DEXSIMWAM: SCALING DEXTEROUS WORLD ACTION MODELS WITH VERIFIED SIMULATION DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning transferable dexterous manipulation policies is constrained by the scarcity of large-scale robot data. Simulation offers scale, but synthesizing dexterous trajectories is substantially harder than for grippers: the data must cover diverse multi-finger contacts and hand–object configurations while satisfying language-specified grasp and motion semantics. We introduce *DexSimWAM*, which combines an agent-guided, retrieval-based synthetic-data pipeline with large-scale World Action Model (WAM) pre-training and task-specific post-training. Given a language-specified task and verifier feedback, the agent guides scene sampling, grasp retrieval, optimization-based path planning, and bounded expansion of a physically validated object-frame grasp bank. Deterministic filters and fixed whole-task verification yield a 16k-hour corpus spanning diverse objects, scenes, grasp styles, and unilateral and bimanual behaviors. On this corpus, DexSimWAM jointly pre-trains visual-dynamics prediction and synchronized arm–hand action generation. Task-specific post-training then adapts the shared WAM with scalable simulation data and targeted real-robot trajectories, supporting 6D object reorientation, functional pouring, articulated-object interaction, long-horizon semantic rearrangement, direct sim-to-real deployment, and physical-domain refinement. Project website: https://dexsimwam-anonymous.github.io/.

1 INTRODUCTION

Recent progress in large robot policies has been driven by large and diverse collections of robot trajectories, which enable end-to-end learning of perception and closed-loop control (Brohan et al., 2022; 2023; Kim et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025). Although dexterous manipulation has made substantial progress on individual contact-rich skills (Liu et al., 2025; Niu et al., 2026), it remains comparatively data-limited. Real-robot collection is expensive, and articulated hands enlarge the action and contact spaces: demonstrations must cover diverse multi-finger contact modes and hand–object configurations while remaining consistent with the semantic intent of the task. These requirements make it difficult to obtain the scale and diversity needed to learn broadly reusable dexterous representations, limiting transfer across objects, contact modes, scenes, and downstream tasks.

Simulation provides a scalable alternative, but synthesizing useful dexterous trajectories is substantially more challenging than generating trajectories for grippers. Existing synthesis methods generate diverse stable grasps, taxonomy-conditioned contacts, or task-specific demonstrations (Wang et al., 2022; Zhang et al., 2024; Chen et al., 2025; Jiang et al., 2024). Scaling such data to World Action Model (WAM) pre-training additionally requires complete arm–hand trajectories that remain reachable, collision-free, contact-stable, and task-valid across many objects, poses, scenes, and visual conditions. We therefore require task-aligned grasping, in which language instructions specify the task-relevant object part, grasp style, and downstream motion constraints. Regenerating and optimizing such task-aligned grasps independently for every randomized scene is both expensive and prone to uneven coverage.

Generating trajectories at scale addresses only the data bottleneck; transferring the resulting experience across downstream tasks also requires a model that captures both how interactions evolve visually and which actions produce those changes. WAMs provide such a formulation by jointly

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

learning visual interaction dynamics and action generation (Goswami et al., 2025; Lyu et al., 2026; Yuan et al., 2026). However, it remains unclear whether physically verified simulation experience can be generated at sufficient scale and diversity, and whether a WAM pre-trained on this experience can transfer effectively across downstream simulated and real-world tasks. This motivates our central question: *Can large-scale, physically verified simulation data yield dexterous WAMs that adapt efficiently and transfer effectively across simulated and real-world tasks?*

DexSimWAM answers affirmatively by combining an agent-guided, retrieval-based simulation-data synthesis pipeline with large-scale WAM pre-training and task-specific post-training. The agent receives a language-specified task together with the grasp-bank summary, preference statistics, and verifier reports. It returns a schema-constrained configuration for scene and grasp sampling, optimization-based path planning, and bounded grasp-bank expansion. The bank stores taxonomy-conditioned object-frame grasp poses that have passed fixed free-hand physics checks. Deterministic modules use the configuration to retrieve grasps and plan complete arm–hand trajectories. A fixed whole-task physics verifier determines which trajectories enter the dataset. Its success and failure reports update the context for the next production round. We use the resulting 16k-hour corpus to jointly pre-train interaction-conditioned visual-dynamics prediction and synchronized arm–hand action generation on a balanced mixture of unilateral and bimanual trajectories. For each downstream task, post-training initializes from the complete shared checkpoint and updates the WAM using task-specific simulation data, real-robot data, or their mixture. This procedure adapts the shared representation to 6D object reorientation, functional pouring, articulated-object interaction, and long-horizon semantic rearrangement.

Our contributions are threefold:

- We introduce a scalable synthetic-data pipeline that covers diverse objects, scenes, and grasp configurations while satisfying the semantic requirements of each task. Given a language-specified task and verifier feedback, an agent guides scene sampling, grasp retrieval, optimization-based path planning, and bounded expansion of a physically validated object-frame grasp bank.
- We construct a 16k-hour corpus of verified dexterous simulation trajectories and jointly pre-train visual-dynamics prediction and arm–hand action generation. The resulting checkpoint serves as an effective initialization for task-specific post-training with simulation data, real-robot data, or their mixture.
- We evaluate DexSimWAM on six unilateral and bimanual dexterous tasks under matched simulation and real-robot settings, including position and visual generalization. We further conduct thorough ablations on pre-training data scale, task-specific adaptation, and the simulation–real composition of post-training data.

2 Related Work

Vision-language-action and world action models. Generalist VLAs scale closed-loop manipulation through heterogeneous robot and web data. Open X-Embodiment, Octo, and OpenVLA establish cross-dataset policy learning (Open X-Embodiment Collaboration, 2023; Octo Model Team et al., 2024; Kim et al., 2024), while π_0 and $\pi_{0.5}$ introduce flow-based action generation and broaden open-world generalization (Black et al., 2024; Physical Intelligence et al., 2025). GR00T N1 and its N1.7 release, together with LingBot-VLA 2.0, further extend pre-training across humanoid, whole-body, and dexterous-hand embodiments (NVIDIA et al., 2025; Wu et al., 2026). WAMs complement direct action prediction with learned interaction dynamics. DreamZero adapts a video-diffusion backbone into a real-time zero-shot policy; Fast-WAM studies video co-training without test-time imagination; AHA-WAM separates low-frequency world planning from high-frequency control; and LaWAM and Faster-WAM provide efficient future conditioning through latent subgoals or selective prediction (Ye et al., 2026; Yuan et al., 2026; Cai et al., 2026; Chen et al., 2026; Zhao et al., 2026). Harness VLA provides a complementary composition strategy by steering a frozen VLA with memory-guided routing and fixed analytic primitives (Zhang et al., 2026a). Closest to our data-centric setting, GraspVLA and MolmoB0T scale synthetic expert trajectories, while synthetic-prior WAM training studies direct zero-shot sim-to-real transfer (Deng et al., 2025; Deshpande et al., 2026; Wang et al., 2026). These results establish synthetic pre-training predominantly for grippers;

DexSimWAM instead scales physically verified trajectories for articulated hands, diverse multi-finger contacts, and unilateral and bimanual dexterous control.

Dexterous manipulation and data generation. DexGraspNet (Wang et al., 2022) introduces a large-scale dataset and efficient synthesis of stable, diverse grasps, while DexGraspNet 2.0 (Zhang et al., 2024) adds a cluttered-scene benchmark and local-geometry-conditioned generation. UniDexGrasp (Xu et al., 2023) couples diverse pose proposals with goal-conditioned execution, whereas UniDexGrasp++ (Wan et al., 2023) improves universal policy learning through a geometry-aware curriculum and iterative generalist–specialist learning. Other work includes taxonomy-conditioned synthesis in Dexonomy (Chen et al., 2025), joint taxonomy, contact, and affordance conditioning in OmniDexVLG (Zhang et al., 2025), scalable keypoint flow matching in KPGrasp (Huang et al., 2026), a foundation controller trained from dexterous motion primitives in DexterityGen (Yin et al., 2025), and simulation-generated bimanual demonstrations in DexMimicGen (Jiang et al., 2024) and UltraDexGrasp (Yang et al., 2026). Our pipeline extends Dexonomy with support-aware filtering, an expandable object-frame grasp bank, agent-configured task planning and sampling, and whole-trajectory physical verification.

Dexterous VLAs and WAMs. Dexterous VLAs increasingly combine heterogeneous visual experience with high-DoF control. EgoScale (Zheng et al., 2026) transfers large-scale egocentric human-video pretraining to dexterous manipulation, while X-DiffVLA (Li et al., 2026) learns a unified diffusion action head across grippers and dexterous hands. XL-VLA learns a cross-hand latent action space, and recent high-DoF VLA post-training combines supervised adaptation, corrective demonstrations, and residual reinforcement learning (Jiang et al., 2026; Zhu et al., 2026). Dexora (Zhang et al., 2026b) natively targets dual-arm, dual-hand control with matched synthetic and real data, while T-Rex (Niu et al., 2026) couples future-visual-latent prediction and action generation with asynchronous high-frequency tactile refinement. Complementary WAMs explicitly model interaction dynamics: DexWM (Goswami et al., 2025) learns dexterous action-conditioned hand–object dynamics from human and robot videos, while LDA-1B (Lyu et al., 2026) jointly learns latent dynamics, policy, and visual forecasting, with gains on dexterous tasks. After paired human–robot meta-training across gripper and dexterous embodiments, WAM-TTT (Feng et al., 2026) adapts only a lightweight memory from unlabeled human videos while keeping the WAM frozen. DexSimWAM instead scales dexterous WAM pre-training with 16k hours of physically verified simulation trajectories and adapts the resulting model to diverse downstream tasks in simulation and on real robots.

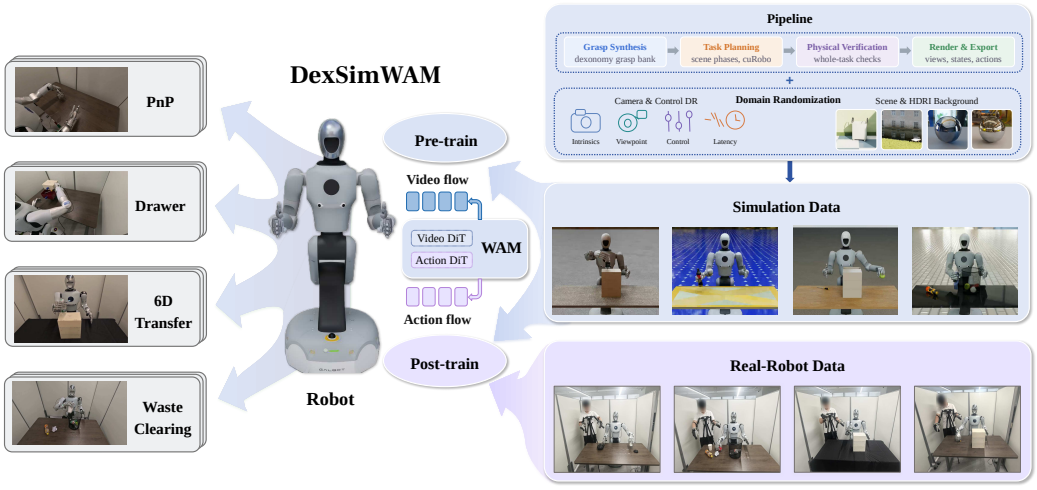


Figure 1: Overview of DexSimWAM. A fixed synthesis pipeline generates diverse, physically verified dexterous trajectories for large-scale WAM pre-training. The shared initialization is then post-trained separately in simulation and on the real robot, and evaluated in the corresponding environment.

3 DEXSIMWAM

DexSimWAM combines three stages: task-conditioned generation of physically verified dexterous trajectories, large-scale WAM pre-training on the resulting 16k-hour simulation corpus, and task-specific post-training with simulated and limited real-robot data. Figure 1 summarizes this progression from verified synthesis to transferable unilateral and bimanual control.

TASK-CONDITIONED DATA GENERATION

Figure 2 summarizes the pipeline. Dexonomy (Chen et al., 2025) generates grasp poses, which are then filtered to form an expandable initial object-frame grasp bank. An Agent-as-Guide module converts a task description and production feedback into a structured configuration for scene sampling, grasp retrieval, bank expansion, and cuRobo (Sundaralingam et al., 2023) planning. MuJoCo (Todorov et al., 2012) verifies each execution, and Isaac Sim (NVIDIA, 2026) renders accepted trajectories. Appendix C gives implementation details.

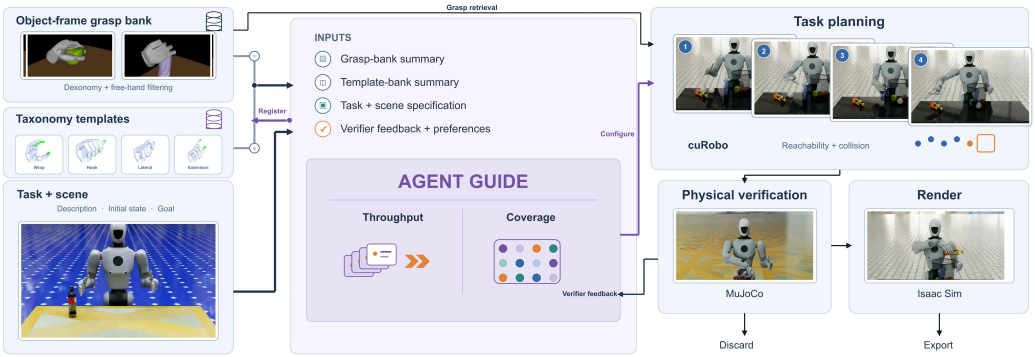


Figure 2: Task-conditioned trajectory production. Agent-as-Guide receives a task description, preference statistics, and verifier reports, then configures scene and grasp sampling, planning, and grasp-bank expansion. Deterministic filters and whole-task verification gate every exported trajectory.

Grasp-pose generation and filtering. We use Dexonomy (Chen et al., 2025) to generate grasp poses from the object geometry, tabletop support, Sharpa hand model, and taxonomy templates. Each proposal contains a pre-contact configuration, an object-relative palm pose, and finger targets. Free-hand replay checks contact, penetration, joint limits, lift, and hold before admission. The agent may request bounded perturbations around useful templates or grasp poses to expand the template or pose bank; every addition passes the same filters and object-frame deduplication. The resulting bank can therefore grow between production rounds.

Agent-as-Guide task planning. The LLM agent receives the task description, scene schema, grasp-bank summary, preference information, and verifier reports. It returns a structured configuration containing the phase graph, goal relations, cuRobo constraints and budgets, and scene and grasp sampling distributions. Preference is context rather than a fixed scheduler: the agent can favor high-yield regions for throughput or weak and under-covered regions for robustness, and can expand the template and pose candidates when the current set is insufficient.

Each retrieved pose proceeds through the task-specific sequence of cuRobo planning stages and offline filters to produce a raw trajectory. Figure 3 shows representative context supplied to the agent, including scene success maps, object heading, successful surface-use evidence, and Drawer state coverage.

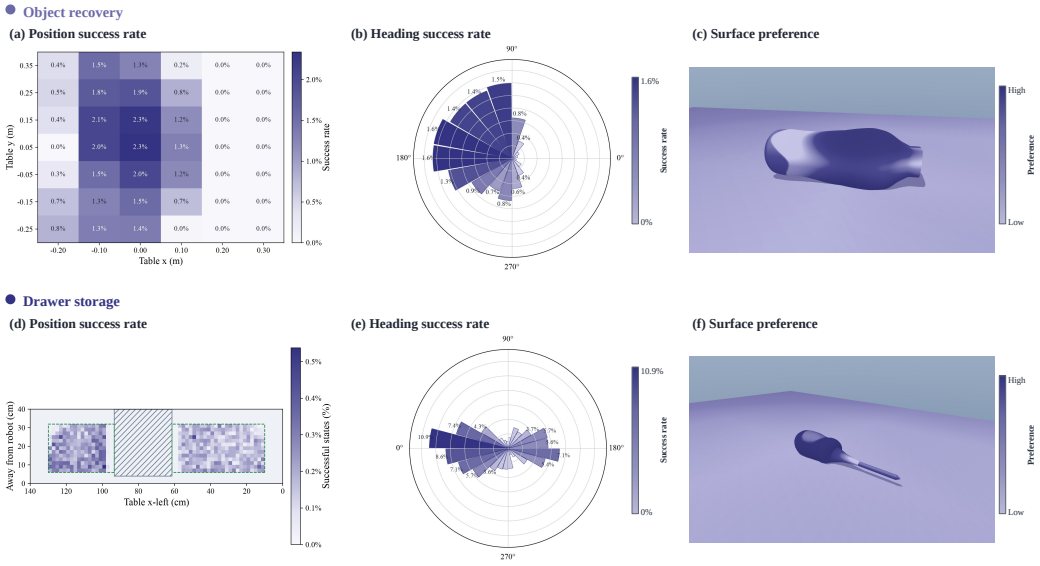


Figure 3: Preference context for agent-guided production. In the top row, each percentage is the bin-wise whole-task quality-success rate: the number of verified successful trials divided by all trials assigned to that position or heading bin. The heading distribution indicates that right-hand object recovery is more successful when the bottle top points to the left. The bottom row summarizes successful Drawer scene states. Here, each percentage is the fraction of all successful states that falls in the corresponding position or heading bin. A screwdriver is easier to grasp when placed laterally relative to the robot, so the largest fractions occur near $0°$ and $180°$. Panels (c) and (f) map grasp-pose contact hotspots from successful trajectories onto a bottle-shaped object and a screwdriver, respectively. Continuous terrain meshes follow the object surfaces; smoothly interpolated displacement along the local surface normal and color encode successful-use evidence, with taller peaks indicating stronger preference.

Simulation verification and rendering. The raw trajectory is replayed and verified in simulation. MuJoCo executes the complete arm–hand trajectory with gravity, contacts, actuator limits, and randomized physical parameters. A fixed whole-task verifier checks approach, intended contact, lift and hold stability, transport, task-specific interaction, release, settling, and retreat. Failed proposals trigger another retrieved grasp, replanning, or scene resampling.

Only successful trajectories are replayed in Isaac Sim using the assets and visual domain randomization in Appendix A. Physical verification is therefore separated from rendering: one valid trajectory can produce multiple visual realizations while preserving its action semantics. Each training sample contains synchronized images from the two head cameras and two wrist cameras together with robot joint states and executed actions. Appendix Figure 12 shows representative rendered trajectories, with eight frames sampled in temporal order from each trajectory.

LARGE-SCALE WAM PRE-TRAINING

The synthesis pipeline produces 16k hours of verified simulation experience for WAM pre-training. A shared vision-language encoder conditions a video diffusion branch and an action diffusion branch on the instruction, synchronized multi-view observations, and robot state. The video branch learns interaction dynamics, while the action branch predicts synchronized arm and articulated-finger action chunks. Pre-training balances task families, object assets, grasp styles, and scene randomization so that the resulting checkpoint provides a broad initialization rather than overfitting to a single downstream task. Architecture, data composition, and optimization details are provided in Appendix D.

CROSS-DOMAIN POST-TRAINING

Each downstream Task-WAM starts from the shared simulation-pretrained checkpoint and is adapted with simulation trajectories, real-robot trajectories, or their combination. Simulation post-training provides inexpensive coverage of poses, clutter, and failure-prone interaction stages; targeted real data corrects residual appearance, calibration, contact, and dynamics gaps. Domain randomization during synthesis supports direct sim-to-real deployment; the effects of mixing simulation and real-robot data during post-training are studied separately in Appendix G. All variants retain the same WAM architecture and task-level success criteria, allowing the experiments to isolate the effects of pre-training data scale, post-training domain, and data composition. We also explored a more adaptive post-training loop driven by rollout feedback. Optional CodeAgent orchestration and validation-gated model updates are detailed in Appendix H. Its task-dependent gains and higher training cost suggest that the fixed recipe presented here offers a favorable balance between effectiveness and efficiency.

4 EXPERIMENTS

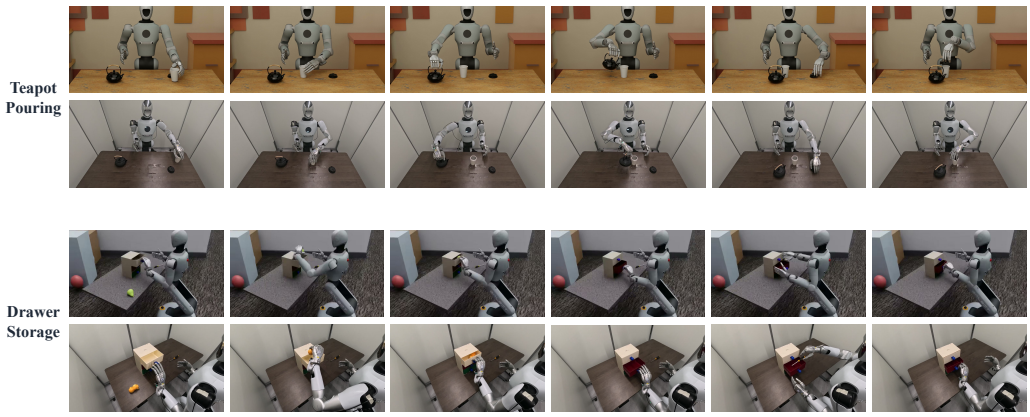


Figure 4: Qualitative simulation and real-robot rollouts for Teapot Pouring and Drawer Storage. Within each task block, the upper and lower rows show simulation and real-robot execution, respectively. The sequences illustrate cup relocation, pouring, and lid placement for Teapot Pouring, and drawer opening and object placement for Drawer Storage. Rollouts for the remaining key evaluations are provided in Appendix G.

PLATFORM AND IMPLEMENTATION

We instantiate DexSimWAM on a dual-arm Galbot G3 equipped with two Sharpa Wave dexterous hands. Figure 4 illustrates matched simulation and real-robot rollouts for Teapot Pouring and Drawer Storage. Additional real-robot rollouts in Appendix G cover the remaining evaluation tasks and visual-generalization examples; together, the main-text and appendix galleries span all six tasks. The benchmark covers unilateral and bimanual manipulation in randomized settings with indoor and outdoor backgrounds, using bottles, cups, teapots, tools, toys, waste objects, a three-layer drawer cabinet, a waste bin, and other task-relevant assets. Scene instances vary object geometry and pose, physical properties, appearance, illumination, and camera pose; Appendix A details the scene, object, robot, and camera assets.

Our WAM adopts a vision-language-conditioned dual-DiT Mixture-of-Transformers, with a video DiT for learning interaction dynamics and an action DiT for generating action chunks. Qwen3-VL-4B-Instruct (Bai et al., 2025) jointly encodes the instruction and multi-view observation, and its projected hidden states condition both branches. Detailed architecture, initialization, and training settings are provided in Appendix D.

A shared warm-up pretraining stage on broad synthetic data jointly optimizes video and action flow matching with Qwen3-VL frozen, producing the initialization for the WAM family. Task-specific

post-training adapts this initialization to each downstream task. Each Task-WAM then retains its own task data, verifier configuration, checkpoint, and registry entry while sharing the semantic tool schema and report interface. Unless otherwise specified, all main results and ablations use standard task-specific post-training without RTC. We evaluate Training-Time RTC separately as an optional latency-oriented variant that pairs prefix-conditioned training with overlapping action-chunk execution at deployment. Qwen3-VL and both DiTs are updated during post-training, while the Wan VAE (Team Wan et al., 2025) remains frozen. At inference, a single video-DiT forward pass over the current observation produces latent features that condition iterative action-DiT denoising, without explicitly generating future video. The standard training configuration is reported in Appendix D, and the optional RTC construction in Appendix F.

Tasks and Evaluation Protocol

We evaluate six tasks, with representative rollouts shown in Figure 4 and Appendix G. *Single-Object Pick-and-Place* transfers one object to a target near the robot's center-front. *Sequential Multi-Object Pick-and-Place* transfers multiple objects in sequence without disturbing earlier placements. *6D-Pose Transfer* lifts an object lying on the table and places it upright on a three-layer cabinet. *Teapot Pouring* moves a cup, pours from a teapot into it, sets down the teapot, and closes the cup with a lid. *Drawer Storage* sorts tabletop tools and toys into assigned drawer layers and closes all drawers. Matched simulation and real-robot rollouts for these two contact-rich, multi-stage tasks are shown in Figure 4. *Tabletop Waste Clearing* moves a waste bin to the target position and deposits all waste objects inside it. Appendix B specifies the task settings, terminal success criteria, and object-generalization factors; Appendix A documents the asset pools.

All tasks use matched simulation and real-robot protocols under three non-ordered evaluation settings: **Basic (Base)**, **Position Generalization (P.G.)**, and **Visual Generalization (V.G.)**. Basic uses familiar scenes, visual appearances, and nominal object-position distributions. Position Generalization keeps the scene appearance and other conditions approximately fixed while expanding the random sampling range of object positions, testing generalization to spatial variation. Visual Generalization retains the Basic task layouts and object-position distributions while changing the tabletop appearance and adding task-irrelevant distractors, testing generalization to visual-environment variation. The three settings characterize distinct evaluation conditions, and no ordering is implied among them. Success is determined by the same dual-view semantic verifier and task-specific physical gates in both domains. We additionally report invocation fidelity, retries and recompositions, completion time, collision rate, and final-pose error where applicable; all rate metrics include trial counts.

Main Results

Comparison to baselines. We compare against $\pi_{0.5}$ (Physical Intelligence et al., 2025), LDA-1B (Lyu et al., 2026), and Fast-WAM (Yuan et al., 2026). All learned methods receive the same Galbot G3 observations, action representation, and task post-training demonstrations where their architectures permit. For the main results, all methods are post-trained on 100 trajectories per task, using exclusively simulation data for simulation evaluation and exclusively real-robot data for real-robot evaluation. Mixed simulation–real post-training is studied separately in Appendix G. All compared methods use their official standard model and training configurations. Figure 5 reports the task-wise comparison in simulation and on the real robot; complete numerical results are provided in Appendix Tables 10 and 11. The task definitions, terminal success criteria, and object-generalization protocol are provided in Appendix B.

Simulation-guided real-world post-training. Warm-up pretraining uses simulation-rendered trajectories. For real-world post-training, simulation-synthesized and verified plans guide the collection of real-robot trajectories. This simulation-first workflow reduces the cost of physical collection while retaining real observations and actions for strong real-robot performance. Following domain-randomization principles (Tobin et al., 2017), we vary camera parameters, image formation, background, illumination, object appearance, physical properties, and task-valid poses during simulation generation. Appendix B defines the matched task and object-generalization conditions. Deployment uses only camera observations, proprioception, and executable actions. The real-robot results in Figure 5 cover the same six tasks and the Base, P.G., and V.G. settings. Across all six tasks, DexSimWAM ranks first or remains competitive with the strongest baseline in simulation, and

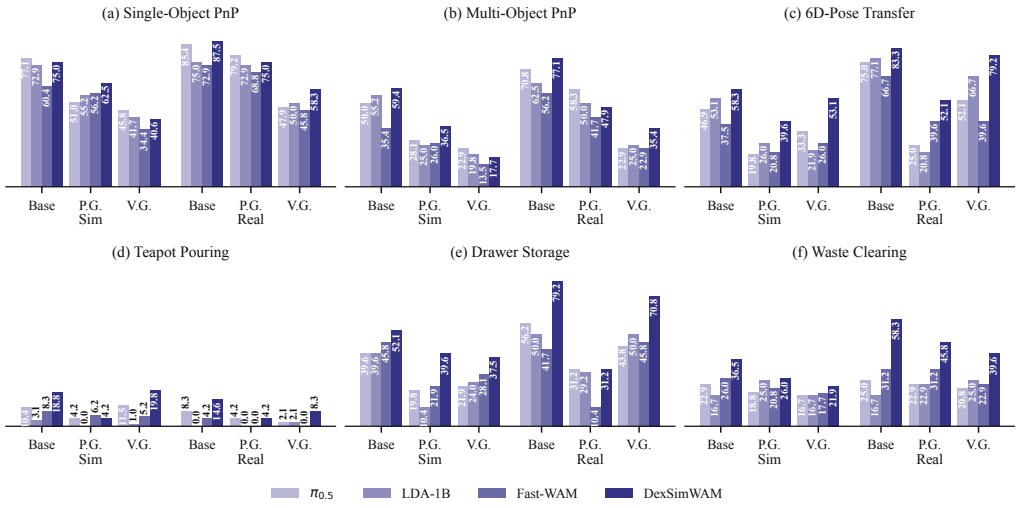


Figure 5: Task-wise simulation and real-robot success rates under the Base, P.G., and V.G. evaluation settings after task-specific post-training.

this advantage largely carries over to real-robot evaluation. The benefit of dexterous-manipulation pretraining is particularly evident on contact-rich and multi-stage tasks, including teapot pouring, drawer storage, and waste clearing, where DexSimWAM demonstrates stronger overall capability than models without large-scale dexterous-manipulation pretraining.

Ablation Studies

We first evaluate the efficiency of task-conditioned data generation, and then examine the effects of pre-training data scale and task-specific adaptation. Additional ablations on post-training optimization and VLM updates, action-head capacity, Training-Time RTC, and the simulation–real post-training data mixture are reported in Appendix G.

Data-generation efficiency. We compare per-scene Dexonomy generation, fixed-bank retrieval, and expandable-bank production under an agent guide (GPT-6 Astra) or a Jev (Almeida, 2026) guide. All methods use the same whole-task verifier and worker schedule. Table 1 reports the initial bank cost, agent-call time, accepted grasps per 8-GPU (RTX PRO 5000) node-hour, and acceptance rate.

Data-generation method	Initial bank GPU-h ↓	Agent call (s) ↓	Accepted / 8-GPU (RTX PRO 5000) node-h ↑	Acceptance ↑
Regenerate Dexonomy grasps per scene	–	–	0.46	2.73%
Fixed bank, uniform retrieval	∼5.93	–	4.47	0.39%
Fixed bank, global grasp preference	∼5.93	–	21.56	1.95%
Agent-guided Expandable bank (GPT-6 Astra, ours)	∼5.93	11.75	**166.1**	**16.11%**
Agent-guided Expandable bank (Jev)	∼5.93	1.40	<u>126.6</u>	<u>12.21%</u>

Table 1: Data-generation efficiency and agent-call time. Arrows indicate the preferred direction; bold and underlined values mark the best and second-best production results.

Global grasp preference improves fixed-bank retrieval over uniform sampling, showing the value of verified production history. Replacing the GPT-6 Astra guide with Jev reduces the mean agent-call time from 11.75 to 1.40 s, but its lower guide accuracy reduces acceptance from 16.11% to 12.21% and production throughput from 166.1 to 126.6 accepted grasps per node-hour.

We therefore use the GPT-6 Astra-guided expandable bank for production. It uses verified history to guide preference sampling and expands the candidate bank when needed. It provides the best balance of verification efficiency, production yield, and adaptation across task scenes.

Pre-training data scale and task-specific adaptation. Table 2(a) isolates the effect of pre-training data scale on real-robot Tabletop Waste Clearing. All three variants undergo matched task-specific post-training. The no-warm-up variant skips pre-training, while the other two follow the same training schedule and differ only in whether they use a fixed 50% subset or the complete data mixture. Performance improves consistently with increasing pre-training data scale across all three evaluation settings.

Table 2(b) separately measures the effect of task-specific adaptation on simulated Sprite-can Pick-and-Place. It evaluates the complete pre-trained checkpoint directly and compares it with the same checkpoint after matched PnP post-training. All three evaluation settings use task scenes unseen during pre-training. Task-specific PnP post-training further improves performance in every setting, with the largest gains under Base and P.G. These results show that task adaptation is effective, while the pre-trained model itself already exhibits meaningful generalization to new scenes and strong robustness to visual variation.

(a) Pre-training data scale: real-robot Waste Clearing

Initialization	Base SR (%)	P.G. SR (%)	V.G. SR (%)
No warm-up	33.3	27.1	25.0
50% pre-training	45.8	35.4	33.3
Full pre-training (ours)	**58.3**	**45.8**	**39.6**

(b) Task-specific adaptation Simulated Sprite-can PnP

Model stage	Base SR (%)	P.G. SR (%)	V.G. SR (%)
Full pre-training only	59.4	56.3	53.1
+ PnP post-training	**84.4**	**68.8**	**55.2**

Table 2: Effects of pre-training data scale and task-specific adaptation. Panel (a) evaluates real-robot Tabletop Waste Clearing after matched task-specific post-training. Panel (b) evaluates the complete pre-trained checkpoint before and after PnP post-training on simulated Sprite-can Pick-and-Place.

Agent-Guided Training Scheduling. We evaluate whether extending agent guidance from data production into a simulation-only training loop improves task-specific adaptation (Appendix H). At each evaluation boundary, the CodeAgent reads simulation rollout and subtask reports and guides the generation of simulation datasets for the next training interval; fixed simulation evaluators measure every checkpoint. On Drawer, agent-guided training reached a higher peak success rate than static scheduling (87.5% vs. 57.3%), although the margin narrowed later in training. However, agent-guided training does not provide a clear improvement on other tasks, such as Teapot Pouring. It also requires substantially more training compute, as detailed in Appendix H; we therefore report it as an exploratory result rather than a primary method.

5 Conclusion

We introduced DexSimWAM, a framework that combines scalable simulation-data generation with large-scale pre-training and task-specific post-training of dexterous World Action Models. Our pipeline generates 16k hours of physically verified simulation trajectories covering diverse task configurations and both single-arm and dual-arm manipulation. The resulting initialization supports task-specific post-training with simulation data, real-robot data, or their combination, as well as direct sim-to-real deployment. Together, scalable verified synthesis, broad WAM pre-training, and flexible cross-domain post-training provide a practical route to general and transferable dexterous manipulation.

6 Limitations and Future Work

The DataGenerator is currently strongest for rigid objects with reliable geometry; deformable or poorly reconstructed objects require richer perception and contact models. Although CodeAgent decisions avoid simulator-privileged observations, using synthesized plans for real-robot collection still depends on accurate geometry, physical verification, and safe execution. Appendix H shows that agent-guided training can reach a higher peak success rate, but does not consistently outperform static scheduling and incurs substantially higher training cost, motivating future work on more stable and cost-effective adaptation, as well as scaling the real-world data flywheel, tool-transition calibration, local agent distillation, and embodied RLVR (DeepSeek-AI, 2025) from physical and semantic verification.

AI USE STATEMENT

Generative AI is an integral component of our method. An LLM-based Agent-as-Guide converts task descriptions, grasp-bank summaries, preference statistics, and verifier feedback into structured configurations for scene and grasp sampling, grasp-bank expansion, and motion-planning constraints. Our WAM further uses pretrained vision-language and generative video components for multimodal conditioning and interaction-dynamics learning. Appendix H additionally studies a broader CodeAgent framework for training coordination, data generation, model registration, and inference-time orchestration. These uses are explicitly described in the main paper and appendices.

Beyond the proposed system, we **used generative AI tools to** assist with literature search, reference formatting, and manuscript editing. We also used generative AI tools to provide feedback on experimental design and assist in interpreting experimental results, with all suggestions and interpretations reviewed by the authors against the experimental evidence. **All AI-assisted research outputs were reviewed and finalized by the authors.** AI-suggested related work was checked against the original sources. The authors determined the final research questions, method design, experimental protocol, reported claims, and conclusions, and take full responsibility for the content of this work.

The following uses are not applicable to this work: AI was not used to formulate or prove mathematical claims or to assist in writing mathematical proofs.

REPRODUCIBILITY STATEMENT

We provide detailed specifications of the simulation setup, robot platform, task definitions, data-generation pipeline, model architecture, training configuration, and evaluation protocol. Appendices A–F document the assets, tasks, verified trajectory-generation pipeline, WAM architecture and training, system implementation, and Training-Time RTC. Appendix G reports complete numerical results and controlled ablations, while Appendix H describes the optional agentic training and inference framework. We also report hardware, training budgets, dataset composition, trial counts, and matched comparison conditions to support independent reimplementation and verification.

REFERENCES

Diogo Almeida. Introducing system one models & jev. TypeSafe AI Blog, September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev. Published September 15, 2026; accessed September 26, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL technical report, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024.

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.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics transformer for real-world control at scale, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

Jisong Cai, Long Ling, Shiwei Chu, Zhongshan Liu, Jiayue Kang, Zhixuan Liang, Wenjie Xu, Yinan Mao, Weinan Zhang, Xiaokang Yang, Ru Ying, Ran Zheng, and Yao Mu. AHA-WAM: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing, 2026.

Jialei Chen, Kai Wang, Kang Chen, Shuaihang Chen, Feng Gao, Wenhao Tang, Zhiyuan Li, Weilin Liu, Zhuyu Yao, Boxun Li, Yuanbo Xu, and Chao Yu. LaWAM: Latent world action models for efficient dynamics-aware robot policies, 2026.

Jiayi Chen, Yubin Ke, Lin Peng, and He Wang. Dexonomy: Synthesizing all dexterous grasp types in a grasp taxonomy, 2025.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

Shengliang Deng, Mi Yan, Songlin Wei, Haixin Ma, Yuxin Yang, Jiayi Chen, Zhiqi Zhang, Taoyu Yang, Xuheng Zhang, Wenhao Zhang, Heming Cui, Zhizheng Zhang, and He Wang. GraspVLA: a grasping foundation model pre-trained on billion-scale synthetic action data, 2025.

Abhay Deshpande, Maya Guru, Rose Hendrix, Snehal Jauhri, Ainaz Eftekhar, Rohun Tripathi, Max Argus, Jordi Salvador, Haoquan Fang, Matthew Wallingford, Wilbert Pumacay, Yejin Kim, Quinn Pfeifer, Ying-Chun Lee, Piper Wolters, Omar Rayyan, Mingtong Zhang, Jiafei Duan, Karen Farley, Winson Han, Eli Vanderbilt, Dieter Fox, Ali Farhadi, Georgia Chalvatzaki, Dhruv Shah, and Ranjay Krishna. MolmoB0T: Large-scale simulation enables zero-shot manipulation, 2026.

Yusen Feng, Bingchen Han, Jiangran Lyu, Kai Liu, Yixin Zheng, Yuxuan Wan, Weiheng Liu, Sun Han, Ruiqin Li, Yulong Zhang, Fangfu Liu, Xuesong Shi, Libin Liu, Yizhou Wang, Zhizheng Zhang, and He Wang. WAM-TTT: Steering world-action models by watching human play at test time, 2026.

Raktim Gautam Goswami, Amir Bar, David Fan, Tsung-Yen Yang, Gaoyue Zhou, Prashanth Krishnamurthy, Michael Rabbat, Farshad Khorrami, and Yann LeCun. World models for learning dexterous hand-object interactions from human videos, 2025.

Yuansen Huang, Jiayi Chen, Haoran Liu, Yubin Ke, Bing Han, Jiangran Lyu, Mi Yan, Li Yi, and He Wang. KPGrasp: Scalable keypoint flow matching for dexterous grasp generation, 2026.

Guangqi Jiang, Yutong Liang, Jianglong Ye, Jia-Yang Huang, Changwei Jing, Rocky Duan, Pieter Abbeel, Xiaolong Wang, and Xueyan Zou. Cross-hand latent representation for vision-language-action models, 2026.

Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. DexMimicGen: Automated data generation for bimanual dexterous manipulation via imitation learning, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model, 2024.

Boyu Li, Chaoyi Xu, Haoqi Yuan, Xinrun Xu, Börje F. Karlsson, Haoran Li, Zongqing Lu, and Dongbin Zhao. X-DiffVLA: X-embodied diffusion action heads for vision-language-action models, 2026.

Xueyi Liu, He Wang, and Li Yi. DexNDM: Closing the reality gap for dexterous in-hand rotation via joint-wise neural dynamics model, 2025.

Jiangran Lyu, Kai Liu, Xuheng Zhang, Haoran Liao, Yusen Feng, Wenxuan Zhu, Tingrui Shen, Jiayi Chen, Jiazhao Zhang, Yifei Dong, Wenbo Cui, Senmao Qi, Shuo Wang, Yixin Zheng, Mi Yan, Xuesong Shi, Haoran Li, Dongbin Zhao, Ming-Yu Liu, Zhizheng Zhang, Li Yi, Yizhou Wang, and He Wang. LDA-1B: Scaling latent dynamics action model via universal embodied data ingestion, 2026.

Dantong Niu, Zhuoyang Liu, Zekai Wang, Boning Shao, Zhao-Heng Yin, Anirudh Pai, Yuvan Sharma, Stefano Saravalle, Ruijie Zheng, Jing Wang, Ryan Punamiya, Mengda Xu, Yuqi Xie, Yunfan Jiang, Letian Fu, Konstantinos Kallidromitis, Matteo Gioia, Junyi Zhang, Jiaxin Ge, Haiwen Feng, Fabio Galasso, Wei Zhan, David M. Chan, Yutong Bai, Roei Herzig, Jiahui Lei, Li Fei-Fei, Ken Goldberg, Jitendra Malik, Pieter Abbeel, Yuke Zhu, Danfei Xu, Linxi Fan, and Trevor Darrell. T-Rex: Tactile-reactive dexterous manipulation, 2026.

NVIDIA. Isaac Sim. Software, 2026. URL https://github.com/isaac-sim/IsaacSim. Documentation at https://docs.isaacsim.omniverse.nvidia.com/latest/; accessed September 26, 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang

Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025.

NVIDIA Corporation. NVIDIA vMaterials 2.4. https://developer.nvidia.com/vmaterials, 2026. Accessed September 14, 2026.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy, 2024.

Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models, 2023.

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.

Poly Haven. Poly haven: The public 3d asset library. https://polyhaven.com/, 2026. CC0 asset library; license information at https://polyhaven.com/license; accessed September 14, 2026.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. cuRobo: Parallelized collision-free minimum-jerk robot motion generation, 2023.

Team Wan, 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. Wan2.2-TI2V-5B model and VAE: https://huggingface.co/Wan-AI/Wan2.2-TI2V-5B.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Weikang Wan, Haoran Geng, Yun Liu, Zikang Shan, Yaodong Yang, Li Yi, and He Wang. UniDexGrasp++: Improving dexterous grasping policy learning via geometry-aware curriculum and iterative generalist-specialist learning, 2023.

Ruicheng Wang, Jialiang Zhang, Jiayi Chen, Yinzhen Xu, Puhao Li, Tengyu Liu, and He Wang. DexGraspNet: A large-scale robotic dexterous grasp dataset for general objects based on simulation, 2022.

Zixing Wang, Kausik Sivakumar, Jinghuan Shang, Yafei Hu, Zhaoming Xie, Ran Gong, Xiaohan Zhang, and Karl Schmeckpeper. Efficient sim-to-real transfer of world-action models from synthetic priors, 2026.

Wei Wu, Fangjing Wang, Fan Lu, He Sun, Shi Liu, Yunnan Wang, Yibin Yan, Yong Wang, Shuailei Ma, Xinyang Wang, Yibin Liu, Shuai Yang, Tianxiang Zhou, Kejia Zhang, Lei Zhou, Cheng Su, Nan Xue, Bin Tan, Han Zhang, Youchao Zhang, Fei Liao, Xing Zhu, Yujun Shen, and Kecheng Zheng. From foundation to application: Improving VLA models in practice, 2026.

Yinzhen Xu, Weikang Wan, Jialiang Zhang, Haoran Liu, Zikang Shan, Hao Shen, Ruicheng Wang, Haoran Geng, Yijia Weng, Jiayi Chen, Tengyu Liu, Li Yi, and He Wang. UniDexGrasp: Universal robotic dexterous grasping via learning diverse proposal generation and goal-conditioned policy, 2023.

Sizhe Yang, Yiman Xie, Zhixuan Liang, Yang Tian, Jia Zeng, Dahua Lin, and Jiangmiao Pang. UltraDexGrasp: Learning universal dexterous grasping for bimanual robots with synthetic data, 2026.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies, 2026.

Zhao-Heng Yin, Changhao Wang, Luis Pineda, Francois Hogan, Krishna Bodduluri, Akash Sharma, Patrick Lancaster, Ishita Prasad, Mrinal Kalakrishnan, Jitendra Malik, Mike Lambeta, Tingfan Wu, Pieter Abbeel, and Mustafa Mukadam. DexterityGen: Foundation controller for unprecedented dexterity, 2025.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination?, 2026.

Jialiang Zhang, Haoran Liu, Danshi Li, Xinqiang Yu, Haoran Geng, Yufei Ding, Jiayi Chen, and He Wang. DexGraspNet 2.0: Learning generative dexterous grasping in large-scale synthetic cluttered scenes, 2024.

Lei Zhang, Diwen Zheng, Kaixin Bai, Zhenshan Bing, Zoltan-Csaba Marton, Zhaopeng Chen, Alois Christian Knoll, and Jianwei Zhang. OmniDexVLG: Learning dexterous grasp generation from vision language model-guided grasp semantics, taxonomy and functional affordance, 2025.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents, 2026a.

Zongzheng Zhang, Jingrui Pang, Zhuo Yang, Kun Li, Minwen Liao, Saining Zhang, Guoxuan Chi, Jinbang Guo, Huan-ang Gao, Modi Shi, Dongyun Ge, Yao Mu, Jiayuan Gu, Rui Chen, Hao Dong, Huazhe Xu, Li Yi, Yixin Zhu, Hang Zhao, Pengwei Wang, Shanghang Zhang, Guocai Yao, Jianyu Chen, Hongyang Li, and Hao Zhao. Dexora: Open-source vla for high-dof bimanual dexterity, 2026b.

Weiheng Zhao, Haoyi Jiang, Xin Shi, Liu Liu, Fan Huang, Zhizhong Su, Wei Sui, and Xinggang Wang. Faster-WAM: Efficient inference-time future conditioning for robust world action models, 2026.

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.

Junlei Zhu, Shenzhe Yao, Chaogui Huang, Wenkai Zhu, Jingwei Peng, Guanqi He, Soren Schwertfeger, Jiahao Chen, and Yide Liu. Towards high-DoF dexterous manipulation through VLA post-training, 2026.

A SCENE, OBJECT, ROBOT, AND CAMERA ASSETS

SCENE AND OBJECT ASSETS

A scene is composed from a tabletop geometry, an indoor or outdoor background, task-relevant objects, optional distractors, and task-specific receptacles or tools. Some object assets are reconstructed from real-object scans and converted into simulation-ready geometry. Rendering assets expose material, texture, reflectance, and color parameters. Figure 6 shows representative renderings of scenes and objects. Environment illumination is randomized by sampling high-dynamic-range images (HDRIs) from Poly Haven (Poly Haven, 2026), whose HDRIs, textures, and 3D models are released under the CC0 license for research and commercial use. Tabletop and ground appearances are sampled from NVIDIA vMaterials 2.4 (NVIDIA Corporation, 2026) and rendered through the NVIDIA Material Definition Language (MDL). For traceability, the rendering metadata records the HDRI filename, material file, exported material name, and source asset pool selected for each sample.



Figure 6: Scene and object assets.

ROBOT AND CAMERA ASSETS

All tasks use a dual-arm Galbot G3 equipped with two Sharpa Wave dexterous hands. The head-mounted SENSING S56B stereo module provides left and right head views. Two SENSING SG3S11AFSK cameras, one on each wrist, provide the left and right wrist views. Together these form the four-view camera rig shown in Figure 7. Camera parameters are domain-randomized during rendering.

Table 3 summarizes the domain randomization applied to physical, robot, and camera parameters during trajectory generation and rendering. Physical and robot/task parameters are sampled before planning and MuJoCo verification, while camera extrinsics are sampled during Isaac Sim rendering.

B TASK LIST, SUCCESS CRITERIA, AND OBJECT GENERALIZATION

We evaluate six task families. Each episode fixes the target objects, initial scene state, goal relations, and verifier before execution. A task is successful only when the complete action sequence finishes and the terminal scene passes the corresponding physical checks; an intermediate grasp or collision-free plan alone is insufficient.

Single-Object Pick-and-Place. One object is sampled in a $0.50 \times 0.30\,\mathrm{m}$ tabletop region. The robot picks it and places it at a target near the center-front of the robot. Success requires the object to be released,

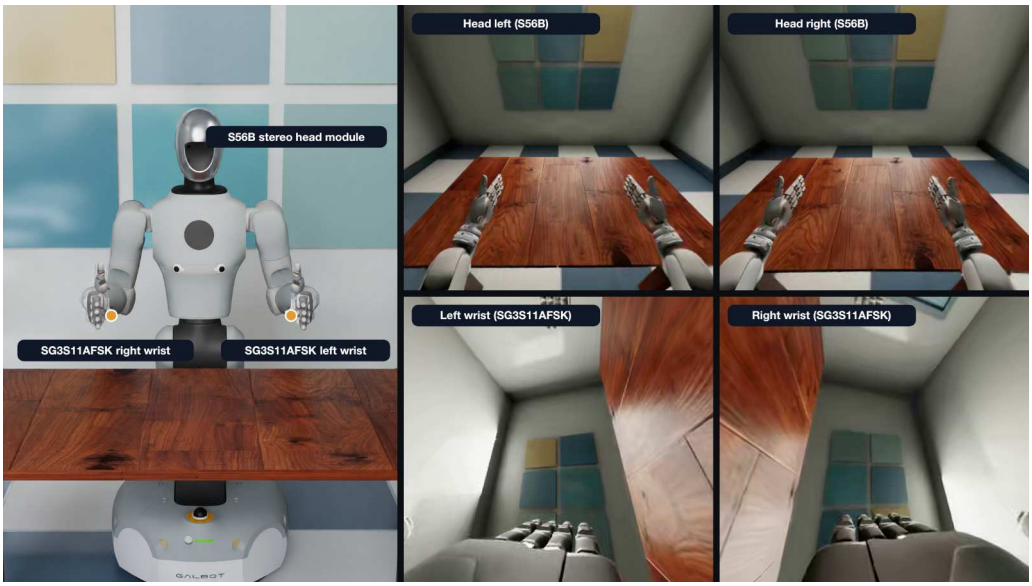


Figure 7: Robot and camera assets.

Group	Parameter	Range	Sampling and scope
Simulation physics	Object mass and inertia	Mass scale $\mathcal{U}(0.8, 1.2)$; inertia scaled identically	Sampled per scene for MuJoCo verification; fixed during Isaac Sim replay
	Contact friction and restitution	Friction scale $\mathcal{U}(0.7, 1.3)$; restitution $\mathcal{U}(0, 0.1)$	Sampled per scene for MuJoCo verification; fixed during Isaac Sim replay
Robot/task geometry	Mobile-base translation and yaw	x, y: ± 0.05 m; yaw: $\pm 10^\circ$	Sampled once and held fixed for the complete trajectory
	Tabletop height	0.78 ± 0.04 m	Sampled once and shared by planning and verification
Robot execution	Phase-speed multiplier	$\mathcal{U}(0.8, 1.2)$	Sampled independently by phase; exported at 50 Hz
Robot dynamics	Link mass/inertia, joint stiffness/damping, and actuator gains	Scale $\mathcal{U}(0.95, 1.05)$ relative to nominal values	Sampled once per trajectory and held fixed during the MuJoCo rollout
Camera extrinsics	Head stereo rig	Pretrain: ± 5 mm, $\pm 1^\circ$; post-train: ± 2 mm, $\pm 0.5^\circ$ per axis	One rigid perturbation per episode
	Left/right wrist cameras	Pretrain: ± 10 mm, $\pm 3^\circ$; post-train: ± 5 mm, $\pm 1.5^\circ$ per axis	Each wrist is sampled independently once per episode
Camera intrinsics	Default pinhole model	Fixed	110° horizontal field of view and centered principal point

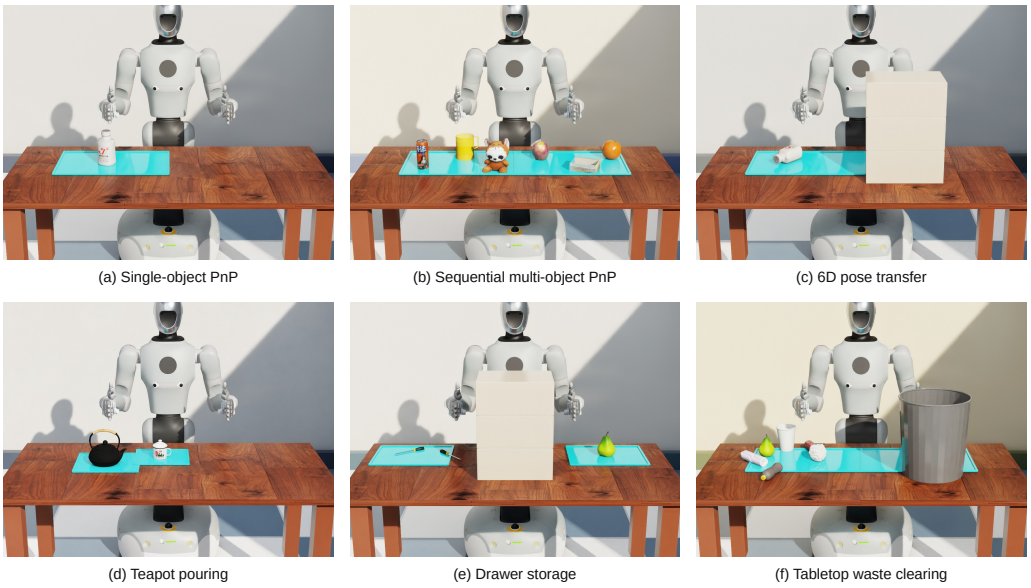
Table 3: Simulation-physics, robot, and camera parameter domain randomization. Physics parameters are randomized during MuJoCo verification and remain fixed during Isaac Sim replay. Camera translation and rotation ranges are symmetric uniform bounds about the calibrated poses.

dynamically settled, and fully contained in the target region; it must not be dropped. Generalization varies object category and instance, stable support pose, initial pose, target region, hand side, and surrounding clutter.

Sequential Multi-Object Pick-and-Place. Multiple objects appear sequentially in a 1.00×0.30 m tabletop region formed by mirroring the single-object region about the table centerline and removing their overlap. The robot selects the hand according to each object's relative position and performs successive pick-and-place operations, placing every object at the target. Success requires all specified objects to be released and stably

contained in the target region. No object may be dropped, toppled outside its allowed pose class, or disturbed after placement. Generalization varies object category and instance, object count and order, stable support pose, initial pose, target region, hand side, spacing, clutter, and the accumulated scene state.

Figure 8 shows one simulation setting for each task family, including all task-relevant objects.



(a) Single-object PnP (b) Sequential multi-object PnP (c) 6D pose transfer

(d) Teapot pouring (e) Drawer storage (f) Tabletop waste clearing

Figure 8: Isaac Sim settings for the six evaluation tasks. Cyan tabletop surfaces mark the task-specific pose-generalization regions and are rendered as part of each 3D scene.

6D-Pose Transfer. The robot picks an object lying on the table and places it upright on top of a three-layer cabinet. Initial object positions are sampled from a $0.515 \times 0.26\,\mathrm{m}$ region. Scene generation proposes fallen poses with a 60°–120° pitch and a full $\pm 180^\circ$ axial-angle range. Success requires the settled object pose to match the target and the object to remain stably upright on the elevated surface. Generalization varies object category and instance, initial position, fallen support pose and axial angle, cabinet pose, target pose, grasp, and hand side.

Teapot Pouring. The robot first moves a cup from the left side of the table to a nearby central position. It then lifts the teapot on the right, aligns the spout with the cup, pours water into the cup, and places the teapot to one side. Finally, it pinches the lid from the left side of the table and places it on the cup. Success requires every stage to complete, no object to be dropped, a pouring motion to be performed, and the lid to remain stably seated on the cup. Generalization varies cup, lid, and teapot category and instance, handle and spout geometry, initial object poses, pouring pose and angle, liquid level, hand assignment, and surrounding clutter.

Drawer Storage. All three drawers are initially closed. Object centers are sampled from two tabletop regions on opposite sides of the cabinet, measuring $0.515 \times 0.26\,\mathrm{m}$ and $0.365 \times 0.26\,\mathrm{m}$; the gap between them is reserved for the cabinet sweep corridor. The tabletop contains screwdrivers, toy models, and other objects. The robot classifies and stores tools in the middle drawer and toy models in the top drawer, then closes every drawer. Success requires every target object to be released and stably contained in its assigned drawer, with the tools and toys correctly classified and all drawers closed at the end. Generalization varies object category and instance, source region, tabletop layout and support pose, cabinet pose, drawer layer, insertion corridor, hand assignment, and surrounding clutter.

Tabletop Waste Clearing. The scene contains a movable waste bin and heterogeneous waste objects sampled in a $1.00 \times 0.30\,\mathrm{m}$ tabletop region. The robot first moves the distant bin to the target position in front of itself, then uses the left and right hands to collect waste from their respective $0.50 \times 0.30\,\mathrm{m}$ half-regions and place it inside the bin. Success requires the bin to be stable at its target and every waste object to be released, settled, and fully contained in the bin. Generalization varies waste category and instance, object count and order, stable support pose, initial pose, bin pose, drop location, hand assignment, and clutter.

C TASK-CONDITIONED DATA-GENERATION PIPELINE

The pipeline produces trajectories from a task description. Dexonomy (Chen et al., 2025) generates taxonomy-conditioned grasp poses and free-hand replay forms an expandable object-frame bank. An Agent-as-Guide interface, implemented by an LLM agent, receives the task description, bank summary, preference statistics, and verifier reports, and returns a structured configuration for scene sampling, grasp retrieval, bank expansion, and cuRobo planning. MuJoCo verifies complete executions and Isaac Sim renders accepted trajectories. Figure 9 summarizes the process.

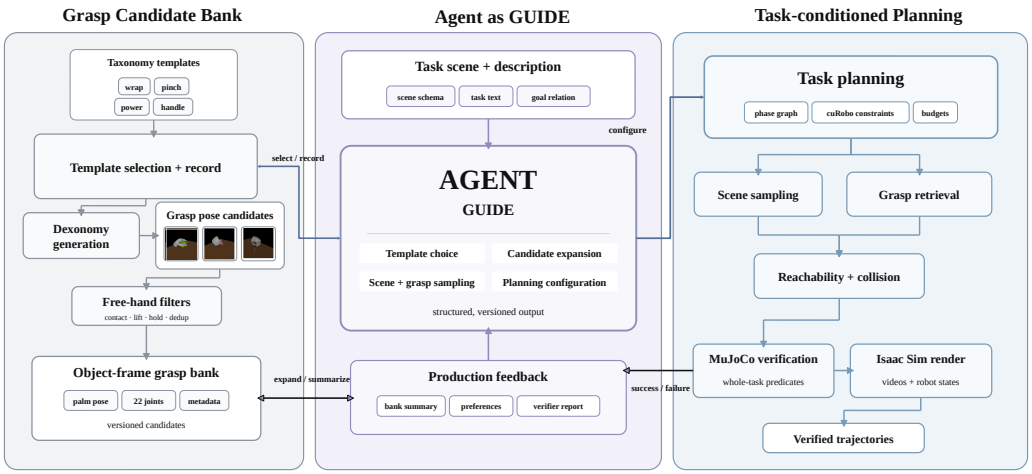


Figure 9: Task-conditioned trajectory production. Agent-as-Guide configures planning and sampling from the task description and verified production feedback; deterministic filters and whole-task verification remain fixed gates.

Algorithm 1 formalizes the complete production pipeline. The agent converts the task description, scene schema, grasp-bank summary, preference information, and verifier report into a schema-constrained production configuration. This configuration specifies scene and grasp sampling, grasp-bank expansion, the task phase graph, goal relations, per-phase cuRobo constraints, and planning and retry budgets. The agent proposes these choices but cannot bypass grasp admission, offline planning checks, or whole-task verification. We denote the production history by H; each entry (r, x, g, Φ, o) records the round, sampled scene state, retrieved grasp, instantiated phase graph, and planning or verification outcome.

Online expansion calls Dexonomy with the agent-selected taxonomy templates, target grasp families, candidate budget, and bounded template or pose perturbations. Generated candidates enter the bank only after the same free-hand physics filters and object-frame deduplication used for the initial bank. For planning, the agent specifies an ordered phase graph and the goal relation, collision constraints, tolerances, time budget, and retry budget of each phase. These fields are instantiated from the sampled scene and retrieved grasp before cuRobo is called. Every planned segment must pass deterministic reachability, collision, continuity, and phase-goal checks; the concatenated raw trajectory is then accepted only by the fixed whole-task MuJoCo verifier. Failure stage and verifier output are added to H, so the next round can revise sampling, planning, or expansion while the physical gates remain unchanged.

GRASP-POSE GENERATION AND FILTERING

Each Dexonomy generation job receives the processed object geometry and a tabletop scene configuration containing the object's metric scale, mass, stable pose, and support plane. It also receives the Sharpa hand model, the associated skeleton, keypoint, and body-group annotations, and a taxonomy grasp template that specifies the intended contact pattern and initial finger configuration. Unlike

Algorithm 1 Agent-guided task-conditioned trajectory production

Require: task description T, asset pool A, expandable object-frame grasp bank B, production rounds R, scene budget N
Ensure: verified rendered dataset D

$D \leftarrow \emptyset$
$H \leftarrow \emptyset$ ▷ production history
for $r = 1, \ldots, R$ **do**
  $P_r \leftarrow \text{PreferenceStatistics}(H, B)$; $V_r \leftarrow \text{VerifierReport}(H)$
  $C_r \leftarrow \text{AgentGuide}(T, \text{SceneSchema}(A), \text{Summarize}(B), P_r, V_r)$
  $\text{ValidateConfiguration}(C_r, T, A)$
  if C_r requests bank expansion **then**
    $E_r \leftarrow \text{ExpansionSpecification}(C_r)$ ▷ templates, perturbations, families, budget
    $G_r^+ \leftarrow \text{DexonomyGenerate}(A, E_r)$
    $B^+ \leftarrow \text{ObjectFrameDeduplicate}(\text{FreeHandFilter}(G_r^+))$
    $B \leftarrow \text{VersionedMerge}(B, B^+)$
  end if
  for $j = 1, \ldots, N$ **do**
    $x \leftarrow \text{SampleTaskScene}(T, A, C_r.\text{scene})$
    $G \leftarrow \text{RetrieveGrasps}(B, x, C_r.\text{grasp})$
    for all $g \in G$ in retrieval order **do**
      $(\Phi, \mathcal{Y}, \mathcal{K}, \beta) \leftarrow \text{InstantiateTaskPlan}(T, x, g, C_r)$
        ▷ instantiate the agent plan for scene x and grasp g
        ▷ Φ: phases; $\mathcal{Y}$: goals; $\mathcal{K}$: constraints; β: planning/retry budgets
      $\pi \leftarrow \emptyset$; $z \leftarrow$ feasible ▷ π: trajectory; z: feasibility/failure status
      for all $\phi \in \Phi$ in phase order **do**
        $\pi_\phi \leftarrow \text{CuRoboPlan}(\phi, \mathcal{Y}_\phi, \mathcal{K}_\phi, \beta_\phi)$
        $z \leftarrow \text{OfflineCheck}(\pi_\phi, \mathcal{Y}_\phi, \mathcal{K}_\phi)$
        if $z \neq$ feasible **then**
          break
        end if
        $\pi \leftarrow \text{Concatenate}(\pi, \pi_\phi)$
      end for
      if $z =$ feasible **then**
        $o \leftarrow \text{MuJoCoReplayAndVerify}(T, x, g, \pi)$
      else
        $o \leftarrow z$
      end if
      $H \leftarrow H \cup \{(r, x, g, \Phi, o)\}$
      if o satisfies the whole-task verifier **then**
        $D \leftarrow D \cup \{\text{IsaacSimRenderAndExport}(x, g, \pi, o)\}$
        break
      end if
    end for
  end for
end for
return D

an isolated floating-object setup, the support geometry remains active when generated grasps are checked.

For each object scene and template, Dexonomy samples points on the object surface and rotates the template about each sampled surface normal. Our default campaign setting uses 1024 surface points, 8 in-plane rotations, 5 initialization epochs, and 100 matching-optimization steps. We vary the taxonomy template, stable object pose, scale variant, and random seed to obtain distinct grasp families; the number of surface samples, in-plane rotations, epochs, optimization steps, and retained candidates per template are adjustable coverage and compute-budget parameters. The optimizer returns a palm pose and 22 finger-joint targets.

Stage	Implemented filter	Acceptance test	Production threshold
Input	Payload validity	Require finite object-local palm poses, unit quaternions, and 22 finger-joint commands	29 values per row; quaternion-norm error $< 10^{-5}$
Free-hand replay	Measured joint limits	Check the simulated joint positions throughout pregrasp, closure, lift, and hold	Maximum limit violation ≤ 0.01 rad
	Penetration guards	Reject excessive self, table, pregrasp–object, or grasp–object penetration	Self ≤ 1 mm; table ≤ 0.2 mm; pregrasp–object ≤ 0.2 mm; grasp–object ≤ 2 mm
	Force-bearing contacts	Require positive normal force on at least two distinct digits after the hold	At least 2 digits; normal force per counted digit $\geq 10^{-5}$ N
	Lift and hold stability	Require the object to be lifted and retained without excessive translation or rotation	Lift ≥ 0.10 m; hold 2 s; slip ≤ 0.01 m; rotation $\leq 10^{\circ}$
Bank construction	Object-frame deduplication	A retained pose must differ from every previously retained palm pose in translation or rotation	Translation ≥ 0.012 m or rotation $\geq 15^{\circ}$

Table 4: Free-hand grasp admission and candidate-bank construction. These checks use the isolated hand and do not include arm IK, robot reachability, or task-motion planning.

The initial template–surface proposals form the seed set. From verified production feedback, the agent may request bounded perturbations of selected templates, object-relative palm poses, and finger targets. Every expanded proposal repeats the same free-hand physical checks and object-frame deduplication as a seed proposal. Accepted proposals are stored with the object identity, hand side, template label, and generator version, allowing one qualified grasp to be transformed to different valid object poses without rerunning the generator.

Candidate-bank admission is performed with the isolated Sharpa hand in the object frame. It deliberately excludes full-robot inverse kinematics, arm reachability, arm collision, and task-motion planning; those checks are applied only after a free-hand grasp is retrieved for a sampled scene. Each candidate first passes payload and pose checks, after which MuJoCo settles the object on a horizontal table and executes the candidate's pregrasp, closure, lift, and hold sequence with gravity and the native hand controller enabled. Table 4 reports the implemented admission tests and thresholds used by the audited Drawer free-hand library.

Candidates that pass the physical replay are deduplicated in the object frame by palm translation and rotation. World-frame copies and variants that change only the closing command do not count as new poses. Each object-local bank snapshot is versioned and may be expanded between production rounds. Figure 10 shows representative retained poses: each row contains one object and eight distinct hand–object configurations.

We analyze successful grasp-pose diversity with PCA in Figure 11. Each descriptor contains object-local palm position, continuous orientation, and 22 Sharpa joint targets. We draw equally sized sets of distinct poses for the left and right hands, sampling without replacement in proportion to successful-use counts, and fit one shared standardized PCA space. PC1 and PC2 explain 29.4% and 13.0% of the variance. Broad, multimodal coverage shows that production retains varied grasps rather than collapsing to one template, enabling retrieval across object poses and scene states.

AGENT-GUIDED TASK PLANNING

Each sampled scene instance has a new task-valid physical configuration: object identity and scale, stable initial and target poses, distractor placement, articulated state, and physical parameters. Geometric constraints prevent penetration, unreachable placements, and invalid target relations. Held-out object and scene configurations are kept separate across dataset splits, so retrieval reuses grasp knowledge without collapsing the diversity of physical task conditions.



Figure 10: Visualization of generated grasp poses for selected objects.

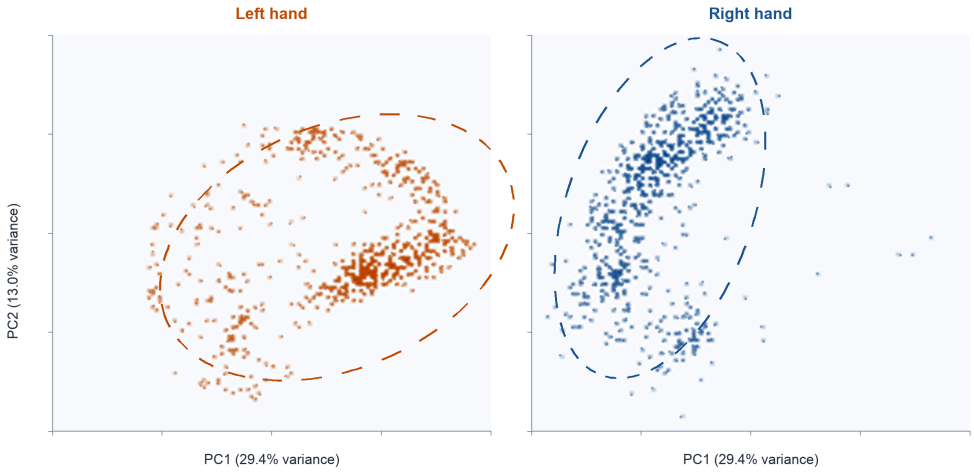


Figure 11: Hand-balanced PCA of successful grasp-pose diversity. Both panels use equally sized pose samples and identical rendering settings. Darker regions indicate greater density; dashed curves enclose 90% coverage.

After each production round, the system aggregates verifier outcomes by scene state, grasp, template, and failure stage. The resulting context contains trial counts, empirical success, coverage, successful surface-use evidence, and representative failures. It exposes evidence to the agent without prescribing a fixed sampling rule.

The LLM agent receives this context together with the task description and a schema for valid production settings. It outputs the phase graph, goal relations, cuRobo parameters and budgets, scene and grasp sampling distributions, and optional bank-expansion requests. For rapid collection it can favor high-yield regions; for robustness it can emphasize weak or under-covered regions. Its output is parsed into a versioned configuration and rejected if it violates the schema or task geometry.

For each configured scene, the system queries the bank by object, taxonomy, hand side, and support pose. Each object-frame grasp is transformed to the measured object pose and checked for full-robot reachability and collision before cuRobo plans the task phases. Agent decisions never bypass grasp admission, collision checking, planning, or physical verification. Figure 3 shows representative context supplied to the agent.

Simulation Verification and Rendering

A collision-free plan is only a proposal. MuJoCo executes the controller commands with contacts, gravity, actuator limits, and randomized physical parameters. Stage-level predicates check approach collisions, intended hand–object contact, lift and hold stability, transport collision and slip, task-specific placement or interaction success, release, settling, and retreat. Only trajectories that satisfy the fixed whole-task verifier are accepted. Failures return to the responsible stage: another grasp for grasp or lift failure, replanning for reachability or path failure, and scene resampling for invalid layouts. The outcomes update the preference statistics used by subsequent scene instances.

Accepted trajectories are replayed in Isaac Sim using the scene and object assets introduced in Section A; camera and appearance randomization do not change the verified scene geometry or trajectory. The exported training sample contains four synchronized camera streams, robot joint states, executed actions, and versioned scene, grasp, plan, and verifier metadata. Figure 12 shows representative rendered trajectories.

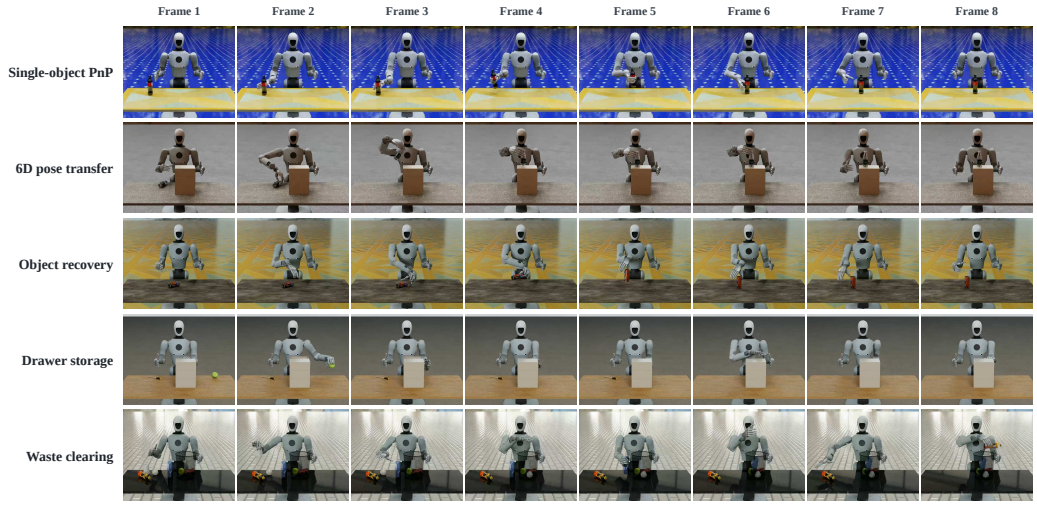


Figure 12: Representative rendered trajectories. Each row shows one physically verified trajectory, and the columns show eight frames in temporal order.

D WAM Architecture, Pretraining, and Post-Training Configuration

Model Architecture

Our DexSimWAM contains three parts: (1) a Qwen3-VL-4B-Instruct multimodal encoder with condition and proprioception projections; (2) a video branch consisting of the Wan2.2 VAE and a Wan2.2-TI2V-5B Video DiT (Team Wan et al., 2025); and (3) a 1B-parameter ActionDiT branch.

The video and action branches form a Mixture-of-Transformers (MoT) architecture, with 30 transformer blocks in each branch.

The WAM receives three current RGB views, a language instruction, the current 58-dimensional robot state, a noised video latent, and a noised 32×58 action sequence. The three 256×256 views are concatenated into a 256×768 image. Qwen3-VL produces 2560-dimensional tokens, which are projected to 4096 dimensions. The robot state is independently projected from 58 to 4096 dimensions and appended as one context token. Both experts cross-attend to this shared context.

The Qwen processor places the three images before the instruction and left-pads each batch. The implementation retains the longest right-aligned suffix that contains no padding for any sample, capped at 512 tokens. Thus, $L = \min(512, \min_i \sum_j m_{ij})$, where m is the Qwen attention mask. This common-length context avoids a partial padding mask in every MoT cross-attention layer. It is batch-dependent: if tokenized inputs have different lengths, the longer inputs discard their oldest prefix tokens. The task-specific post-training run uses fixed-resolution images and one instruction template, so its samples have the same valid token length.

During training, the frozen causal VAE receives a tensor of shape $B \times 3 \times 9 \times 256 \times 768$, where B is the batch size, 3 is the number of RGB channels, 9 is the number of video frames, and 256×768 is the height and width of each three-view image. It produces a latent of shape $B \times 48 \times 3 \times 16 \times 48$, where 48 is the latent-channel dimension, 3 is the number of latent temporal steps, and 16×48 is the latent spatial resolution. The three temporal steps represent the current observation, future frames 1–4, and future frames 5–8. The current step remains clean, while the two future steps are noised with a sampled video flow timestep. A $(1, 2, 2)$ patch embedding preserves the three temporal steps and divides each 16×48 latent map into an 8×24 grid of spatial patches. This gives $3 \times 8 \times 24 = 576$ video tokens, each with width 3072. The action input is projected from 58 to a hidden width of 1024. Its Q/K/V projections map each token to a 3072-dimensional attention space, formed by 24 heads of width 128, and the attention output is projected back to 1024 dimensions. The video and action experts use separate Q/K/V and output projections.

The two branches directly predict video and action flow fields with shapes $B \times 48 \times 3 \times 16 \times 48$ and $B \times 32 \times 58$. These tensors are intermediate model outputs rather than decoded RGB frames or executable robot commands. On the video branch, flow integration produces a video latent, which the VAE decoder converts to RGB frames. On the action branch, flow integration produces a normalized $B \times 32 \times 58$ action sequence, and inverse normalization then yields the absolute joint-position actions sent to the controller. Table 5 summarizes the WAM architecture. Figure 13 illustrates this complete training-time data flow.

Within the video branch, queries, keys, and values all come from video tokens; this operation is video self-attention. Its visibility mask restricts current-video queries to current-video keys and values,

Component	Architecture	Initialization and training state
Multimodal encoder	Qwen3-VL-4B-Instruct; width 2560; at most 512 retained tokens	Official pretrained weights; frozen in pretraining and updated in post-training
Condition projections	Multimodal $2560 \rightarrow 4096$; proprioception $58 \rightarrow 4096$	PyTorch default linear initialization; trained in both stages
Wan VAE encoder/decoder	Encoder: $3 \times 9 \times 256 \times 768 \rightarrow 48 \times 3 \times 16 \times 48$; decoder maps the integrated latent back to RGB video	Official Wan2.2 weights (Team Wan et al., 2025); frozen in both stages
Video DiT	30 layers; hidden width 3072; QKV width 3072; output projection $3072 \rightarrow 3072$; FFN width 14336; 24 heads of width 128; patch size $(1, 2, 2)$	Official Wan2.2-TI2V-5B initialization (Team Wan et al., 2025); trained in both stages
ActionDiT	1B parameters; 30 layers; hidden width 1024; QKV width 3072; output projection $3072 \rightarrow 1024$; FFN width 4096; 24 heads of width 128	Entire expert initialized from scratch; trained in both stages
MoT	30 paired Video/Action blocks; no additional Q/K/V parameters	Uses the two experts' independent projections; no Wan text encoder

Table 5: WAM architecture.

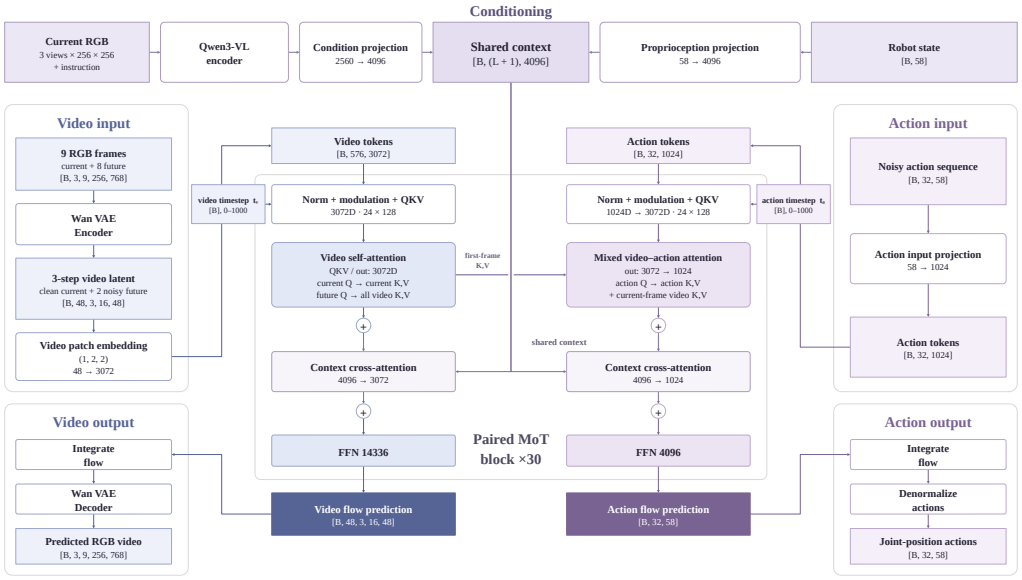


Figure 13: WAM architecture.

Task or operation	Approx. episodes	Approx. hours
Phase-labeled PnP—right hand	455K	3.97K
Phase-labeled PnP—left hand	557K	5.05K
General-object upright transfer	629K	5.65K
Bottle/box upright transfer	34.8K	464
Teapot/mug upright transfer	14.0K	191
Bowl upright transfer	12.1K	79
Paired failure—left-hand attempt	16.6K	106
Paired failure—right-hand attempt	22.5K	130
Bimanual upright transfer	24.6K	230
Close bottom drawer	4.0K	18
Close middle drawer	11.7K	54
Close top drawer	4.1K	20
Open bottom drawer	4.0K	13
Open middle drawer	11.9K	35
Open top drawer	4.1K	14
Total	**1.80M**	**16K**

Table 6: Pretraining data composition. Values are rounded; K and M denote thousand and million.

while future-video queries attend to all video keys and values. Video queries do not attend to action tokens. Separately, action queries attend to all action keys and values plus the current-video keys and values. Each branch then cross-attends to the projected Qwen tokens and the proprioception token.

Pretraining Data

The pretraining mixture contains approximately 1.80M simulation-rendered episodes and 961M recorded frames. Converting frames at 16.67 Hz gives approximately 16K h. Table 6 gives a task and operation partition: pick-and-place is separated by acting hand, object family, failure side, and bimanual execution, while drawer interaction is separated by operation and drawer level. All pick-and-place success data move an object to a target region and require an upright final placement.

Pretraining Process and Configuration

Each training example contains a language instruction, three synchronized camera streams, one current 58-dimensional proprioceptive state, and a 32-step sequence of 58-dimensional absolute joint-position targets. Simulation and robot deployment use the same three-view policy interface.

Each view is resized to 256×256, and the three views are concatenated along width to form one 256×768 RGB image per time index.

For a sample anchored at model step t, the state and current image are s_t and I_t, the action targets are absolute joint positions $(q_t, q_{t+1}, \ldots, q_{t+31})$, and the video targets are $(I_t, I_{t+4}, \ldots, I_{t+32})$. Thus, before VAE encoding, the batched video shape is $B \times 3 \times 9 \times 256 \times 768$. After temporal VAE compression, I_t, $(I_{t+4}, \ldots, I_{t+16})$, and $(I_{t+20}, \ldots, I_{t+32})$ form the three latent steps.

The 58-dimensional states and actions use dataset-specific min–max normalization. The same versioned statistics are used for training, server-side normalization, and deployment-time denormalization.

The current image and instruction are sent to Qwen3-VL-4B-Instruct, producing $B \times L \times 2560$ multimodal tokens; the trainable projections then produce the shared $B \times (L+1) \times 4096$ context. The full video is encoded by the frozen Wan VAE. The current latent remains clean, and the two future latent steps are noised with one sampled video timestep. Qwen3-VL-4B-Instruct and the Wan VAE are frozen during pretraining.

The action tensor has shape $B \times 32 \times 58$ and contains the current action position followed by 31 future positions. The video and action branches use independently sampled flow timesteps. ActionDiT consumes the noised $B \times 32 \times 58$ action tensor, its flow timestep, and the same context. The clean current-video latent supplies the first-frame keys and values visible to the action branch. The two branches return video and action predictions with shapes matching their respective training targets.

Video and action use independent continuous-time flow-matching schedulers with a 1000-point scheduler grid and shift $s = 5$. For each branch, $u \sim \mathcal{U}(0, 1)$ is sampled independently and

$$\sigma = \frac{5u}{1 + 4u}, \qquad t = 1000\sigma, \qquad x_\sigma = (1 - \sigma)x + \sigma\epsilon, \qquad v^* = \epsilon - x. \tag{1}$$

For each sample, the video error is averaged over feature dimensions and future latent positions that have supervision; the clean current latent and tail padding are excluded. At each action step, the squared error is first averaged over the 58 action dimensions. The result is then averaged over the steps in the 32-step chunk that contain ground-truth actions; tail padding after the end of an episode is excluded. The optional RTC variant in Section F additionally excludes its clean action prefix from the loss. With $y(t) = \exp[-2((t - 500)/1000)^2]$ and $y_{\min} = \min_{k \in \{0,\ldots,999\}} y(k)$, the normalized timestep weight is

$$w(t) = \frac{y(t) - y_{\min}}{\frac{1}{1000}\sum_{k=0}^{999}[y(k) - y_{\min}] + 10^{-10}}. \tag{2}$$

Let e_i^v and e_i^a denote the resulting video and action errors for sample i. With independently sampled video and action timesteps t_i^v and t_i^a, the batch losses are

$$\mathcal{L}_{\text{video}} = \frac{1}{B}\sum_{i=1}^{B} w(t_i^v) e_i^v, \qquad \mathcal{L}_{\text{action}} = \frac{1}{B}\sum_{i=1}^{B} w(t_i^a) e_i^a. \tag{3}$$

Each sample draws one noise–time pair per branch, and the reported run uses equal branch weights (Table 7):

$$\mathcal{L}_{\text{pre}} = \lambda_v \mathcal{L}_{\text{video}} + \lambda_a \mathcal{L}_{\text{action}}, \qquad \lambda_v = \lambda_a = 1.0. \tag{4}$$

To accelerate pretraining with frozen encoders, 24 H100 GPUs produce Qwen and VAE features while 24 H100 GPUs train on the resulting feature cache. The run saves a checkpoint every 2k optimizer steps and retains the latest five. Section E describes this producer–trainer pipeline and its throughput.

Post-Training Configuration

Post-training starts from the complete pretraining checkpoint at step 1.082M. The WAM architecture and base video/action flow targets remain unchanged, and a new optimizer and learning-rate scheduler are created at post-training step zero; the pretraining optimizer state is not resumed. Unless otherwise specified, all main results and ablations use this standard non-RTC configuration. Section F separately describes Training-Time RTC as an optional latency-oriented variant.

Optimization parameter	Value
Flow matching	Independent video/action schedules; 1000-point scheduler grid; shift $s = 5$; one noise–time pair per sample and branch
Loss weights	Video 1.0; action 1.0
Optimizer	NVIDIA Apex FusedAdam in AdamW mode; $\beta = (0.9, 0.95)$, $\epsilon = 10^{-6}$
Learning rates	10^{-4} for the entire `action_model` group, including both experts and projections; 10^{-5} for other trainable parameters
Weight decay / gradient clipping	0.01 / global norm 1.0
Schedule	Cosine with minimum LR 5×10^{-7}; 1k optimizer-step warm-up
Global batch	24 trainer ranks $\times$ micro-batch 6 $\times$ accumulation 1 $= 144$
Precision / distribution	BF16; DeepSpeed ZeRO-2 without CPU optimizer offload
Expert gradient checkpointing	Disabled
Model checkpointing	Every 2k optimizer steps; retain the latest five checkpoints
Hardware	24 trainer + 24 frozen-feature producer NVIDIA H100 80GB GPUs
Random seed	42
Training steps	1.082M optimizer steps

Table 7: Pretraining optimization settings.

The post-training data domain and composition are specified by each experiment, with the simulation–real mixture studied separately in Appendix G. Real-robot trajectories are collected by adapting simulation-synthesized plans to the physical scene.

Unlike pretraining, the Qwen3-VL multimodal backbone is trainable and its hidden states are computed online. The Qwen-to-Wan projection, proprioception projection, Video DiT, and ActionDiT are also updated. The Wan VAE remains frozen. Gradients from the video and action losses therefore update the multimodal backbone and both experts end to end.

The input shapes remain the same as in pretraining: three 256×256 camera views, nine video frames, a 58-dimensional state, and a 32×58 action sequence. The task data uses absolute joint positions normalized with its own min–max statistics. Camera observations are sampled every three dataset steps, and future video frames every four robot steps. Table 8 summarizes the post-training configuration.

E Training-System Efficiency and Throughput

Pretraining with frozen conditioning features

Pretraining caches the frozen Qwen3-VL hidden states and Wan VAE latents, so training ranks construct neither encoder and read conditioning features from disk. On top of that cache, the final $48\times$H100 path tunes the ZeRO-2 and NCCL communication configuration, raises the micro-batch to 16 with activation checkpointing restricted to the first 20 video blocks, adds a local cache of decoded episode parquet records, and switches to a distribution-preserving sample-stream cache with shard background prefetch that keeps the original frame sampling distribution. Table 9 gives the resulting speedup at each stage.

Stage	Speedup	Cumulative
Online baseline	–	$1.00\times$
Frozen feature cache and communication tuning	$2.78\times$	$2.78\times$
Micro-batch 16, selective checkpointing, episode cache	$1.14\times$	$3.17\times$
Sample-stream cache and shard prefetch	$1.12\times$	$3.56\times$

Table 9: The final $48\times$H100 pretraining path, in the order the components were added. The factors are consecutive throughput ratios of this path ($2.7789 \times 1.1393 \times 1.1234$) and multiply to the $3.557\times$ end-to-end speedup over the online baseline.

Optimization parameter	Value
Initialization	Complete WAM checkpoint at pretraining step 1.082M; new optimizer and scheduler at post-training step zero
Trainable modules	Qwen3-VL backbone, Qwen-to-Wan and proprioception projections, Video DiT, and ActionDiT; Wan VAE remains frozen
Flow matching	Same base flow targets and video/action objectives as pretraining; full supervised action-chunk loss without RTC prefix conditioning; 1000-point scheduler grid; shift $s = 5$; video/action weights 1.0/1.0
Optimizer	AdamW; $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.01
Learning rates	5×10^{-7} for the multimodal backbone; 10^{-5} for both experts and the two projections
Schedule / training length	Cosine with minimum LR 10^{-7}; 1k-step warm-up; 60k optimizer steps
Global batch	16 GPUs $\times$ micro-batch 8 $\times$ accumulation 2 $= 256$
Precision / distribution	BF16; Accelerate with DeepSpeed ZeRO-2, without optimizer or parameter offload
Gradient checkpointing	Enabled for all 30 Video DiT and 30 ActionDiT blocks
Checkpointing	Every 2k steps; permanent model checkpoints at 10k, 30k, and 60k steps
Hardware / seed	2 nodes $\times$ 8 GPUs / 42
Gradient clipping	Global norm 1.0

Table 8: Post-training configuration.

The online baseline runs Qwen3-VL, the Wan VAE, and full-expert recomputation inside the step, at 2.44 s/step and 157 samples/s; the final path reaches 0.6877 s/step and 558.42 samples/s. The deployed arrangement instead keeps 24 cards producing features and 24 cards training inside one non-restarting job, which sustains 0.560 s/step and 257 samples/s end to end. NVDEC decoding ($1.824\times$ producer throughput) and standalone producers that avoid a persistent collectives slow state belong to that arrangement only and are not part of the product above.

POST-TRAINING WITH A TRAINABLE ENCODER

Post-training unfreezes Qwen3-VL, so the pretraining feature cache no longer applies and the encoder runs inside the training step. Two experiments with identical data, micro-batch 8, and global batch 256 give a full-step median of 2.799 s before and 2.515 s after the optimizations, a throughput of 91.46 to 101.77 samples/s ($1.113\times$), a `model_time` median of 2.828 s to 2.522 s ($1.121\times$), and a `data_time` median of 61.5 ms to 0.171 ms ($360\times$). Only the data path and the aggregate comparison are separately measured; the remaining items are qualitative.

The data path decodes with `torchvision_av`, persistent workers, prefetch, and pinned memory. `video_input_range=uint8` removes a per-step GPU reduction, Qwen `forward_features` skips LM logits over an unused 151k-token vocabulary, and structured Flash SDPA avoids the slow arbitrary-mask backend. BF16 ZeRO-2 with expert and mixed-attention checkpointing fits micro-batch 8, and the NCCL `Simple` setting and large ZeRO bucket carry over from the H100 tuning. Startup uses `CompactStepIndex`, a pre-generated `meta/stats.json`, a rank-0 mmap checkpoint load, and skips the DiT preload that the full checkpoint overwrites, which together lower startup memory, index construction, and multi-node I/O for 36–46 GB of weights.

F TRAINING-TIME RTC

All main results and ablations use standard post-training without RTC unless explicitly noted. We separately study training-time Real-Time Chunking (RTC) (Black et al., 2025) as an optional latency-oriented variant. It pairs prefix-conditioned post-training with overlapping action-chunk execution at deployment to reduce waiting time and maintain continuity across chunks; it is not part of the default post-training pipeline.

TRAINING-TIME ACTION CONDITIONING

Each inference pass produces an action chunk of length $H = 32$. Adjacent chunk origins are separated by a request stride $S = 16$, leaving at most $d_{\max} = H - S = 16$ model-action steps during which the preceding chunk can cover inference delay.

For each training example, we sample a delay $d \in [0, d_{\max}]$. The first d actions are supplied as a clean, fixed prefix, while flow matching is applied only to the remaining postfix actions. The WAM thus learns to complete a chunk consistently with the current observation, instruction, robot state, and a prefix that has already been committed to execution. Although the prefix receives no direct flow-matching loss, it conditions the postfix through the action expert's attention. This construction matches deployment, where the prefix is formed by the overlap from the preceding chunk.

The optional RTC variant requires token-wise flow timesteps within an action chunk rather than one shared timestep per example. We extend ActionDiT accordingly without adding trainable parameters and initialize it from the same warm-up WAM checkpoint as the default non-RTC model.

DELAY DISTRIBUTION AND PREFIX CONDITIONING

The training delay distribution is derived from end-to-end measurements on the physical robot. With $H = 32$ and $S = 16$, the measured delay has median six model-action steps, 95th percentile nine steps, and maximum 13 steps. We sample d from a mixture of 80% empirical request-delay estimates and 20% uniform samples over $[0, 16]$. The empirical component emphasizes typical deployment latency, while the uniform component covers zero-delay requests and uncommon short or long delays within the supported range.

At inference, the Server represents the overlap prefix in the same normalized action space used in training and performs the standard five-step forward-Euler flow sampling procedure. It clamps the clean prefix, with timestep zero, both before and after every flow step. This prevents numerical integration from altering actions already committed by the preceding chunk, while allowing the postfix to attend to the prefix. RTC therefore adds only per-token timestep assignment and prefix overwrites; it introduces neither additional model forward passes nor trainable parameters. The deployed Server and Client use the same observation preprocessing and normalized action representation as the WAM training pipeline.

ASYNCHRONOUS DEPLOYMENT PROTOCOL

The Client issues a continuation request after $S = 16$ logical steps of the active chunk while continuing to execute its remaining actions. The overlapping segment is sent as a candidate prefix. Before generation, the Server selects a conservative prefix length d_{est} as the maximum measured delay among the six most recent requests; this determines how many overlap actions are clamped. During initialization, we use $d_{\text{est}} = 12$ until measured delays are available.

When the response arrives, the Client measures the actual elapsed number of model-action steps, d_{elapsed}. Thus, d_{est} determines how many leading actions of the new chunk are fixed during generation, while d_{elapsed} determines the first action of the new chunk that the robot executes. If a continuation result arrives after the preceding chunk has been exhausted, the Client holds the last command, refreshes the observation, and issues a prefix-free request rather than executing a temporally stale postfix.

RTC EVALUATION PROTOCOL

The model without RTC is the default configuration used by the main results and other ablations. The optional RTC variant starts from the same WAM initialization and matches task data, optimizer-step budget, robot hardware, whole-task success criteria, and test set. Only RTC conditioning during task-specific training and the corresponding chunk-handoff protocol differ. Section G reports success and end-to-end completion time from invocation to verified termination.

G ADDITIONAL RESULTS

This section reports complete numerical main results, qualitative rollouts, ablations, and other experimental results beyond those presented in the main paper. It includes WAM initialization, post-training, action-head, Training-Time RTC, and simulation–real data-mixture studies, followed by qualitative and quantitative video-generation results. All comparisons use the same task definitions, success predicates, evaluation splits, and reporting conventions as the main experiments. Unless a row explicitly changes the data or optimization budget, comparisons keep them matched. The reference WAM uses the architecture and training configuration in Section D.

COMPLETE TASK-WISE RESULTS

Tables 10 and 11 report the exact success rates summarized in Figure 5 for all six tasks, three evaluation settings, and four methods.

Task	Setting	$\pi_{0.5}$	LDA-1B	Fast-WAM	DexSimWAM
Single-Object PnP	Base	77.1	72.9	60.4	75.0
	P.G.	51.0	55.2	56.2	62.5
	V.G.	45.8	41.7	34.4	40.6
Multi-Object PnP	Base	50.0	55.2	35.4	59.4
	P.G.	28.1	25.0	26.0	36.5
	V.G.	22.9	19.8	13.5	17.7
6D-Pose Transfer	Base	46.9	53.1	37.5	58.3
	P.G.	19.8	26.0	20.8	39.6
	V.G.	33.3	21.9	26.0	53.1
Teapot Pouring	Base	10.4	3.1	8.3	18.8
	P.G.	4.2	0.0	6.2	4.2
	V.G.	11.5	1.0	5.2	19.8
Drawer Storage	Base	39.6	39.6	45.8	52.1
	P.G.	19.8	10.4	21.9	39.6
	V.G.	21.9	24.0	28.1	37.5
Waste Clearing	Base	22.9	16.7	26.0	36.5
	P.G.	18.8	25.0	20.8	26.0
	V.G.	16.7	16.7	17.7	21.9

Table 10: Complete simulation success rates (%) corresponding to the task-wise main results in Figure 5.

Task	Setting	$\pi_{0.5}$	LDA-1B	Fast-WAM	DexSimWAM
Single-Object PnP	Base	85.4	75.0	72.9	87.5
	P.G.	79.2	72.9	68.8	75.0
	V.G.	47.9	50.0	45.8	58.3
Multi-Object PnP	Base	70.8	62.5	56.2	77.1
	P.G.	58.3	50.0	41.7	47.9
	V.G.	22.9	25.0	22.9	35.4
6D-Pose Transfer	Base	75.0	77.1	66.7	83.3
	P.G.	25.0	20.8	39.6	52.1
	V.G.	52.1	66.7	39.6	79.2
Teapot Pouring	Base	8.3	0.0	4.2	14.6
	P.G.	4.2	0.0	0.0	4.2
	V.G.	2.1	2.1	0.0	8.3
Drawer Storage	Base	56.2	50.0	41.7	79.2
	P.G.	31.2	29.2	10.4	31.2
	V.G.	43.8	50.0	45.8	70.8
Waste Clearing	Base	25.0	16.7	31.2	58.3
	P.G.	22.9	22.9	31.2	45.8
	V.G.	20.8	25.0	22.9	39.6

Table 11: Complete real-robot success rates (%) corresponding to the task-wise main results in Figure 5.

QUALITATIVE REAL-ROBOT ROLLOUTS

Figure 14 provides real-robot rollouts for the evaluation tasks not shown in the main-text gallery. Its PnP rows show representative object transfers used in the Single-Object and Sequential Multi-Object Pick-and-Place evaluations, followed by 6D-Pose Transfer and Tabletop Waste Clearing. Figure 15 additionally shows real-robot Visual Generalization rollouts for 6D-Pose Transfer and Teapot Pouring. Together with Figure 4, these figures span all six evaluation tasks while separating the additional task coverage from the visual-generalization examples.



Figure 14: Representative real-robot rollouts for the evaluation tasks not shown in the main-text gallery. The first four rows show PnP transfers with a bottle, bowl, cup, and pear; the remaining rows show 6D-Pose Transfer and Tabletop Waste Clearing.

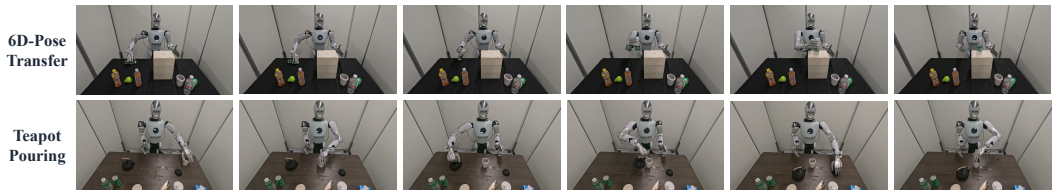


Figure 15: Representative real-robot rollouts under Visual Generalization (V.G.) for 6D-Pose Transfer (top) and Teapot Pouring (bottom).

POST-TRAINING METHOD AND VLM UPDATES

The full post-training pipeline loads the complete pretraining checkpoint, creates a new optimizer and scheduler, keeps the Wan VAE frozen, and updates the multimodal encoder, both condition projections, Video DiT, and ActionDiT. It uses both the video and action flow-matching losses. We compare our full pipeline with freezing the VLM, increasing the VLM learning rate by a factor of three, and removing the video loss while retaining the action loss. All variants use standard non-RTC post-training with the same task data, global batch, optimizer-step budget, and real-robot Single-Object Pick-and-Place evaluation suite. As shown in Table 12, our full pipeline achieves the highest success rate across all three evaluation settings, indicating that joint VLM updating at the base learning rate and the video-training objective are both important for effective task-specific adaptation.

Post-training variant	VLM update	Base SR (%)	P.G. SR (%)	V.G. SR (%)
Frozen VLM	frozen	62.5	52.1	43.8
VLM LR $\times 3$	trainable ($3\times$ LR)	29.2	27.1	20.8
Action loss only	trainable (base LR)	37.5	27.1	25.0
Ours	trainable (base LR)	**87.5**	**75.0**	**58.3**

Table 12: Post-training method and VLM-update ablation on real-robot Single-Object Pick-and-Place.

ACTION-HEAD SIZE

The reference ActionDiT is a 30-layer, approximately 1B-parameter expert with hidden width 1024, QKV width 3072, and FFN width 4096. All three ActionDiT variants are randomly initialized and trained from scratch, whereas the paired Video DiT in every variant is initialized from the same official Wan2.2-TI2V-5B weights (Team Wan et al., 2025). All other initialization and training conditions are identical. Smaller and larger heads retain the 58-dimensional action representation, 32-step action chunk, multimodal context, training data, optimization budget, and hardware. All variants are evaluated in simulation on Single-Object Pick-and-Place under the Base setting, and training time is the wall-clock time required to complete the common optimizer-step budget. As shown in Table 13, increasing the action head from Small to Base improves success from 42.2% to 70.3% while adding 3 h 05 min of training time. Scaling from Base to Large yields a smaller additional gain of 6.3 percentage points, adds 1 h 52 min of training time, and increases the parameter count by approximately 59%. We therefore use Base as the reference architecture because it captures most of the observed accuracy while retaining a substantially smaller action head and a shorter training time than Large.

Action head	Width	Layers	Params.	Training time ↓	Success (%)
Small	768	24	$\sim$0.58B	**49 h 49 min**	42.2
Base (ours)	1024	30	$\sim$1.02B	52 h 54 min	70.3
Large	1280	36	$\sim$1.62B	54 h 46 min	**76.6**

Table 13: Action-head capacity and wall-clock training time on simulated Single-Object Pick-and-Place under the Base evaluation setting.

TRAINING-TIME RTC TASK EFFICIENCY

We compare the default WAM trained and deployed without RTC with the optional Training-Time RTC variant on the real-robot Drawer Storage task under the Base setting. Both variants start from the same checkpoint and use matched task data, optimizer-step budget, robot hardware, and success criteria. Task efficiency is measured by whole-task success and end-to-end completion time, from invocation to verified termination. As shown in Table 14, Training-Time RTC reduces completion time from 68.2 s to 47.0 s, a 31.1% reduction, while success decreases from 79.2% to 75.0%. We therefore treat RTC as an optional latency–success trade-off rather than part of the default post-training pipeline.

Method	Success (%)	Completion time (s)
Default WAM w/o RTC	**79.2**	68.2
Optional WAM w/ Training-Time RTC	75.0	**47.0**

Table 14: Optional Training-Time RTC task-efficiency ablation on real-robot Drawer Storage under the Base evaluation setting.

SIMULATION–REAL POST-TRAINING DATA MIXTURE

The pretraining mixture in Section D is entirely simulation-rendered. This ablation varies only the simulation–real composition used for matched task-specific post-training and evaluates real-robot Pick-and-Place with a Sprite can and a pear under the Base setting. Real trajectories in the mixed

settings are collected from simulation-synthesized plans through the pipeline in Section C. All variants use the same total number of post-training samples and optimizer steps. As shown in Table 15, success increases from 45.8% with simulation-only post-training to 83.3% with an equal simulation–real mixture and 87.5% with real-only post-training. The 50:50 mixture therefore recovers most of the real-only performance while retaining simulation data, trailing the real-only result by only 4.2 percentage points.

Sim : real post-training data	Real-robot success (%)
100 : 0	45.8
50 : 50	83.3
0 : 100	**87.5**

Table 15: Simulation–real post-training data-mixture ablation on real-robot Pick-and-Place with a Sprite can and a pear under the Base evaluation setting.

VIDEO GENERATION AND CROSS-VIEW CONSISTENCY

This section presents the video-generation results of the WAM video branch.

Pretraining. Figure 16(a) shows synchronized multi-view predictions for a bimanual manipulation trajectory. From top to bottom, the views are the left head camera, left wrist camera, and right wrist camera; for each camera, Prediction is shown first, followed by Ground truth sampled from the same-task test set. Figure 16(b) shows bowl manipulation, middle-drawer closing, and top-drawer opening in the left head camera. All results use the pretrained model.

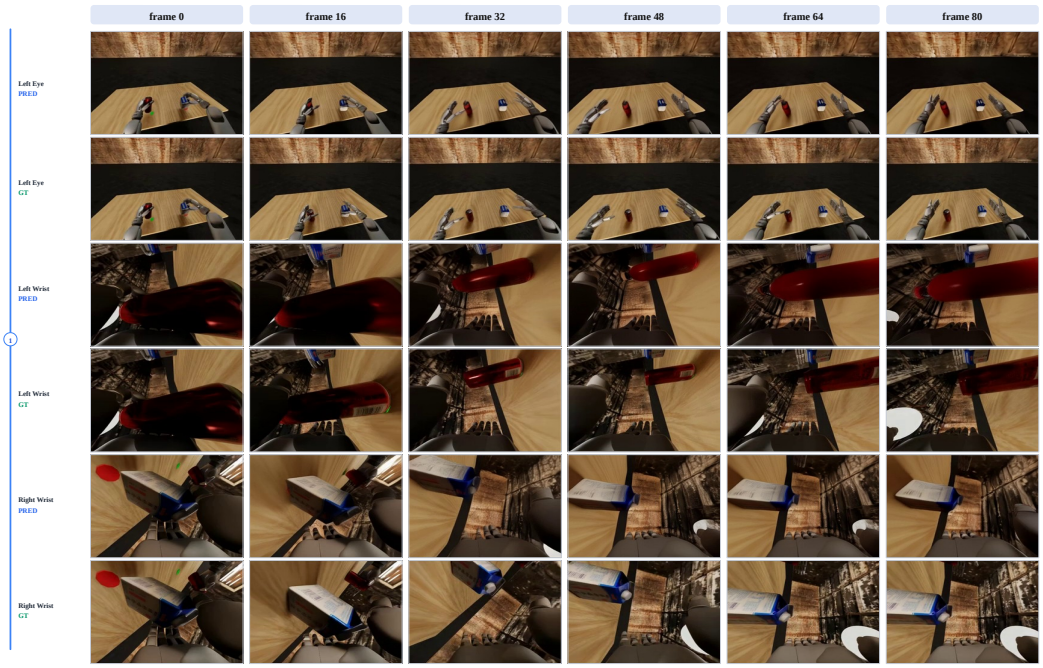
Post-training. Figure 17(a) uses a synchronized multi-view layout to show an object-conditioned Pick-and-Place trajectory. Figure 17(b) shows drawer storage, screwdriver manipulation, and Pick-and-Place in the left head camera. All results use the model post-trained for the corresponding task.

H AGENTIC TRAINING AND INFERENCE FRAMEWORK

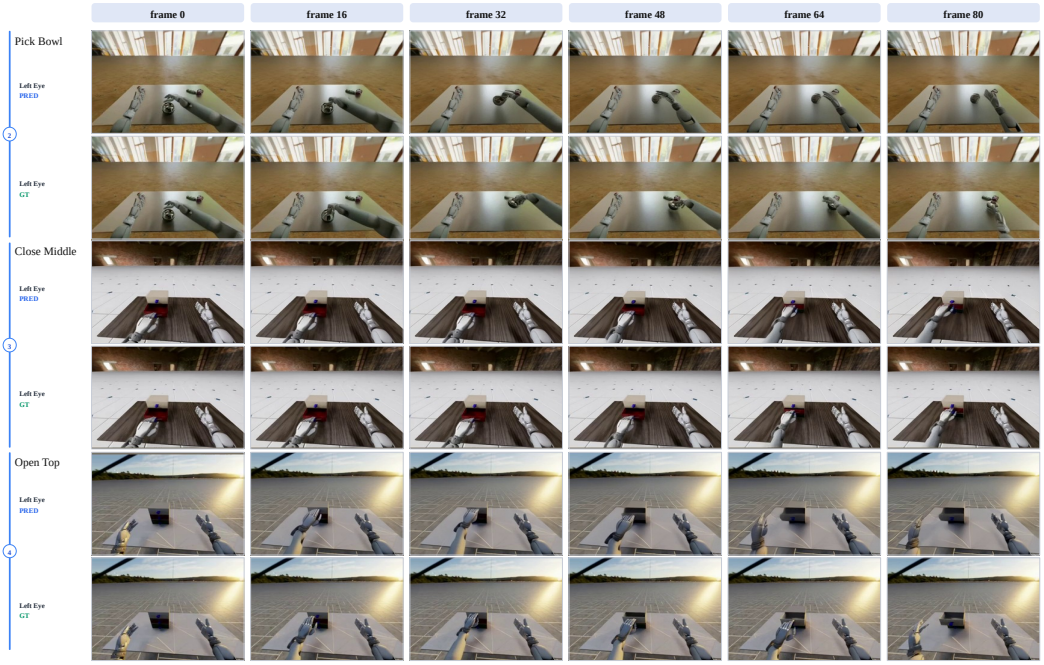
Recent advances in code agents have attracted growing interest in their application to manipulation tasks. Our central takeaway is that a CodeAgent can provide unified coordination across inference and training: it routes among multiple Task-WAMs, prepares both a simulation-data synthesis plan and a real-robot data-collection plan when initiating training for a new task, and uses subtask success rates measured in simulation to retain the current data or request one or more new simulation datasets with targeted subtask compositions during training. When new data are requested, the selected synthesized dataset may augment the current training data or replace part or all of the current simulation dataset. DexSimWAM uses verified execution outcomes to propose and select targeted data plans, while requiring gains on targeted substages to remain consistent with whole-task success under a fixed criterion.

BASIC INTRODUCTION

The system begins with a small family of learned, task-scoped executors and our data-generation pipeline introduced in Section 3, a callable primitive-based system that prepares executable trajectories for simulation data generation and real-robot collection. The high-level agent routes supported requests to Task-WAMs for multi-phase closed-loop execution and initiates post-training when the family must be extended or strengthened. Before training, the agent prepares complementary plans for synthesizing simulation trajectories and collecting physical robot trajectories, producing an initial mixed task dataset from both domains. For each training task, the system also defines a subtask decomposition whose success rates can be evaluated in simulation. During training, these measurements determine whether the current simulation data are retained or one or more candidate simulation datasets are synthesized with dynamically chosen subtask compositions. Data-side evaluation compares the generated candidates, and the selected new data may augment or replace the current simulation data, while the initially collected real-robot data remain fixed. A fixed whole-task success criterion and regression tests remain unchanged throughout the process and determine whether



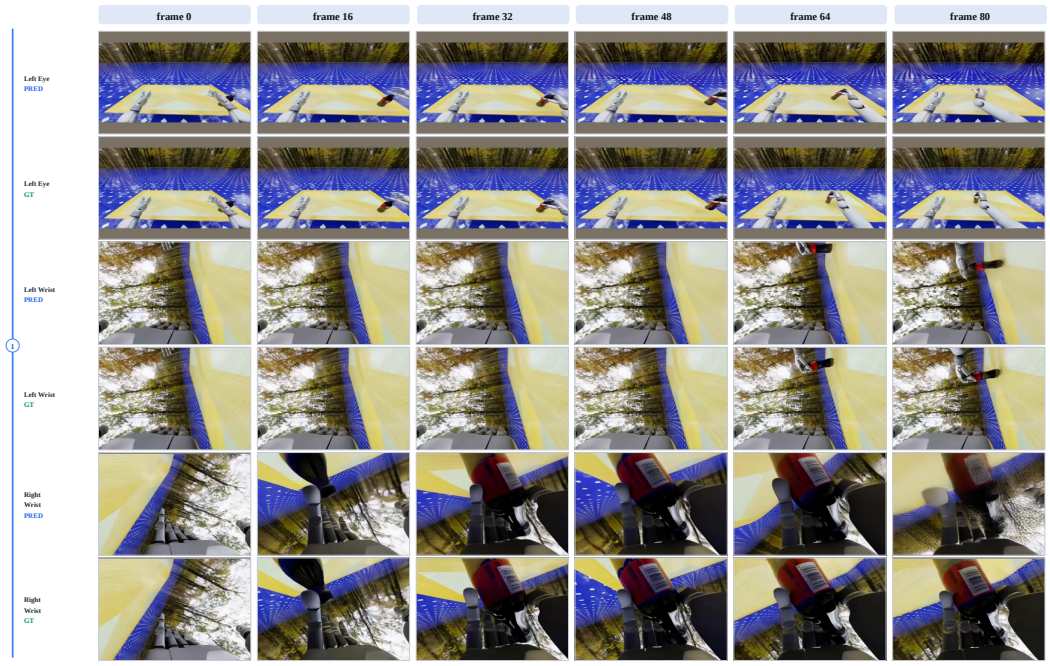
(a) Synchronized multi-view predictions.



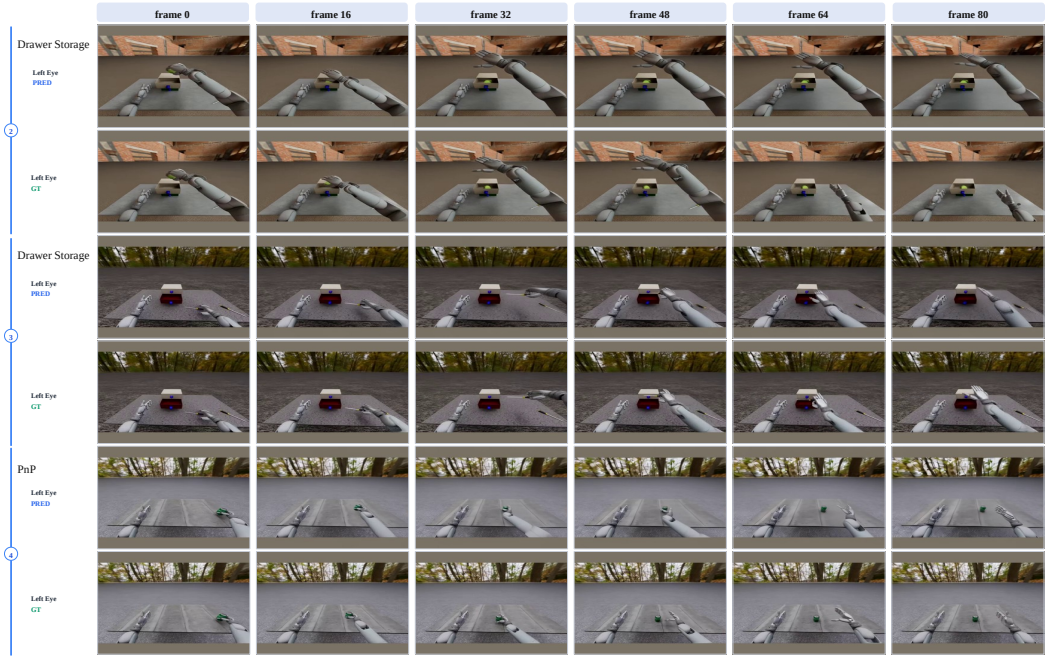
(b) Task coverage in the left head camera.

Figure 16: Pretraining video generation. Columns show frames 0, 16, 32, 48, 64, and 80. Panel (a) pairs prediction and ground truth for the left head, left wrist, and right wrist cameras. Panel (b) shows three manipulation tasks in the left head camera.

a trained candidate can enter the deployed WAM family. Agentic-WAM thus assigns instruction following, task progress understanding, data preparation, and training coordination to the high-level agent, while Task-WAMs provide coherent manipulation execution.



(a) Synchronized multi-view predictions.



(b) Task coverage in the left head camera.

Figure 17: Post-training video generation. Columns show frames 0, 16, 32, 48, 64, and 80. Panel (a) pairs prediction and ground truth for the left head, left wrist, and right wrist cameras. Panel (b) shows three post-training tasks in the left head camera.

Agentic-WAM couples closed-loop Task-WAM execution with a success-rate-driven post-training loop, as illustrated in Figure 18. The CodeAgent selects and composes registered Task-WAMs at

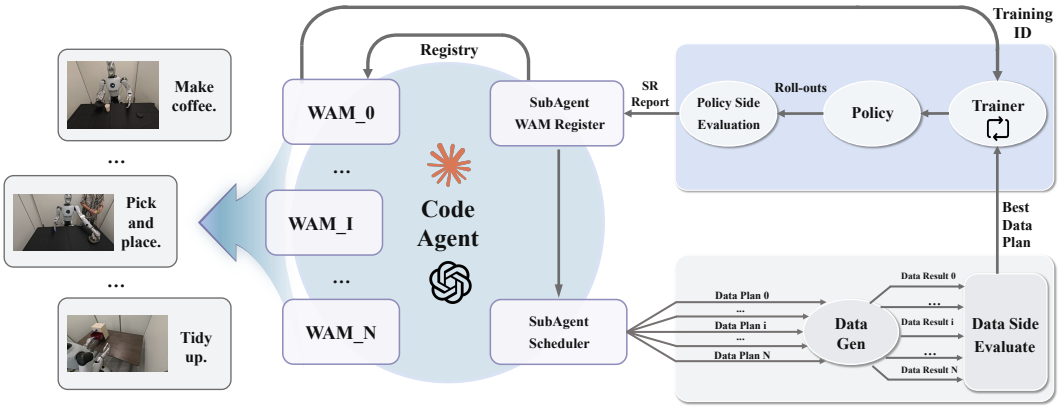


Figure 18: CodeAgent-guided WAM orchestration and improvement. The CodeAgent routes tasks among registered WAMs and launches improvement with a training ID. The Scheduler proposes data plans; generation and data-side evaluation select the best one for training. Policy-side rollouts return success-rate (SR) reports to the WAM Register, closing the registry-update loop.

inference time. At training initiation, it coordinates simulation-data synthesis and real-robot data collection; during training, it uses simulation-evaluated subtask success rates to retain the current data or dynamically propose one or more candidate simulation-data plans. Data-side evaluation selects among the generated candidates, and the chosen dataset may augment or replace the current simulation data. Only model candidates that pass fixed whole-task and regression validation advance the versioned registry.

Inference-Time Orchestration

Each registry entry associates a Task-WAM with its callable task schema, checkpoint and version, training lineage, validated capability range, and capability strength. The capability range records the task conditions under which the WAM has been validated, including relevant objects, scene conditions, task stages, and evaluation settings. Capability strength summarizes validated reliability within that range using whole-task and subtask success statistics; it is not an unconstrained score assigned by the agent.

At each decision boundary, the CodeAgent grounds the instruction against the current scene state, task progress, and verifier feedback. It filters the registry by capability range and then uses the registered capability strengths to select the most reliable compatible Task-WAM for the current conditions. The resulting call, such as `pose_transfer`, `restore`, or `pour`, specifies the relevant objects, goals, affordances, grasp style, and interaction constraints. If no registered WAM satisfies the required conditions, the CodeAgent returns an unsupported-task decision and may initiate the training workflow described below, in which simulation and real-robot data preparation occurs before training rather than inside inference routing.

Each WAM is a vision-language-conditioned policy that executes all phases of one complete task. During training, a video diffusion head predicts the interaction-conditioned visual future while an action diffusion head learns synchronized arm and articulated-finger action chunks from the shared context. At inference time, the video head processes the clean current observation once to condition iterative action-head denoising, and the resulting chunks are executed until the task-level success predicate, failure condition, or maximum execution horizon is reached; no future video is explicitly generated. The CodeAgent is invoked at task boundaries and after a Task-WAM returns. It uses the updated state and verifier outcome to continue with the current WAM, route to another compatible WAM, retry, recompose the remaining task, or terminate execution.

TRAINING COORDINATION

Given the subtask decomposition established during training initiation, the CodeAgent maintains a shared training context containing the current WAM version, simulation-evaluated per-subtask success rates, whole-task evaluation results, previous validation outcomes, the current simulation-dataset composition, data-generation history, training schedule, and training lineage. This context persists across training rounds so that changes are based on measured task performance rather than isolated failures.

At each coordination point, the current WAM is evaluated on the subtasks in simulation. From their success rates, the CodeAgent may retain the current data or dynamically propose one or more candidate simulation-data plans for the next training interval, adjusting their subtask compositions and generation budgets. The DataGenerator synthesizes and verifies trajectories for the proposed plans, which may allocate more generation budget to weaker subtasks while retaining coverage of stronger subtasks to limit regression. Data-side evaluation compares the generated candidates, after which the selected dataset may be appended to the existing simulation data or replace selected or complete portions of it. Both the adaptive signal and any newly synthesized data are simulation-only: the inner training loop does not require repeated real-robot evaluation or initiate another round of physical data collection.

The CodeAgent aggregates the subtask success report with training and validation reports to decide whether to continue with the current data, request and compare one or more candidate simulation datasets, append or replace data, stop training, or submit the model candidate for registration. Accepted decisions may update the simulation-data generation configuration, dataset manifest, training schedule, training YAML, and necessary training code inside an isolated workspace. The task definition, subtask success criteria, held-out whole-task evaluator, regression suite, safety gates, and checkpoint-admission rule remain fixed.

TRAINING INITIATION AND WAM REGISTRATION

Training initiation begins by decomposing the requested task into subtasks and, when needed, finer-grained execution stages using the task description, expected execution sequence, and empirical task boundaries. Each subtask is paired with a success criterion that can be evaluated in simulation so that training coordination can use its success rate directly. This decomposition defines the units used to specify the composition of newly synthesized datasets and to organize evaluation; it does not replace the fixed whole-task success criterion used for final validation.

The SolutionRegister compares the requested task with the registered WAM family and determines whether training should update an existing Task-WAM, broaden the registered range of an existing Task-WAM, or initialize a new task-scoped member. It produces a versioned training solution specifying the target WAM, task and subtask definitions, initialization checkpoint, required training changes, and validation contract. The CodeAgent then prepares two complementary data plans. The simulation-data synthesis plan specifies the task scenes, object and state distributions, trajectories, and collection quotas to be generated and verified in simulation. The real-robot data-collection plan specifies the physical task settings and executions to be collected on the robot, using simulation-synthesized and verified motions where appropriate. Each requested data plan states whether its data are rendered in simulation or collected on the real robot from a simulation-synthesized plan. For a new physical scene, the plan first requests a scan; an object absent from the asset pool can likewise be scanned to provide geometry for a simulation asset. The DataGenerator processes one plan at a time and returns verified simulation trajectories, synthesis-guided real-robot trajectories, or their mixture, together with yield and failure information. Executing both plans produces the initial simulation and physical trajectories used to initiate training. During training, simulation-evaluated subtask success rates may trigger additional rounds of simulation-only data synthesis. In each such round, the agent dynamically proposes one or more candidate plans with targeted subtask compositions; data-side evaluation selects a generated dataset, which may augment or replace the current simulation data. No new physical trajectories are collected in this inner loop. Registering the training solution and preparing its data initiate the Trainer; they do not yet admit the resulting model to the deployed family.

After training, the PolicySideEvaluator evaluates the candidate using the fixed whole-task criterion and the regression suite for previously registered capabilities. A candidate is admitted only if it satis-

fies the registration criteria fixed before training. The resulting registry entry records the callable task schema, supported capability range, validated capability strength, per-subtask performance profile, checkpoint and version, and training lineage. Updating an existing Task-WAM revises this metadata only for conditions supported by the new validation results; creating a Task-WAM adds a new versioned entry.

The capability metadata connects training back to inference. The CodeAgent uses the registered range to determine which Task-WAMs are applicable to the current situation and uses their validated strengths to route among compatible candidates. Consequently, registration exposes not only a callable model but also the evidence needed for situation-dependent orchestration.

RESULTS

We use GPT-6 Astra as the underlying large language model for the CodeAgent in all agentic experiments reported below.

Orchestration Results. We evaluate orchestration using two single-hand pick-and-place Task-WAMs, one for each hand. By routing between these two models, the CodeAgent enables continuous dual-hand pick-and-place execution. During the experiment, an operator repeatedly places an object at a random location on either side of the table. The robot must pick the object and place it in the target region at the center of the table. After each successful placement, the operator removes the placed object and randomly positions the next object. The CodeAgent observes the current camera views and robot proprioceptive state to route each step to the appropriate left- or right-hand pick-and-place Task-WAM. This orchestration achieves a per-object pick-and-place success rate of 72.9%. Qualitative execution sequences are shown in Figure 19.

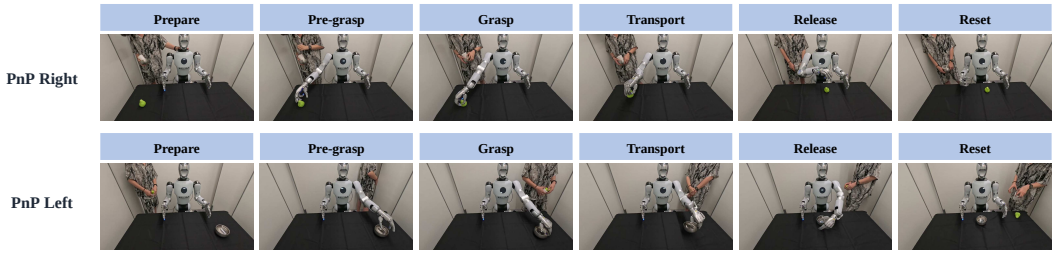


Figure 19: Qualitative orchestration results for continuous dual-hand pick-and-place. The CodeAgent routes between the right-hand (top) and left-hand (bottom) Task-WAMs according to the current observation and robot state. Each row shows the prepare, pre-grasp, grasp, transport, release, and reset stages of one routed execution.

Agent-Guided Training Scheduling Results. We isolate the drawer-opening and drawer-closing portion of Drawer Storage, covering Close, Open-middle, and Open-top. This experiment uses only simulation data for both training and evaluation. At each evaluation boundary, the CodeAgent receives the simulation-evaluated success rates of the three subtasks and may retain the current data or dynamically propose candidate simulation-data plans for the next training interval. Proposed plans specify the generation proportions assigned to the three subtasks; after data-side evaluation, the selected newly synthesized dataset may augment or replace the current simulation data. The static baseline instead uses a dataset with a predefined composition. Detailed records of how the CodeAgent modifies the training schedule are available on the project website.

As shown in Figure 20, agent-guided training attains a higher peak success rate and outperforms the predefined-data baseline at several evaluation boundaries. However, the agent-guided training run uses 851.8 GPU·h of training compute, compared with 563.8 GPU·h for the static baseline—an increase of 288.0 GPU·h, or 51.1%. The gains are not consistent throughout training and are not commensurate with this additional compute. We therefore present agent-guided training as an exploratory result rather than a primary method or core contribution.

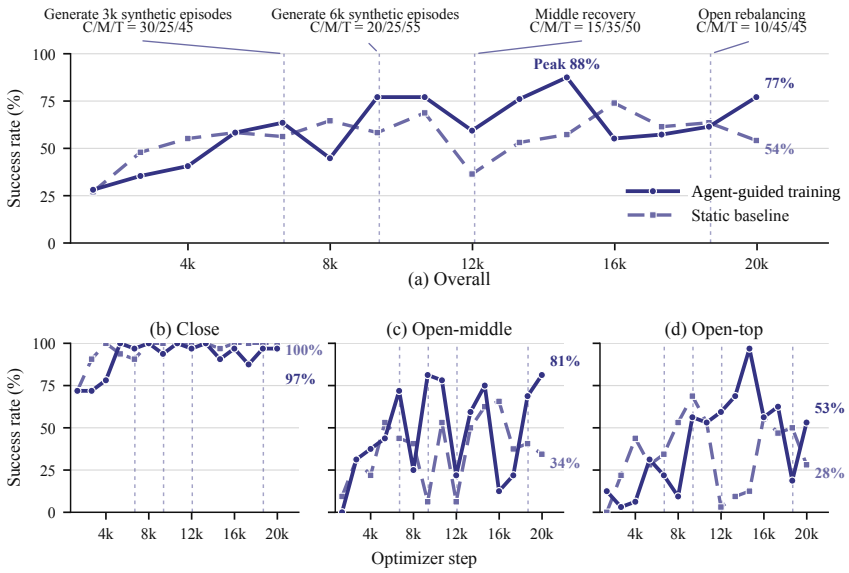


Figure 20: Simulation-only agent-guided training scheduling on Drawer Storage. (a) Aggregate evaluation success rate across optimizer steps for agent-guided training and a sample-matched static baseline. C/M/T denotes the synthetic-data generation proportions assigned to Close, Open-middle, and Open-top. (b–d) Success rates for the three simulation evaluation subtasks.

Future Directions. Our preliminary experiments show that agent-coordinated WAMs can handle flexible tasks and synthesize targeted simulation datasets from evaluation feedback. Future work should improve the stability of this closed-loop process and extend it to sustained cross-task evolution, with stronger validation of when new data should augment or replace prior data.