial, we aligned the object initialization with the simulation using projected overlays. We find that our simulator predicts policy performance with high correlations with the real world, with a Pearson r up to 0.93 for task progress and BT Elo score. Since evaluations in both the real world and simulation can be non-deterministic, and the number of trials per task is limited, there is an effective statistical ceiling to how high these correlations can be for even a perfect simulator. Following Lage-Castellanos et al. (2019), we estimate this ceiling by assuming a perfect simulator. For checkpoint c on task t, let $\hat{p}_{ct} = s_{ct}/n_{ct}$ be its real-world success rate (s_{ct} successes in n_{ct} trials), which we treat as its true success probability. In each repetition, we generate two independent evaluations from these true values, one standing in for the real robot and one for a perfect simulator, each with n_{ct} trials:

$$\tilde{x}_c = \frac{1}{T_c}\sum_t \frac{\mathrm{Bin}(n_{ct},\, \hat{p}_{ct})}{n_{ct}}, \qquad \tilde{y}_c = \frac{1}{T_c}\sum_t \frac{\mathrm{Bin}(n_{ct},\, \hat{p}_{ct})}{n_{ct}}, \tag{21}$$

where T_c is the number of tasks for checkpoint c and the two binomial draws are independent. This gives one real and one perfect-simulator score per checkpoint, and we compute the Pearson correlation $\tilde{r} = \mathrm{corr}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$

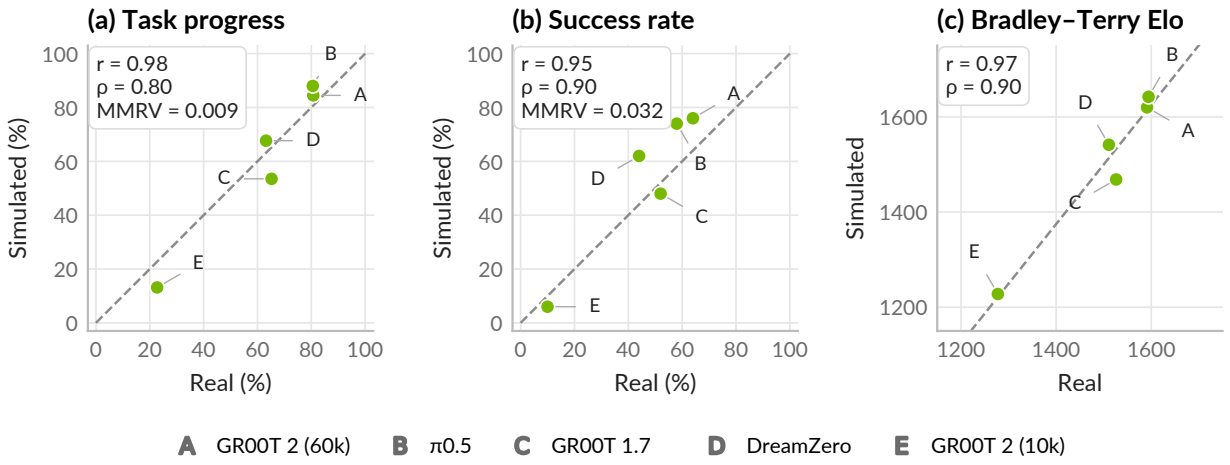


Figure 66: Simulated vs. real-world performance per policy on DROID. We evaluate 5 generalist checkpoints on the DROID embodiment across 10 tasks, with 5 real-world and 10 simulated trials per task, on the same layouts for every checkpoint. Panels and metrics as in Fig. 18: (a) task progress and (b) success rate averaged over tasks (dashed: $y = x$), and (c) Bradley-Terry Elo (dashed: least-squares fit).

across the 14 checkpoints. Since $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ share the same true values, they differ only because of the limited number of trials, so the median of $\tilde{r}$ over $1{,}000$ repetitions is the expected correlation any simulator could achieve with this evaluation budget. As shown in Fig. 65, our simulator produces correlations that are within $95\% - 99\%$ of this ceiling.

Results on DROID embodiment. Separately, we evaluate five generalist DROID checkpoints including GR00T 2 and publicly available models on environments reconstructed from real-world DROID setups. We use the same five layouts per task for each checkpoint in the real-world and the same ten layouts in simulation, to minimize variation in trial initializations. Once again, we see high Pearson r of $0.98, 0.95, 0.97$ for task progress, success rate and BT Elo respectively. This demonstrates that our agentic real-to-sim reconstruction method is viable for different embodiments.

J.2.2. GR00T 2 Inference Optimization through Simulation

Inference optimization aims to help GR00T 2 respond sooner to new observations while preserving task success and smooth motion. We optimize model execution and the processing and transfer of observations, and vary the number of steps used to refine video and action predictions (see Accelerating Model Inference). Simulation provides a repeatable setting for testing how these changes affect behavior. We can restart each trial from the same scene without repeatedly setting up a physical workcell, making it practical to screen settings before robot testing.

Changes that reduce prediction time can also change action predictions, so task success and motion quality must be checked together. For each checkpoint, we keep the number of actions per prediction fixed while comparing original PyTorch execution with TensorRT and CUDA Graphs and testing different numbers of refinement steps. In a YAM cube-handover pilot, several settings completed the task but differed in motion smoothness. Follow-up trials of the same checkpoint and task on a physical YAM robot also showed this distinction. In separate runtime comparisons, the optimized implementation supported more frequent updates to the action plan with comparable task progress in simulation.

Simulation also helps us prototype real-time chunking (RTC), which connects new action predictions to actions already scheduled for execution. In separate simulation experiments (see Policy Smoothness), we compare RTC training with and without noise added to the previous action sequence. Repeated trials from matched scenes with controlled prediction delays help us assess whether smoother transitions between action sequences

preserve task success, without testing every delay setting on physical robots.

We measure prediction latency directly on the target hardware, separately from simulated task duration. Ordinary synchronous simulation pauses while waiting for predictions, so it does not reproduce the effect of inference delay on a moving robot. The RTC experiments explicitly model this delay, but physical-robot tests remain necessary to check behavior under actual deployment timing. Simulation narrows the settings taken to these tests, where we assess task success and motion quality on the target robot.

J.2.3. GR00T 2 Pretraining Acceleration with Simulation

Simulation enabled faster iteration on GR00T 2 pretraining through many different means.

Monitoring of pretraining progress and checkpoint selection. Evaluating a single pretraining checkpoint across a large set of real-world tasks can take hours. Simulation provides a faster way to compare policies across a diverse task set during model development. During GR00T 2 pretraining, we evaluated intermediate checkpoints on a set of tasks from our simulation suite to inspect their closed-loop behavior and compare model capabilities as training progressed. These comparisons help us identify overfitting as we iterate on the model, and assess the relative performance of our checkpoints and external baselines without running extensive real-world evaluations for every candidate.

Simulation also guided checkpoint selection: we used a subset of our simulation suite (25 tasks) to determine which checkpoints showed promising closed-loop evaluation results. This provided a stable way to understand if checkpoints were worth physical robot evaluation. As a concrete example, we used this workflow to finalize our pretrained model: simulation results narrowed the candidates to three checkpoints, which we then evaluated on physical robots to select the final checkpoint. In this way, simulation was a tool to guide checkpoint analysis and development, but we still used physical-robot evaluation to validate conclusions about real-world performance.

Optimizing the data pipeline. Changes to the data loading and processing pipeline are costly to assess through full-scale pretraining. We expect a new pipeline that processes the data correctly and preserves training dynamics to produce model performance comparable to the existing pipeline. We test this expectation with small-scale training and closed-loop simulation before adopting a new pipeline for pretraining.

As an example, we compared five configurations that varied sequence packing, mixed-shard sampling, and text-token masking. For each configuration, we post-trained a model on real-robot teleoperation data from the YAM Scoop and Unscrew tasks, then evaluated it on simulated versions of those tasks. Simulation was used only for evaluation. The new data loading pipeline showed no observed regression in closed-loop simulation performance relative to the existing pipeline, giving us confidence to adopt it for pretraining.

J.2.4. Agentic Skill Development in Simulation

Simulation also helps us develop specialized manipulation skills. As described in Sec. G.1, ASPIRE learns skills that code-as-policy (CaP) can invoke alongside GR00T 2 to carry out difficult steps or recover from failures. For example, the toy plane assembly task (Fig. 64) provided a repeatable setting for developing recovery skills that transferred to physical robots. In simulation, we could test candidate behaviors, inspect failures, and refine the skills before evaluating them on the physical system.

J.3. Simulation Benchmarking

We evaluate GR00T 2 on five simulation benchmarks that span single-arm mobile manipulation (RoboCasa365), bimanual manipulation (RoboDojo), zero-shot real-to-sim tabletop manipulation (RoboLab, MolmoSpaces) and humanoid loco-manipulation (G1 Factory). Unless stated otherwise, each GR00T 2 result is a single post-trained checkpoint initialized from the same pretrained GR00T 2 model, evaluated with the benchmark's official task set, episode budget and success criterion. Among VLA and WAM policies on the public leaderboards,[4] **GR00T 2**

[4]Baselines are taken from the official leaderboards as of November 6, 2026 with the exception of GR00T 1.7 which we retrain and evaluate ourselves.

Table 30: Results on the RoboCasa365 benchmark. We report success rate (%). The best and second-best results are highlighted in bold and underline, respectively.

Method	Average	Atomic-Seen	Comp.-Seen	Comp.-Unseen
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	16.9	39.6	7.1	1.2
GR00T 1.7 (NVIDIA, 2026a)	22.1	41.7	17.7	4.4
Paimon-0	58.1	79.3	55.8	36.7
Axiom-0 (Shu et al., 2026)	61.6	<u>86.5</u>	59.4	35.8
Awomo-0.5V (Awomo-WAM Team, 2026)	62.5	85.0	<u>60.4</u>	39.4
minionx	62.6	81.8	60.0	<u>43.6</u>
Awomo-0.5I (Awomo-WAM Team, 2026)	<u>64.4</u>	<u>86.5</u>	58.8	**45.2**
GR00T 2 (Ours)	**67.2**	**87.4**	**73.6**	38.2

Table 31: Results on the RoboDojo benchmark. Dimension columns report success rate (%), while the last two columns report the average score and success rate (%). GR00T 2 is evaluated on all three layout banks. Tab. 19 reports layout bank 0 only, for all four GR00T 2 configurations including RL and agentic execution. The best and second-best results are highlighted in bold and underline, respectively.

Method	Gen.	Precision	Long-Hor.	Memory	Open	Avg. Score	Avg. SR
GR00T 1.7 (NVIDIA, 2026a)	4.1	1.1	11.8	2.6	1.4	7.65	4.20
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	8.2	5.5	14.7	4.7	1.7	11.44	6.93
Rex-M1 Preview	26.1	23.8	33.0	48.2	6.1	33.79	27.44
Simate-beta	28.0	26.9	<u>43.4</u>	33.0	8.5	33.95	27.96
Awomo-0.5 (Awomo-WAM Team, 2026)	23.8	33.5	26.8	41.2	**22.9**	35.34	29.64
VPP2 (Guo et al., 2026)	29.5	<u>35.2</u>	37.0	<u>55.6</u>	4.1	39.26	32.26
WALL-OSS-1.0-lite	<u>37.9</u>	29.1	32.3	**56.0**	<u>19.6</u>	**42.59**	<u>34.97</u>
GR00T 2 (Ours)	**41.2**	**40.0**	**54.1**	36.0	6.3	<u>41.73</u>	**35.52**

achieves the highest success rate on RoboCasa365 and RoboDojo, places in the top three on RoboLab and MolmoSpaces, and improves on GR00T 1.7 on G1 Factory.

RoboCasa365. RoboCasa365 (Nasiriany et al., 2026) is a large-scale household simulation benchmark with 365 tasks across 2,500 procedurally generated kitchens, in which a mobile manipulator acts from three RGB cameras. Following the official multi-task protocol, we train on the Human300 pretraining mixture and evaluate on 50 target tasks split into 18 Atomic-Seen, 16 Composite-Seen and 16 Composite-Unseen tasks. The last split is absent from training and therefore tests zero-shot composition of learned skills. The overall score is the mean of the 50 per-task success rates. Results are shown in Tab. 30.

GR00T 2 reaches 67.2% average success, 2.8 percentage points above the best prior entry, and it leads on the Atomic-Seen and Composite-Seen splits. Notably, its largest margin is on the Composite-Seen tasks, multi-step activities that chain several learned skills, where it reaches 73.6%, 13.2 points above the next entry (Awomo-0.5V, 60.4%).

RoboDojo. RoboDojo (Chen et al., 2026a) is a bimanual benchmark in Isaac Sim (NVIDIA, 2025) on the 14-DoF ARX-X5 platform, with 42 tasks grouped into five capability dimensions: Generalization (standard and randomized layouts), Precision, Long-Horizon (up to 1,900 control steps), Memory and Open (open-vocabulary instructions with no training demonstrations). Following the official protocol, we run the evaluation over all 42 tasks (54 task configurations) on three layout banks (6,300 episodes) and report the partial-progress score and success rate. Results are shown in Tab. 31.

GR00T 2 achieves the highest success rate, 35.52%, and an average score of 41.73. GR00T 2 ranks first on Generalization, Precision and Long-Horizon, showing that GR00T 2's strength carries over from single-arm

Table 32: Results on the RoboLab benchmark. We report success rate (%) overall and per difficulty tier. The best and second-best results are highlighted in bold and underline, respectively.

Method	Overall	Simple	Moderate	Complex
GR00T 1.7 (NVIDIA, 2026a)	18.9	22.8	15.9	10.9
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	28.0	29.7	31.5	13.5
Atomic-WAM (Shi et al., 2026)	39.6	44.4	40.5	19.4
HiDream-O1-Embodied	39.9	43.1	37.7	32.9
FLUX 3 Action (Black Forest Labs, 2026)	42.9	49.1	39.2	28.2
BiMind v0.5	<u>49.2</u>	<u>51.6</u>	<u>52.6</u>	32.4
Lakeside	**51.0**	**57.6**	44.1	**42.4**
GR00T 2 (Ours)	47.4	45.6	**54.6**	<u>37.6</u>

Table 33: Results on the MolmoSpaces benchmark. We report success rate (%). The best and second-best results are highlighted in bold and underline, respectively. Baselines: $\pi_{0.5}$-DROID (Physical Intelligence et al., 2025), GR00T 1.7 (NVIDIA, 2026a), MolmoAct2 DROID (Fang et al., 2026), PRTS-Droid (Zhang et al., 2026), WALL-OSS-0.5 (Yu et al., 2026b), and Cosmos3-Nano-Policy (NVIDIA Cosmos Team, 2026).

	Articulated		Pick				Pick-and-Place			
Method	Open	Close	v1.5	v2-classic	v2-filament	v2-RandCam	v2	NextTo	Color	Avg.
$\pi_{0.5}$-DROID	22.7	65.1	18.1	6.4	7.0	8.0	12.0	10.2	10.4	17.77
GR00T 1.7	11.9	68.0	35.3	11.7	12.5	7.8	9.4	12.9	11.6	20.12
MolmoAct2 DROID	11.7	71.3	43.4	20.5	21.9	11.1	15.6	18.5	19.3	25.92
PRTS-Droid	12.0	76.3	47.5	18.1	20.5	21.6	17.4	19.0	18.2	27.84
WALL-OSS-0.5	16.8	78.5	**78.4**	<u>47.9</u>	<u>47.2</u>	<u>30.1</u>	13.8	15.9	18.3	38.55
Cosmos3-Nano-Policy	<u>32.0</u>	<u>79.1</u>	66.5	32.3	32.2	25.7	26.0	<u>22.9</u>	34.3	39.00
ZoeInside-VLA	**36.6**	**85.7**	<u>74.8</u>	**51.8**	**55.9**	**45.5**	<u>37.8</u>	21.2	<u>36.2</u>	**49.50**
GR00T 2 (Ours)	20.7	79.0	71.9	42.0	42.9	20.3	**44.2**	**37.5**	**47.6**	<u>45.12</u>

manipulation to coordinated bimanual control. On Precision it leads the strongest prior entry by 4.8 points in success rate (40.0% vs. 35.2%) and reaches the highest partial-progress score (47.7 vs. 43.0), meaning it advances further through fine-grained manipulation even when an episode does not finish.

RoboLab. RoboLab (Yang et al., 2026) is a high-fidelity Isaac Lab benchmark of 120 tabletop tasks on the DROID (Khazatsky et al., 2024) Franka setup, grouped into 64 simple, 39 moderate and 17 complex tasks. Following the official protocol, we run 10 episodes per task (1,200 episodes) with the default instructions and report binary success rate. Our policy is post-trained only on real DROID data, so this evaluation is zero-shot real-to-sim transfer. Results are shown in Tab. 32.

GR00T 2 reaches 47.4% success, placing in the top three among non-agentic entries on the leaderboard. Since GR00T 2 is post-trained only on real-world DROID data, this result indicates GR00T 2's robust zero-shot transfer to unseen simulated scenes and tasks.

MolmoSpaces. MolmoSpaces (Kim et al., 2026) evaluates a Franka DROID policy in procedurally generated MuJoCo houses populated with Objaverse (Deitke et al., 2023) objects. Following the leaderboard's default view, we report success rate on nine suites: articulated Open and Close, four pick suites (Pick-v1.5 and Pick-v2 under the classic renderer, the filament renderer and randomized cameras) and three pick-and-place suites (plain, next-to and color-referred), 8,915 episodes in total. The overall score is the unweighted mean over the nine suites, with unsubmitted suites counted as zero. As with RoboLab, we evaluate the same real-DROID post-trained checkpoint zero-shot. Results are shown in Tab. 33.

GR00T 2 scores 45.12%, second only to ZoeInside-VLA (49.50%), and it has the best score on all three

Table 34: Results on G1 Factory. We report success rate (%) over 100 episodes per task. The best results are highlighted in bold. Baseline: GR00T 1.7 (NVIDIA, 2026a).

	Box			Drill				Push		
Method	Lift	Lift-Floor	Table-to-Shelf	Lift	Lift-Obs	PnP	From-Holder	Shelf	Button	Overall
GR00T 1.7	98	96	77	66	75	69	44	58	42	69.4
GR00T 2 (Ours)	**100**	**100**	**90**	**92**	**87**	**80**	**67**	**74**	**72**	**84.7**

pick-and-place suites (44.2%, 37.5% and 47.6% on v2, NextTo and Color). Notably, it leads the spatially referred NextTo suite by 14.6 points (37.5% vs. 22.9% for Cosmos3-Nano-Policy), indicating strong grounding of spatial language. Its weakest suite is Pick-v2 with randomized cameras (20.3%), indicating sensitivity to camera viewpoint.

G1 Factory. G1 Factory (NVIDIA, 2026c) adapts the nine Unitree G1 loco-manipulation tasks introduced in HumanoidMimicGen (Lin et al., 2026) to NVIDIA Isaac Lab-Arena. The tasks involve lifting and shelving boxes, retrieving and transferring drills, pressing buttons, and pushing a shelving cart with a 43-DoF G1 humanoid. The policy commands the arms, hands, and waist together with base-height and navigation commands that are executed by a lower-body controller. We train on 100 demonstrations per task from the released G1 Factory dataset and evaluate each policy over 100 episodes per task (900 episodes in total). We compare against GR00T 1.7 and report overall success rates in Tab. 34.

GR00T 2 achieves an overall success rate of 84.7%, outperforming GR00T 1.7 (69.4%) by 15.3 percentage points. It improves success rates on all nine tasks and achieves 100% success on box lifting from both the table and the floor. Remaining failures are concentrated in contact-rich tasks that require precise end-effector placement while the robot maintains balance: drill retrieval from the holder, shelf pushing, and button pressing.

K. Infrastructure

This section describes the compute, data, and research infrastructure used to develop GR00T 2.

K.1. Overall Software and Hardware Stack

Training a robotics foundation model like GR00T involves: (1) the ingestion and conversion of large volumes of multimodal recordings, such as video, motion, and language annotations, into training-ready datasets, (2) pretraining and post-training at scale on GPU clusters, and (3) the deployment and evaluation of the resulting policies in simulation and on robots. This section describes the infrastructure we built for these workloads and used to develop GR00T 2. At each layer of the stack we build on widely used open-source software, such as PyTorch, Ray, SkyPilot, and Kubernetes, improving it where needed and adding robotics-specific components on top. We describe the overall stack first and each component separately below.

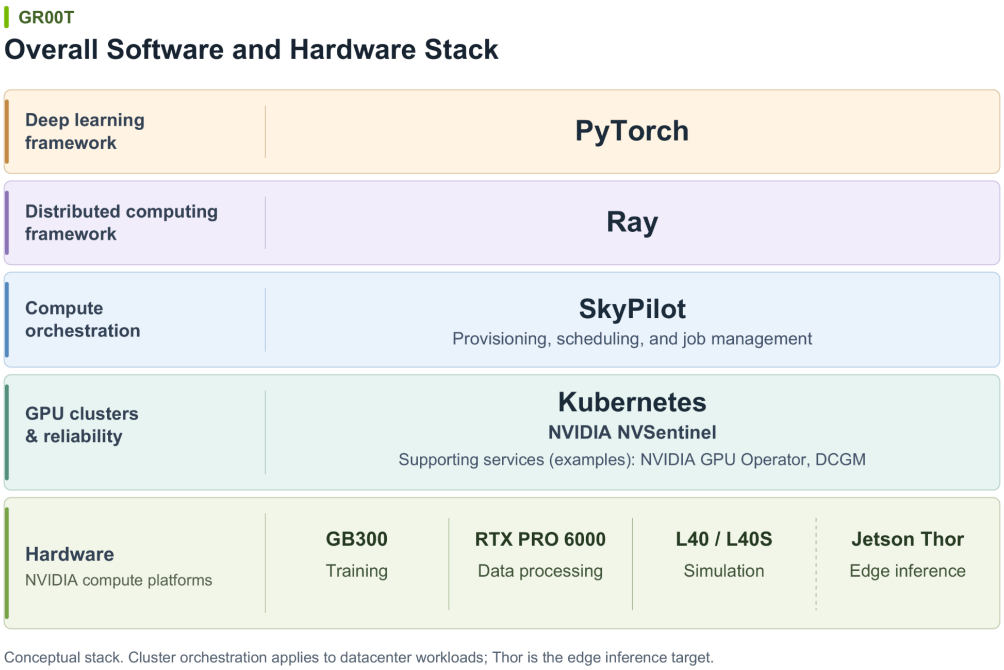


Figure 67: Overall software and hardware stack of GR00T 2.

Fig. 67 shows the stack from the hardware up. Each workload runs on the NVIDIA platform suited to it: GB300 for training, RTX PRO 6000 for data processing, L40 and L40S for simulation and evaluation, and Jetson Thor for inference on the robot. Kubernetes manages the GPU clusters, with NVIDIA NVSentinel monitoring node health and services such as GPU Operator and DCGM alongside it. SkyPilot (Yang et al., 2023) provisions clusters and schedules and manages jobs across them. Ray (Moritz et al., 2018) distributes work within a job, for training workers and for data ingestion alike, and PyTorch is the deep learning framework.

The following subsections describe the hardware, cluster and job management, distributed training, inference, and data infrastructure. Reinforcement-learning post-training runs on the same stack, and its infrastructure is described in Sec. F.3.

K.2. Hardware

GR00T 2 has many workloads that span large-scale foundation-model pretraining, simulation and policy evaluation, online post-training, and latency-sensitive robot deployment. These stages place different demands on accelerator memory, interconnect bandwidth, storage throughput, scheduling, and end-to-end latency. As a

result, each stage has different hardware and system requirements that need to be accounted for.

K.2.1. Pretraining

GR00T 2 pretraining ran on NVIDIA GB300 NVL72 systems because the workload placed a premium on GPU memory capacity, Tensor Core throughput, and networking/collective communication acceleration. Each Blackwell Ultra GPU provides up to 288 GB of HBM3e with 8 TB/s of memory bandwidth and approximately 2.5 PFLOPS of dense BF16/FP16 Tensor Core compute. Because the GR00T 2 model was small enough to fit without pipeline parallelism, this memory capacity allowed the model parameters, each rank's optimizer state, and activations for a per-device batch of 16 to remain in GPU memory. The larger per-device batch provided more matrix-multiplication work for the Tensor Cores and amortized kernel-launch overhead, improving compute utilization (Aubrey and Stam, 2025).

For networking, each GB300 NVL72 rack contains 18 compute trays and 72 GPUs connected as one NVLink domain. Fifth-generation NVLink provides 1.8 TB/s of bidirectional bandwidth per GPU. Within each NVLink domain, NVLink SHARP (NVLS) offloaded reductions to the NVSwitch. East/West communication used the RoCE scale-out fabric, with one 800 Gb/s ConnectX-8 SuperNIC per GPU (NVIDIA, n.d.a,n).

The full cluster contained 2,016 GPUs across 28 GB300 NVL72 racks. The GR00T 2 main pretraining run used 1,024 GPUs, while the remaining cluster capacity supported other training experiments concurrently.

K.2.2. Simulation and Policy Evaluation

Simulation and policy evaluation ran on NVIDIA L40 and L40S clusters because these workloads required graphics, simulation compute, and policy inference on the same accelerator. Both GPUs use the Ada Lovelace architecture and provide 48 GB of GDDR6 memory with 864 GB/s of memory bandwidth, 142 third-generation RT Cores, and 568 fourth-generation Tensor Cores. The RT Cores accelerate hardware ray tracing for rendered observations, while the Tensor Cores accelerate low-precision policy inference. The L40S retains the L40's memory capacity and graphics resources while providing approximately twice its peak low-precision Tensor Core throughput, allowing more inference work to accompany simulation (NVIDIA, 2023, n.d.c).

The 48 GB memory capacity provides headroom to keep the policy checkpoint, simulator state, and rendering buffers resident while processing multiple environments, although achievable concurrency depends on the simulator and task. Unlike GB300 NVL72 systems, L40 and L40S are PCIe Gen4 GPUs without NVLink (NVIDIA, 2023, n.d.c). Evaluation therefore scales primarily by distributing independent environment, checkpoint, and random-seed shards across workers rather than through frequent cross-GPU collectives. Network and storage throughput remain important for staging checkpoints and collecting rollout artifacts, but the primary capacity metrics are environments per GPU, rollout throughput, and checkpoint-evaluation turnaround time.

K.2.3. Online Post-Training

Online post-training ran across heterogeneous H100, GB300, and L40/L40S pools because no single accelerator could satisfy every stage of the workflow. H100 and GB300 systems supported distributed policy optimization. L40 and L40S systems supported simulation-based collection and evaluation, where graphics capabilities were needed. Capacity was allocated independently to each stage and adjusted as workload bottlenecks shifted.

K.3. Cluster and Job Management

K.3.1. Kubernetes

Kubernetes provides the AI factory execution layer for GR00T 2, with the cluster software stack deployed through NVIDIA AI Cluster Runtime (AICR) (NVIDIA, n.d.d). This integrates NVIDIA DSX OS components for GPU enablement, high-performance networking, and infrastructure resilience. NVIDIA GPU Operator and Network Operator manage the GPU and networking software stacks, while KAI Scheduler and Kueue provide workload scheduling and admission control, priorities, resource quotas, and topology-aware placement for multi-node workloads.

On GB300 NVL72 systems, NVIDIA's Dynamic Resource Allocation (DRA) driver manages ComputeDomains, enabling secure multi-node NVLink access for workload pods. NCCL uses this fabric for collective communication within each NVLink domain and RoCE for communication across racks.

NVSentinel, also part of NVIDIA DSX OS, monitors GPU and node health and supports policy-driven isolation and remediation of faulty nodes. Together with SkyPilot's job recovery and the training runtime's checkpoint-resume mechanisms, these components support reliable operation of the GR00T 2 AI factory.

K.3.2. SkyPilot

GR00T 2 training uses SkyPilot (Yang et al., 2023) as the infrastructure orchestration layer for distributed GPU workloads. A shared SkyPilot API control plane provides a common interface for submitting experiments, provisioning workload clusters, monitoring execution, and managing cluster lifecycles. The platform connects this control plane to GPU infrastructure through configured Kubernetes contexts and workspace access controls. Infrastructure configuration and administrative policies are maintained separately from model code, allowing researchers to specify training requirements while platform operators manage credentials, storage integration, and infrastructure access.

Each experiment runs as a SkyPilot managed job. Once resources are available, the workload containers form a Ray cluster, and the training driver creates a distributed worker for each GPU. SkyPilot manages infrastructure execution, Ray places training workers, and PyTorch, NCCL, and the optimizer implementation perform distributed model training.

The infrastructure accounts for the communication and data-access requirements of large-scale joint video-and-action training. Kubernetes and Kueue coordinate admission of multi-node workloads using GPU topology information, while the OCI workload configuration requests RDMA network attachments and placement within GPU clique topology domains. The training runtime additionally orders distributed ranks according to physical topology. Shared Lustre storage supplies datasets, pretrained assets, and persistent checkpoints, while node-local scratch supports temporary runtime files. These components connect resource allocation to the communication and storage requirements of the GR00T 2 training workload.

Launch-time administrative policies validate infrastructure selections, preserve backend-specific placement settings, and configure centralized log forwarding. Workspace access controls govern access to the available compute environments. The platform also supports policies that derive queue assignments and scheduling priorities from administrator-managed workspace configuration. SkyPilot managed-job recovery works together with bounded retries and checkpoint resume in the training stack. This separation enables infrastructure recovery while retaining training state in persistent storage.

K.4. Distributed Training

Distributed training of GR00T 2 requires coordinating multimodal input preparation with synchronous model updates. We design the runtime to reduce memory use, limit waiting between training stages, and preserve progress across interruptions.

K.4.1. Parallelism and Memory Management

We use synchronous data parallelism with DeepSpeed ZeRO-2. Model weights are replicated on each GPU, while gradients and optimizer state are partitioned across ranks to reduce per-device memory use. BF16 gradient synchronization overlaps with model computation to reduce exposed communication time.

K.4.2. Efficient Multimodal Computation

Multimodal training requires computation in both frozen feature encoders and the trainable model. Depending on the recipe, the runtime supports TensorRT video encoding, fused text-encoder operations, efficient attention kernels, and PyTorch compilation.

Activation recomputation reduces memory use at the cost of additional computation. Attention optimizations

retain the model's masking rules, and precision choices are evaluated for numerical stability and model quality.

Where enabled, the runtime prepares the next batch's encoded features during the current batch's gradient synchronization and optimizer update to reduce waiting for feature encoding.

K.4.3. Data Loading

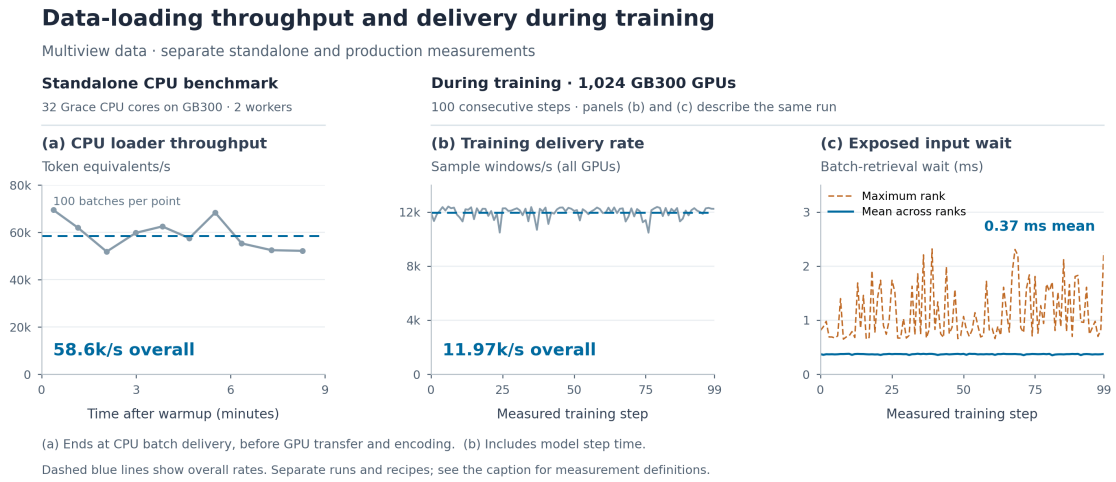


Figure 68: Data delivery during multi-view pretraining on 1,024 NVIDIA GB300 GPUs over 100 consecutive steady-state steps. (a) Delivered sample-window throughput. (b) Exposed batch-retrieval wait, showing the mean and maximum across ranks.

GR00T 2's data loader combines versioned multimodal storage, offline quality filtering, and asynchronous batch preparation. Training recipes specify the data snapshots to read and how samples are selected and interpreted across embodiments. The training mixture used for the measurements below comprises 25 versioned dataset stores, totaling 15.6 million episode records and 46.2 billion recorded time steps before applying the run's sampling views. These totals aggregate the selected stores without deduplicating recording content across stores.

Versioned storage and schema. We store training data in LanceDS, built on the Lance columnar format (Pace et al., 2025; Lance Contributors, 2026). Each dataset contains a step table for state, action, annotations, and quality-control (QC) signals, an episode table, and a media table holding the original videos. Lance's versioning preserves earlier snapshots when episodes are added or annotations or QC signals change, and new QC columns can be added without rewriting existing columns. A per-dataset commit record ties each snapshot's table versions to its normalization statistics and metadata. A training run pins one snapshot per dataset, so data, statistics, and metadata are reproduced together. Snapshots are replicated to the object store and to training clusters in commit order, with the commit record copied last, so a replica serves a pinned snapshot completely or not at all.

Datasets share a schema in which state and action groups are packed into one state column and one action column. Group names and offsets are recorded in schema metadata, allowing the loader to select fields by name despite differences in their stored order. At read time, the recipe's embodiment tag selects groups, cameras, annotations, normalization statistics pooled across datasets of that embodiment, and, for dexterous hands, fingertip force signals and deformation video. The same tag determines the model's embodiment-specific projection layers. Datasets of the same embodiment can be merged with a common group order.

Steps within each episode are stored contiguously, and the loader fetches episode ranges through batched reads. Step-table reads bypass filesystem readahead, which amplified per-shard step-table I/O by 28× to 60× in an earlier production benchmark.

Window-aware QC views. A view records the steps eligible to serve as training-sample starts. User-defined Python filters evaluate an episode's columns over the temporal window consumed by each sample, so eligibility accounts for the observations and action targets that training will read. Filters can also select whole episodes by metadata such as task family, collection date, or data origin. Views are computed offline and stored as runs of consecutive eligible steps per episode. Training samples directly from these runs without evaluating the filters in the loading path.

Each view is identified by a hash of the dataset, filter code and arguments, and sample window. Cosmetic edits to a filter keep its hash. Views are tagged with the table versions they serve and can be shared by identifier across runs. A view is reused at a new version when the dataset's rows and the columns its filters read are unchanged. Views are rebuilt automatically: each new snapshot triggers a build of every registered view in the same job that writes the snapshot, and the snapshot and its views are published together, so recipes can pin new snapshots without a manual build step.

Views also carry normalization statistics (mean, standard deviation, minimum, maximum, and 1st and 99th percentiles) computed over the samples they keep. Each statistics entry is keyed to the training configuration it was computed under: the snapshot, embodiment, action horizon, and which actions are trained relative to state.

For human demonstrations, QC filters check head and wrist motion, hand-pose plausibility, and keypoint visibility, excluding tracking failures such as frozen head poses and abrupt fingertip jumps. For robot demonstrations, filters exclude task-irrelevant motion, idle time, and annotated mistakes or failures. Storing these signals as step-table columns allows curation rules to evolve alongside the versioned dataset.

Sampling and batch delivery. Sampling choices are generated on demand from deterministic per-rank step seeds, avoiding a precomputed sampling sequence. Samples are mixed across shards to target the recipe's dataset proportions in each global step. Sampling without replacement accommodates datasets of different lengths without requiring aligned epoch boundaries.

Batch preparation uses three asynchronous stages. The I/O stage prefetches video shards from shared storage into node-local NVMe. The decoding stage reads cached shards, decodes frames, and prepares trajectory metadata. In a representative three-view shard benchmark, decoding was estimated to account for about 90% of preparation time. A separate sequential-decoding microbenchmark measured approximately 7,000 frames/s with one decoding thread at 320×240. The loader stage applies augmentations (random crops retaining 95% of each spatial dimension, resizing to 224×320, and codec augmentation), collates samples, and prepares asynchronous transfers from pinned host memory to the GPU. Keeping prefetched video shards on NVMe reduces the host RAM required for buffering and allows a deeper prefetch window. Dedicated worker pools, SIMD-vectorized decoding, and C++ augmentation implementations reduce CPU processing overhead.

On clusters with limited shared-storage capacity, the I/O stage instead stages upcoming shards' source files on node-local NVMe through a cluster-lifetime object store. Each file is read from the data source once per cluster and forwarded between nodes, so source traffic does not grow with node count. A preloaded interposer serves data-loader reads from local NVMe at the files' original paths, and a file that cannot be delivered is an error rather than a skipped episode.

Variable-length token packing reduces padding when samples differ in camera configuration or sequence length. In the measured training run, packed sequences occupied 94.5% of the available positions in 32,768-token rows.

Reproducibility and diagnostics. The loader controls random seeds for sampling and spatiotemporal augmentation. Training checkpoints record loader progress (including active shard offsets, sample indices, and worker state) together with the configuration, including each dataset's pinned snapshot. The supported sharded pipeline restores sample-sequence progress when resuming with the same data configuration and rank/worker topology. Each sample carries a composite identifier comprising dataset ID, trajectory ID, and frame index,

linking the training input to its source.

Structured events from each worker and distributed rank trace the batch-loading lifecycle. Correlating these events with sample identifiers helps investigate storage delays, worker starvation, and loss anomalies. A CPU-only benchmark harness uses Gloo and identity model stubs to profile the loading pipeline before GPU runs. Fig. 68 and Tab. 35 report the measured input-delivery performance.

K.4.4. Straggler Control and Communication

A rank delayed by input preparation or a CPU pause can hold up collective operations across the training job. We profile CPU activity, GPU execution, and collective timing together to distinguish late-arriving ranks from slow communication.

Depending on the configuration, the runtime coordinates garbage collection, controls CPU thread placement, balances loader concurrency, and reports metrics asynchronously to limit host delays that interrupt GPU work and synchronization.

K.4.5. Checkpointing and Recovery

Checkpointing coordinates a consistent foreground snapshot of training state, followed by background serialization and writes. Asynchronous saving reduces exposed I/O, and a checkpoint becomes available for continuation once all required writes complete.

Model artifacts provide weights for evaluation and downstream initialization. Continuation checkpoints additionally retain optimizer partitions, scheduler history, random-number-generator state, and supported loader progress. Restoring training requires compatibility between the saved state and the resumed runtime.

K.4.6. Training Efficiency

We assess training efficiency across input retrieval, frozen feature encoding, model computation, gradient synchronization, and optimizer updates. Step time is interpreted alongside model quality, accounting for differences in active modalities, trainable components, and input shapes.

Tab. 35 reports input-delivery performance over 100 steady-state pretraining steps on 1,024 NVIDIA GB300 GPUs. These measurements characterize the data pipeline. They do not measure whole-model FLOP utilization.

Table 35: Measured training-input delivery over 100 steady-state steps on 1,024 NVIDIA GB300 GPUs. The wait-time share is relative to aggregate measured rank-step time.

Measurement	Result
Aggregate sample-window throughput	∼12,000 windows/s
Per-GPU sample-window throughput	11.7 windows/s
Mean exposed data-loader wait	0.37 ms
99th-percentile exposed data-loader wait	0.61 ms
Exposed wait / aggregate measured rank-step time	0.049%

We monitor Overall FLOP Utilization (OFU), a hardware-counter-based complement to application-level Model FLOPs Utilization (MFU), alongside step time and task-level evaluation (Pedersen et al., 2026). DCGM provides GPU telemetry (NVIDIA, n.d.e), and Weights & Biases records step time. Utilization helps diagnose accelerator use, while elapsed time and model quality determine whether a change is useful. An optimization can reduce step time without changing the utilization signal. Assessment of a full training run also accounts for startup, compilation, checkpoint pauses, interruptions, and replayed work.

K.5. Inference Infrastructure

We use a client-server architecture to deploy GR00T 2 across embodiments and execution backends. The server handles policy inference, while robot clients connect it to embodiment-specific sensing and control.

K.5.1. Policy Serving

The robot client sends camera history, robot state, and task instructions through a shared request-reply interface. The server prepares inputs for the loaded model and returns predicted action chunks, which the client maps to the robot's command format. Checkpoint configuration specifies the required history and embodiment-specific input and action representations. The interface supports PyTorch and optimized runtimes, with client and server running on the same host or separate hosts.

For streaming or prefetched observations, the server verifies that prepared visual inputs match the camera history in the prediction request.

K.5.2. Compatibility and Readiness

Compiled runtime artifacts are specific to the checkpoint and execution environment. The server checks checkpoint identity, input shapes, numerical precision, software versions, and GPU target before use, rejecting incompatible artifacts. Backends that require warmup initialize execution with a representative observation from the client. Readiness indicates that the required checks and initialization have completed.

On devices with limited GPU memory, the server can offload text-encoder weights to CPU memory, leaving more accelerator memory for policy execution.

K.5.3. Action Delivery

The client buffers action chunks and schedules commands at the controller's execution rate. With asynchronous prediction enabled, it requests the next chunk while the controller executes the current one. Clients using timestamped waypoints discard expired targets before sending commands to the controller. Inference acceleration and policy-quality validation are described in Sec. I.1. RTC and action-timing methods are described in Sec. I.2 and I.3.

K.6. Scalable Multi-Modal Data Processing

Scaling GR00T 2 across tasks and environments requires turning increasingly large and diverse collections of human demonstrations into training-ready data. Our infrastructure processes petabyte-scale monthly input volumes, with individual production jobs using nearly 10,000 vCPUs. At this scale, preparation must sustain both massive data movement and heterogeneous computation: multi-camera video must be retrieved, decoded, and resized. Motion records must be transformed and, where needed, retargeted to the training hand representation, and observations, actions, and annotations must be aligned in time. Recording duration, resolution, encoding, and retargeting requirements vary across sources, so equal numbers of episodes can impose very different I/O and compute demands. The output is a consistent set of indexed multimodal episodes, packaged into training-ready shards (Fig. 69).

The central challenge is that these conversions share data and compete for limited working storage. A long video can contain many episodes, while one episode can depend on several camera files and separately stored motion records. Staging a shared file once avoids repeated downloads, but its slowest consumer determines when that storage can be reclaimed. A slow conversion or output upload can therefore retain local files, block retrieval of unrelated inputs, and leave compute workers idle. Interruptions compound the problem by adding repeated transfers and expensive recomputation. Sustained throughput requires coordinating data lifetime with execution progress across the entire pipeline.

The ingestion pipeline coordinates execution with the lifetime of its shared inputs: new work is admitted according to committed storage, dispatched where its inputs reside, and releases capacity only after its outputs are safely uploaded. Active files move directly from object storage to processing nodes, allowing conversion to begin before the entire selected collection has been retrieved. This removes intermediate staging on a shared filesystem while making input placement, retention, and reconstruction explicit responsibilities. Fig. 70 shows the mechanisms supporting this lifecycle.

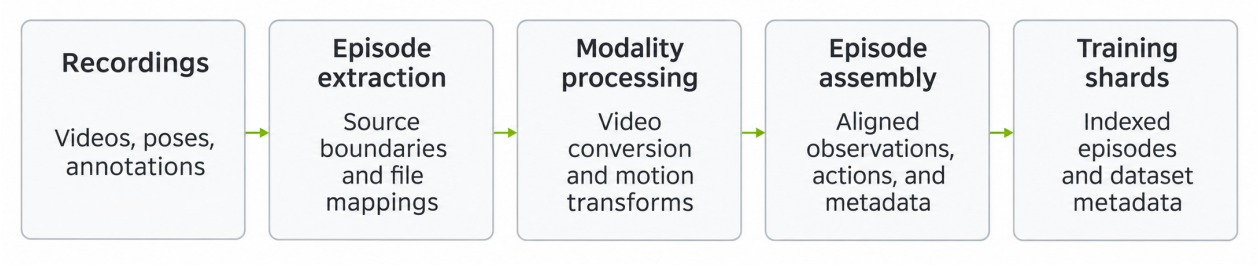


Figure 69: Transformation of human demonstration recordings into aligned multimodal episodes and training-ready dataset shards.

Admit and place work (Fig. 70, panels B and C). Equal episode counts can imply very different file footprints and sharing patterns, so task limits alone cannot control storage demand. Before downloading an input group, admission accounts for resident files, pending transfers, and output headroom against the node's budget. Required inputs must be ready before episode dispatch, and consumers execute on the node holding those inputs. Separate Ray worker pools (Moritz et al., 2018) give loading, conversion, and publication independent concurrency. Bounded queues limit outstanding work while storage admission limits the capacity committed to it.

Retain inputs through completion, then release capacity (Fig. 70, panels A to C). A downloaded video may serve several episodes, including consumers still waiting for execution. The loader establishes their claims before handing off work, protecting shared inputs across queueing, conversion, and upload. An episode releases its claims only after its intermediate results have been uploaded and temporary outputs cleaned up, while other consumers continue to protect the same files. Once the last claim is released, cleanup can reclaim those files and make room for the next input group. **Consumer completion thus governs admission of subsequent inputs.** Retrieval, conversion, and shard publication advance asynchronously around this dependency: they overlap across workers without allowing prefetch to grow independently of the storage that unfinished consumers still hold.

Recover within the same capacity budget (Fig. 70, panel D). Recovery preserves this connection between execution and storage. Stopped attempts release obsolete claims, while lost inputs are reconstructed on an available node through the same admission path before affected episodes retry. Durable conversion-group and publication checkpoints separately identify completed results that job resumption can reuse. Input release after upload and cleanup, and checkpoint-based reuse after restart, are distinct boundaries: an uploaded artifact without recorded completion may still require repeated processing. These rules keep retries within the working-storage budget while preserving reusable progress.

This coordination of shared-input lifetime, admission, and recovery complements prior work on video analysis, multimodal processing, and prefetching (Poms et al., 2018; Chen et al., 2025; Luan et al., 2026; Dryden et al., 2021).

Production evaluation. Tab. 36 summarizes throughput and resource use across three production jobs and recovery across five interrupted jobs. Each job uses 100 nodes with 96 vCPUs and 768 GiB of memory per node. The throughput workload covers 1.30 million episodes and about 18,700 demonstration-hours, with three camera views per episode.

Completed-conversion time efficiency is the share of completed conversion time spent on work that did not need to be repeated after interruption.

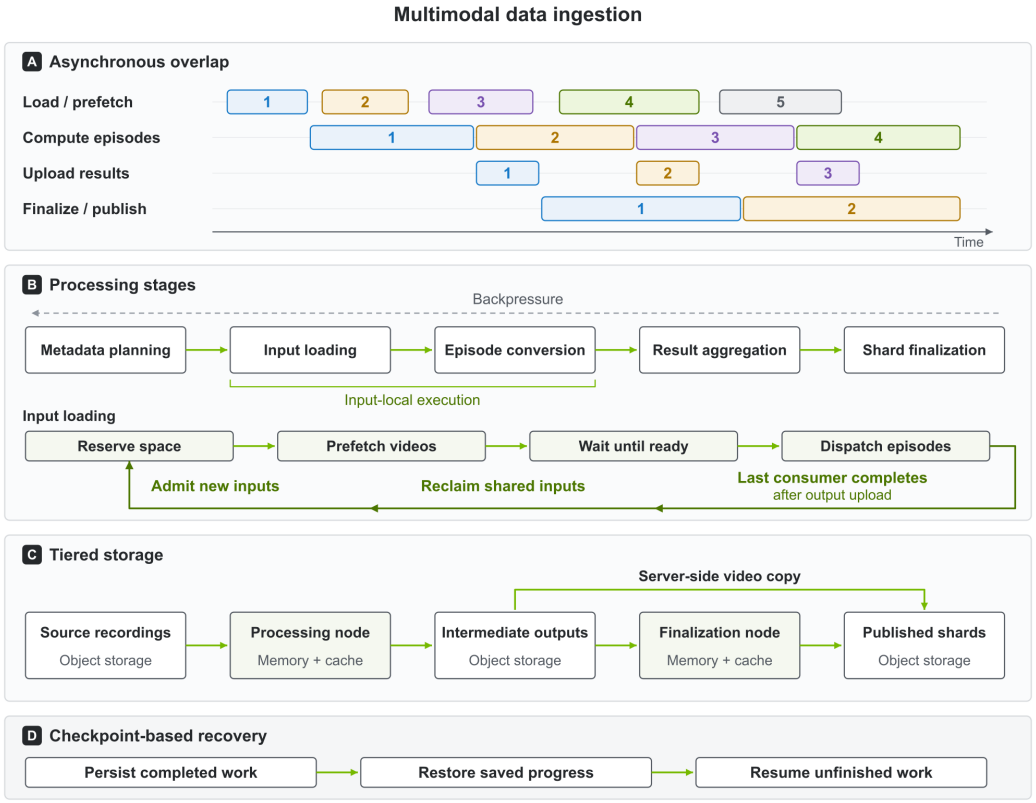


Figure 70: Multimodal data ingestion: (A) asynchronous overlap, (B) processing stages and the feedback from input release to admission, (C) storage tiers, and (D) recovery. Numbers identify shards, and timings are illustrative. Compute/upload overlap occurs across workers.

Table 36: Production evaluation of the data ingestion system. Ranges span jobs. Memory and disk use are per-node means.

Category	Measurement	Result
Throughput and publication (three jobs)	Published demonstration duration per job	∼6,240 h
	Execution time per job	8.0–8.6 h
	End-to-end throughput	722–780 demonstration-hours/h
	Time to first training-ready shard	$\leq$28 min
	Mean worker assignment at 10–80% conversion completion	$>$99.9%
Resource use during active conversion	Mean CPU utilization	97.8%
	Mean memory use / node capacity	335–384 / 768 GiB
	Mean filesystem use / node capacity	80–94 / ∼291 GiB
Recovery (five interrupted jobs)	Completed-conversion time efficiency	89.8–95.6%, pooled 92.4%
	Restart-to-first-publication latency	2.4–5.3 min