MolmoB0T: Large-Scale Simulation Enables Zero-Shot Manipulation

MolmoBot Team[1]

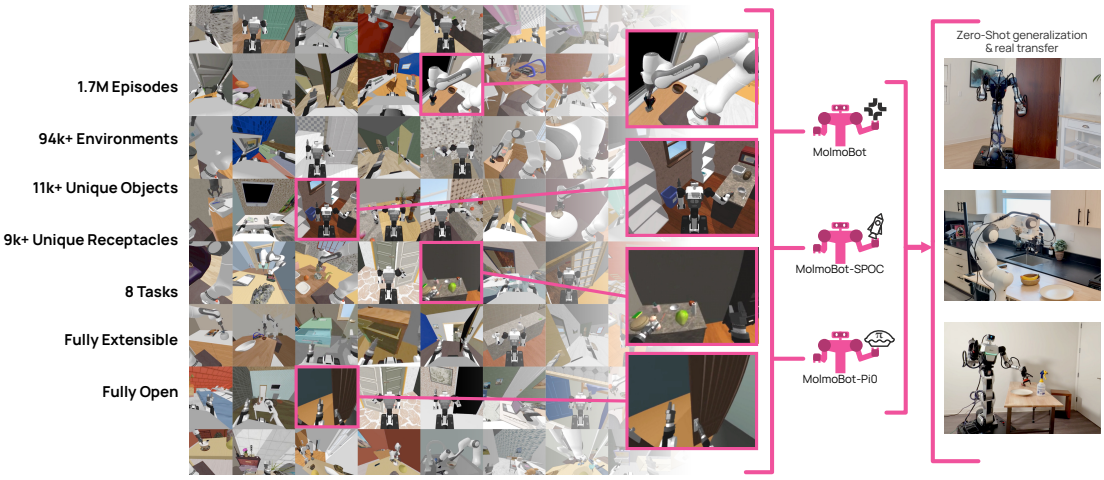

Fig. 1: **MolmoBot** leverages diverse simulation data to achieve zero-shot sim-to-real transfer on multiple robotic tasks and platforms. This unlocks the ability to dramatically scale up the training data for generalist robotic foundation models.

***Abstract*—A prevailing view in robot learning is that simulation alone is not enough; effective sim-to-real transfer is widely believed to require at least some real-world data collection or task-specific fine-tuning to bridge the gap between simulated and physical environments. We challenge that assumption. With sufficiently large-scale and diverse simulated synthetic training data, we show that zero-shot transfer to the real world is not only possible, but effective for both static and mobile manipulation. We introduce *MolmoBot-Engine*, a fully open-source pipeline for procedural data generation across robots, tasks, and diverse simulated environments in MolmoSpaces. With it, we release *MolmoBot-Data*, a dataset of 1.7 million expert trajectories for articulated object manipulation and pick-and-place tasks. We train three policy classes: *MolmoBot*, a Molmo2-based multi-frame vision-language model with a flow-matching action head; *MolmoBot-Pi0*, which replicates the π_0 architecture to enable direct comparison; and *MolmoBot-SPOC*, a lightweight policy suitable for edge deployment and amenable to RL fine-tuning. We evaluate on two robotic platforms: the Franka FR3 for tabletop manipulation tasks and the Rainbow Robotics RB-Y1 mobile manipulator for door opening, drawer manipulation, cabinet interaction, and mobile pick-and-place. Without any real-world fine-tuning, our policies achieve zero-shot transfer to unseen objects and environments. On tabletop pick-and-place, MolmoBot achieves a success rate of 79.2% in real world evaluations across 4 settings, outperforming $\pi_{0.5}$ at 39.2%. Our results demonstrate that procedural environment generation combined with diverse articulated assets can produce robust manipulation policies that generalize broadly to the real world. Videos are available here.**

I. Introduction

Robotics foundation models are increasingly being built by a small number of well-resourced industrial labs — NVIDIA's GR00T [28], Physical Intelligence's π-family [5, 4], and Google DeepMind's Gemini Robotics [35] — yet the data mixtures, training recipes, and scaling regimes behind these systems remain largely undisclosed, concentrating the knowledge required to build such models within a few institutional actors. In the absence of open recipes, much of the community has gravitated toward fine-tuning existing systems, reinforced by a widely held assumption that simulation alone cannot produce robust real-world manipulation policies and that bridging the sim-to-real gap requires real-world data. We challenge that assumption. We introduce MolmoBot-Engine, a fully open-source pipeline for procedural data generation across robots, tasks, and diverse simulated environments, and MolmoBot-Data, a dataset of 1.7M expert trajectories spanning articulated object manipulation and pick-and-place. We train three policy classes: MolmoBot, built on Molmo2 [9] with a layerwise-coupled DiT-based flow-matching action head; MolmoBot-Pi0, which replicates the π_0 architecture for controlled comparison; and MolmoBot-SPOC, a lightweight policy for edge deployment. Evaluated on the Franka FR3 and Rainbow Robotics RB-Y1 without any real-world fine-tuning, our policies transfer zero-shot to unseen objects and environments — MolmoBot achieves 79.2% success on tabletop pick-and-place versus 39.2% for $\pi_{0.5}$, and MolmoBot-Pi0 reaches 46.5% using the same architecture, demonstrating that the performance gap is driven by data. Our results suggest the barrier to general-purpose manipulation is less about an irreducible sim-to-real gap and more about access to sufficiently large, diverse, and open simulation pipelines; we provide that access by open-sourcing all components.

II. MolmoBot-Engine: A Scalable Data Engine

We introduce **MolmoBot-Engine**, a procedural data generation pipeline for scalable robotic manipulation training, illustrated in Fig. 2. Leveraging large-scale procedural generation, massive asset libraries, and aggressive randomization, we

[1]Full author list deferred to Appendix A.

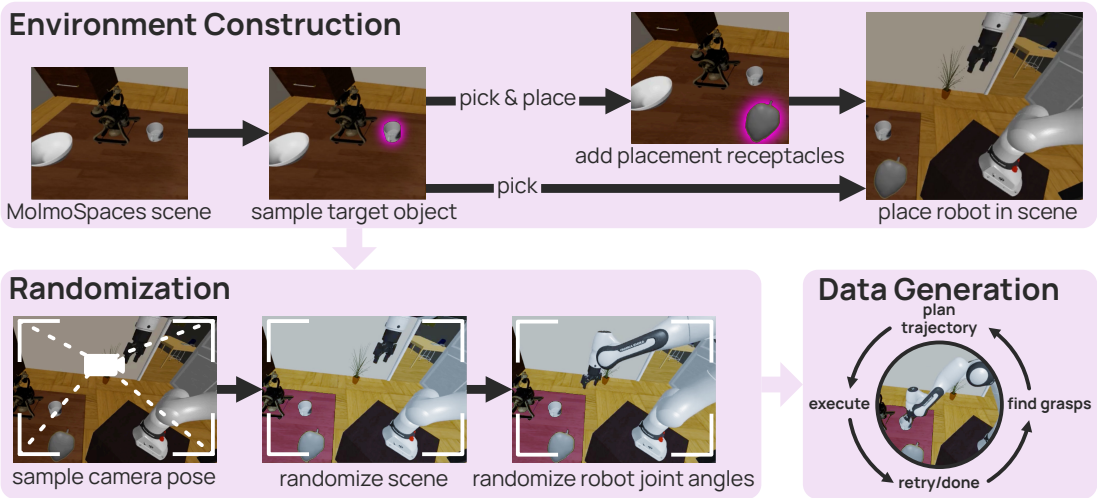

Fig. 2: **MolmoBot-Engine**. Starting from a pre-built MolmoSpaces [19] house, we sample task-relevant objects, randomize visual and physical parameters, and iteratively replan as necessary until a successful trajectory is found.

generate demonstration data at scale, at a fraction of the cost of real-world collection.

A. MolmoSpaces environments and assets

We leverage the objects and scenes in MolmoSpaces [19], a large collection of procedurally generated indoor environments with realistic and diverse room layouts, object placements, and rigid objects that can be programmatically added to any scene.

Environment setup. Each episode takes place in one of the more than 200k available pre-built MolmoSpaces scenes. The layout, furniture, and static objects remain fixed, and additional manipulable objects are inserted as required for task sampling.

Asset sourcing. Rigid objects for pick-and-place tasks are sourced from iTHOR [20] and Objaverse [10], and filtered for manipulability, which includes size and watertight meshes. Objects can be looked up via semantic relevance using the object metadata provided by MolmoSpaces.

Domain randomization. We extensively perform domain randomization across three axes: environment randomization, action randomization, and camera perturbation. Furthermore, during model training we also perform image augmentation.

For environment randomization, we randomize lighting (number of lights, positions, type, intensities, colors, and shadow parameters), textures (surface materials are randomized where supported with procedural textures and those sourced from AI2THOR assets [21]), and dynamics (frictional and inertial properties, joint damping). We also randomize object configurations by sampling initial poses subject to collision and robot reachability constraints.

B. Robot configuration

We generate data for two robot platforms to enable both mobile manipulation and tabletop evaluation. Additional robot platforms can be easily added by future work.

Franka FR3. A 7-DoF Franka FR3 arm in the DROID [18] configuration, mounted on a fixed base. Following DROID, data generation and evaluation are run at 15 Hz.

Rainbow RB-Y1. A mobile bimanual manipulator with a holonomic base and actuated 6-DoF torso and 2-DoF head. Each arm is 7-DoF and equipped with a parallel-jaw gripper.

Robot-related randomization. At episode initialization, we sample initial robot joint positions around the robot-specific nominal home configuration. Additionally, during the episode, the demonstrator's outputted actions are perturbed via proportional action noise injection to ensure state coverage. Further details on joint randomization and action noise injection are provided in App. C. Robot cameras are also randomized in both intrinsics and extrinsics, detailed in App. D.

C. Task definitions

Rigid object manipulation. We define four rigid-body manipulation tasks, each evaluated with both the stationary Franka FR3 and mobile RB-Y1 manipulators: *Pick*, *Pick-and-place*, *Pick-and-place-next-to*, and *Pick-and-place-color*. We defer specific success criteria to App. E1.

Articulated object manipulation. We define two articulated-object tasks, evaluated with the mobile RB-Y1: *Open* and *Open-door*, with specifics detailed in App. E2.

D. Expert planners

For each task, we generate expert demonstrations at scale via scripted demonstrators that iteratively sample grasps, verify feasibility, and execute motion for each task phase. Expert demonstrators for the Franka FR3 use IK-based interpolation, and for RB-Y1 use the CuRobo [34] motion generator to coordinate the many degrees of freedom with collision-aware motion planning. Pregenerated grasps from MolmoSpaces-Grasp [19] are used to cheaply sample stable grasps during data generation. Finally, programmed retry behavior provides recovery data, serving to robustify learned policies. Complete details are provided in App. F.

III. Models and Training

We train three policy classes on MolmoBot-Data, enabling comparison across architectures and against external baselines:

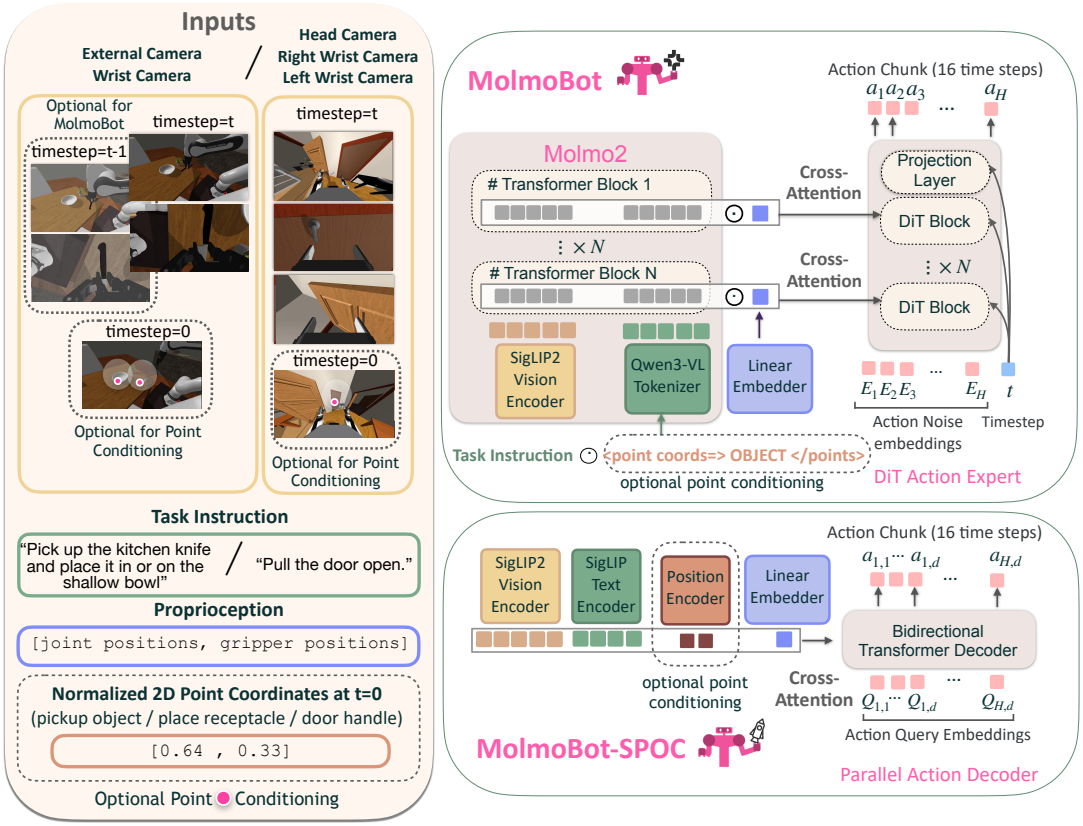

Fig. 3: **Left:** MolmoBot policies ingest multiple RGB camera views (from possibly multiple timesteps), proprioceptive state, and a language task instruction. Optionally, policies can be additionally conditioned on 2D points, represented as a grounding image and a point coordinate in that image. **Top right:** MolmoBot uses a Molmo2-4B vision-language backbone and a DiT-based flow matching action head that cross-attends to state features to denoise action chunks. **Bottom right:** MolmoBot-SPOC is a lightweight transformer-based policy, using a transformer decoder to predict an action chunk in parallel, conditioned on frozen SigLIP2 vision and text embeddings. MolmoBot-Pi0 (not shown) exactly follows the π_0 [3] architecture, initialized from PaliGemma [2] weights and a randomly initialized action expert.

MolmoBot, **MolmoBot-Pi0**, and **MolmoBot-SPOC**. We illustrate MolmoBot and MolmoBot-SPOC in Fig. 3, and defer complete descriptions and details to App. J.

IV. Experiments

To illustrate the performance and generality of MolmoBot policies and the MolmoBot-Engine, we train and evaluate on multiple tasks in various real-world and simulated settings. Crucially, all models are only trained on sim, with zero task-specific or real-world post-training or finetuning. All MolmoBot policies have never seen any real-robot data.

A. Zero-Shot Real-World Transfer

1) Baselines: For our DROID evaluations, we compare against π_0 [3] and $\pi_{0.5}$-DROID [4], which are SOTA open-weights manipulation policies for the DROID platform. Both are trained with >10k hours of real-world data and fine-tuned on the DROID dataset, and $\pi_{0.5}$-DROID further adds innovations to greatly boost generality. The RB-Y1 enjoys less community adoption than DROID, and to our knowledge there are no available generalist policies to compare against. We therefore reserve quantitative comparisons for our DROID evaluations.

2) Static Manipulation Evaluation: **Setup.** We evaluate all Franka policies in four real-world environments across two institutions, totaling 120 trials per policy. Evaluation environments are pictured in Fig. 5, and further details are provided in App. N.

Results In our real-world static manipulation evaluations, MolmoBot policies exhibit strong zero-shot sim2real transfer, as illustrated in Fig. 4. MolmoBot and MolmoBot-Img significantly outperform $\pi_{0.5}$-DROID while MolmoBot-Pi0 is competitive, all without the benefit of $\pi_{0.5}$-DROID's architectural improvements. Additionally, all MolmoBot policies perform much better than π_0. Despite having identical architectures, MolmoBot-Pi0 significantly outperforms π_0 in our evaluations, highlighting the value of MolmoBot-Data. We further study the dynamics of how this data affects policy behavior in App. Q.

3) Mobile Manipulation Evaluation: **Setup.** We evaluate the MolmoBot Door Specialist on a door opening task in three real-world environments, each featuring a distinct pull door with varying textures and handle configurations. As opposed to push doors which can be opened by naively driving into the door, pull doors require precise grasping behavior and

TABLE I: Evaluation on held-out simulation environments, averaged over 1000 episodes per task, with real-world results from Fig. 4. For pick-and-place tasks, we report both oracle success (success condition fulfilled at any timestep) and success at end (success condition fulfilled at the final timestep). We additionally report the half-width of the 95% confidence interval bounds.

Model	Pick MSProc	Pick Classic	Pick	Pick Rand.-Cam.	Pick&Place	PnP Next-To	PnP Color	Avg.	Real
$\pi_{0.5}$ [4]	18.1 ±2.4	6.4 ±1.5	7.0 ±1.6	8.0 ±1.9	11.7 ±2.1/7.6 ±1.7	8.2 ±2.2/6.2 ±1.9	10.4 ±1.9/6.7 ±1.6	10.0	31.3
$\pi_{0.5}$-Finetune	48.0 ±3.1	28.3 ±2.8	25.8 ±2.7	29.7 ±2.9	43.5 ±6.0/37.4 ±5.8	**28.4** ±3.2/14.7 ±2.5	48.3 ±3.1/38.9 ±3.0	36.0	–
StereoVLA [12]	6.6 ±2.6	4.3 ±1.5	1.1 ±1.0	N/A	0	N/A	0	-	–
LAP-VLA [45]	19.4 ±2.4	2.4 ±1.0	3.1 ±1.1	2.7 ±1.0	3.81 ±1.5/1.59 ±1.0	6.48 ±2.8/3.41 ±2.1	3.1 ±1.1/1.5 ±0.8	4.8	–
X-VLA [50]	3.3 ±1.0	0.5 ±0.5	0.7 ±0.5	0.8 ±0.5	0.1 ±0.2/0.1 ±0.2	1.9 ±0.9/1.0 ±0.7	0.9 ±0.6/0.5 ±0.5	1.2	–
MB-Pi0	66.2 ±2.9	35.7 ±3.0	33.3 ±2.9	39.8 ±3.1	44.7 ±3.1/38.2 ±3.1	24.7 ±3.2/13.3 ±2.5	46.2 ±3.1/40.0 ±3.1	41.5	46.7
MolmoBot-Img	92.2 ±1.7	63.5 ±3.0	61.4 ±3.0	62.1 ±3.0	63.0 ±3.0/55.0 ±3.1	21.0 ±2.5/16.4 ±2.3	**67.8** ±2.9/**60.3** ±3.0	61.6	72.5
MolmoBot (F=2)	**93.5** ±1.5	**66.8** ±2.9	**64.0** ±3.0	**63.7** ±3.0	**66.4** ±2.9/**57.7** ±3.0	26.4 ±2.7/20.2 ±2.5	**67.8** ±2.9/60.0 ±3.0	**64.1**	**79.2**
MolmoBot (F=3)	91.3 ±1.8	63.8 ±3.0	59.0 ±3.0	62.7 ±3.0	65.4 ±2.9/55.6 ±3.1	28.3 ±2.8/**22.6** ±2.6	66.1 ±2.9/57.3 ±3.1	62.4	75.0

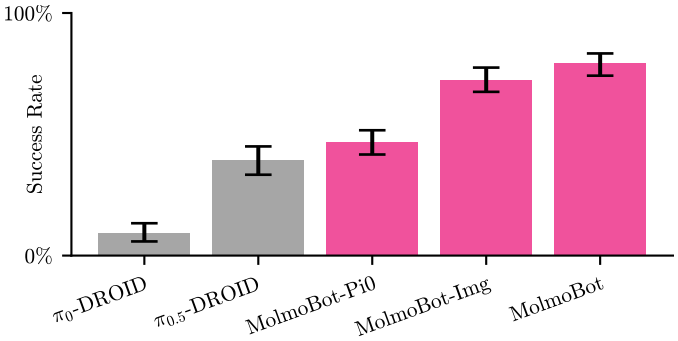

Fig. 4: MolmoBot policies exhibit strong zero-shot sim2real performance across our 120 real-world DROID evaluations, outperforming SOTA policies trained on large-scale real-world demonstrations. Error bars represent 95% confidence intervals, estimated via stratified bootstrapping. MolmoBot denotes the MolmoBot (F=2) variant.

coordinated motion to avoid collision.

Results. Full evaluation results are presented in Tab. VIII. The robot successfully grasped the door handle in 4 of 9 trials, and in 2 of which successfully opened the door. As described in the table, some trials were mired by faults triggered by extant hardware issues. In these cases, the e-stop is triggered, preventing policy rollout. We note that upon successfully grasping the handle without triggering a fault, the robot always succeeded in opening the door, which indicates that HW faults are hampering what otherwise would have been successful rollouts. Further failure mode analysis is detailed in App. O.

B. Simulation Evaluation

For further reduced evaluation uncertainty, we turn to systematic simulation evaluation.

Setup. We evaluate on held-out procedural houses with held-out asset instances, following MolmoSpaces [19]. We evaluate each task on these houses and report oracle success rate (task completion at any timestep), and success-at-end for pick-and-place tasks. Full sim evaluation details are in App. P.

Tasks. We evaluate increasingly difficult tasks on 1000 episode benchmarks, detailed in Tab. I. We evaluate picking tasks with a time budget of 20s under various levels of complexity, which push the evaluations further OOD. We also evaluate on pick-and-place tasks with a time budget of 40s, including placing an object into a receptacle, next to a target, or into a receptacle of specified color. Further details on task details and success criteria are described in App. P.

Baselines. We compare against existing VLAs: $\pi_{0.5}$ [4] is evaluated both zero-shot and after fine-tuning on MolmoBot-Data for 15K steps to adapt it to simulation. We also evaluate StereoVLA [12], LAP-VLA [45], and X-VLA [49] zero-shot.

Results. MolmoBot policies outperform all baselines across all tasks. $\pi_{0.5}$-DROID and $\pi_{0.5}$-Finetune are the strongest baselines, averaging 10% and 36% success respectively, with the latter ameliorated by finetuning on MolmoBot-Data. Other VLA baselines (StereoVLA, LAP-VLA, and X-VLA) generally perform poorly in our evaluations, with average success rates of less than 5%. MolmoBot policies, on the other hand, do far better: MolmoBot-Pi0 averages 41.5% success while Molmo2-based policies achieve 61%–64%. MolmoBot (F=2) performs the best, achieving an average success rate of 64.1%. Further discussion and additional results are presented in App. P.

V. Conclusion

In this work, we demonstrate that zero-shot transfer to the real world is not only possible, but effective for both static and mobile manipulation. We introduce MolmoBot-Engine and use it to generate MolmoBot-Data, with which we train multiple policy classes. Without **any** real-world fine-tuning, our policies demonstrate zero-shot transfer to novel environments and objects: MolmoBot achieves a success rate of 79.2% in our real-world static manipulation evaluations, more than double that of the strongest baseline, $\pi_{0.5}$-DROID, demonstrating broad generalization capability. We release all models, data, and code to enable the community to extend this approach.

MolmoBot represents a promising step for scaling up simulation data, but there remain several open challenges. MolmoBot-Engine is fundamentally constrained to assets and tasks that can be accurately simulated, limiting our focus to rigid body and articulated object manipulation. Extending to contact-rich manipulation, deformable objects, or tasks requiring accurate fluid or granular dynamics poses new difficulties. We believe that coupled with advances in physics-based or generative simulators, our recipe of massive-scale procedural generation may extend to these more challenging tasks. We are excited by the potential frontiers of open research this work enables.

References

[1] AgiBot-World-Contributors. "AgiBot World Colosseo: A Large-scale Manipulation Platform for Scalable and Intelligent Embodied Systems". In: *arXiv preprint arXiv:2503.06669* (2025).

[2] Lucas Beyer et al. "PaliGemma: A versatile 3B VLM for transfer". In: *arXiv preprint arXiv:2407.07726* (2024).

[3] Kevin Black et al. "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164* (2024).

[4] Kevin Black et al. "$\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization". In: *9th Annual Conference on Robot Learning*. 2025.

[5] Kevin Black et al. "π0: A vision-language-action flow model for general robot control. CoRR, abs/2410.24164, 2024. doi: 10.48550". In: *arXiv preprint ARXIV.2410.24164* (2024).

[6] Anthony Brohan et al. "RT-1: Robotics Transformer for Real-World Control at Scale". In: *ArXiv* abs/2212.06817 (2022). URL: https://api.semanticscholar.org/CorpusID: 254591260.

[7] Anthony Brohan et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *ArXiv* abs/2307.15818 (2023). URL: https://api.semanticscholar.org/CorpusID:260293142.

[8] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion". In: *The International Journal of Robotics Research* 44 (2023), pp. 1684–1704. URL: https://api.semanticscholar.org/CorpusID:257378658.

[9] Christopher Clark et al. "Molmo2: Open Weights and Data for Vision-Language Models with Video Understanding and Grounding". In: *arXiv preprint arXiv:2601.10611* (2026).

[10] Matt Deitke et al. "Objaverse: A Universe of Annotated 3D Objects". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 13142–13153. URL: https://api.semanticscholar.org/CorpusID:254685588.

[11] Shengliang Deng et al. *GraspVLA: a Grasping Foundation Model Pre-trained on Billion-scale Synthetic Action Data*. 2025. arXiv: 2505.03233 [cs.RO]. URL: https://arxiv.org/abs/2505.03233.

[12] Shengliang Deng et al. *StereoVLA: Enhancing Vision-Language-Action Models with Stereo Vision*. 2025. arXiv: 2512.21970 [cs.RO]. URL: https://arxiv.org/abs/2512.21970.

[13] Kiana Ehsani et al. "Spoc: Imitating shortest paths in simulation enables effective navigation and manipulation in the real world". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 16238–16250.

[14] Ben Eisner, Harry Zhang, and David Held. "FlowBot3D: Learning 3D Articulation Flow to Manipulate Articulated Objects". In: *Proceedings of Robotics: Science and Systems*. New York City, NY, USA, June 2022. DOI: 10.15607/RSS.2022.XVIII.018.

[15] Jiaheng Hu et al. "FLaRe: Achieving Masterful and Adaptive Robot Policies with Large-Scale Reinforcement Learning Fine-Tuning". In: (2024). arXiv: 2409.16578 [cs.RO]. URL: https://arxiv.org/abs/2409.16578.

[16] Arhan Jain et al. *PolaRiS: Scalable Real-to-Sim Evaluations for Generalist Robot Policies*. 2025. arXiv: 2512.16881 [cs.RO]. URL: https://arxiv.org/abs/2512.16881.

[17] Abhishek Joshi et al. *Procedural Generation of Articulated Simulation-Ready Assets*. 2025. arXiv: 2505.10755 [cs.RO]. URL: https://arxiv.org/abs/2505.10755.

[18] Alexander Khazatsky et al. *DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset*. 2024.

[19] Yejin Kim et al. *MolmoSpaces: A Large-Scale Open Ecosystem for Robot Navigation and Manipulation*. 2026.

[20] Eric Kolve et al. "AI2-THOR: An Interactive 3D Environment for Visual AI". In: *ArXiv* abs/1712.05474 (2017). URL: https://api.semanticscholar.org/CorpusID: 28328610.

[21] Eric Kolve et al. "AI2-THOR: An Interactive 3D Environment for Visual AI". In: *CoRR* abs/1712.05474 (2017). arXiv: 1712.05474. URL: http://arxiv.org/abs/1712.05474.

[22] Xuanlin Li et al. "Evaluating real-world robot manipulation policies in simulation". In: *arXiv preprint arXiv:2405.05941* (2024).

[23] Bo Liu et al. "Libero: Benchmarking knowledge transfer for lifelong robot learning". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 44776–44791.

[24] Corey Lynch et al. "Learning Latent Plans from Play". In: *Conference on Robot Learning*. 2019. URL: https://api.semanticscholar.org/CorpusID:67877011.

[25] Ajay Mandlekar et al. "MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations". In: *7th Annual Conference on Robot Learning*. 2023.

[26] Ajay Mandlekar et al. "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation". In: *Conference on Robot Learning (CoRL)*. 2021.

[27] Soroush Nasiriany et al. "RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots". In: *International Conference on Learning Representations (ICLR)*. 2026.

[28] NVIDIA et al. *GR00T N1: An Open Foundation Model for Generalist Humanoid Robots*. 2025. arXiv: 2503.14734 [cs.RO].

[29] Abby O'Neill et al. "Open X-Embodiment: Robotic Learning Datasets and RT-X Models : Open X-Embodiment Collaboration0". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 6892–6903. DOI: 10.1109/ICRA57147.2024.10611477.

[30] Abby O'Neill et al. "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6892–6903.

[31] William Peebles and Saining Xie. "Scalable diffusion models with transformers". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2023, pp. 4195–4205.

[32] Physical Intelligence. *openpi*. URL: https://github.com/Physical-Intelligence/openpi.

[33] Dean A. Pomerleau. *ALVINN, an autonomous land vehicle in a neural network*. 2015. URL: https://api.semanticscholar.org/CorpusID:18420840.

[34] Balakumar Sundaralingam et al. *cuRobo: Parallelized Collision-Free Minimum-Jerk Robot Motion Generation*. 2023. arXiv: 2310.17274 [cs.RO]. URL: https://arxiv.org/abs/2310.17274.

[35] Gemini Robotics Team et al. "Gemini robotics: Bringing ai into the physical world". In: *arXiv preprint arXiv:2503.20020* (2025).

[36] Yang Tian et al. "InternData-A1: Pioneering High-Fidelity Synthetic Data for Pre-training Generalist Policy". In: *arXiv preprint arXiv:2511.16651* (2025).

[37] Michael Tschannen et al. "SigLIP 2: Multilingual Vision-Language Encoders with Improved Semantic Understanding, Localization, and Dense Features". In: (2025). arXiv: 2502.14786 [cs.CV]. URL: https://arxiv.org/abs/2502.14786.

[38] Yufei Wang et al. "ArticuBot: Learning Universal Articulated Object Manipulation Policy via Large Scale Simulation". In: *Robotics: Science and Systems (RSS)*. 2025.

[39] Jimmy Wu et al. "TidyBot++: An Open-Source Holonomic Mobile Manipulator for Robot Learning". In: *Conference on Robot Learning*. 2024.

[40] Zhenyu Wu et al. "MoTo: A Zero-shot Plug-in Interaction-aware Navigation for General Mobile Manipulation". In: *ArXiv* abs/2509.01658 (2025). URL: https://api.semanticscholar.org/CorpusID:281079068.

[41] Haoyu Xiong et al. "Adaptive Mobile Manipulation for Articulated Objects In the Open World". In: *ArXiv* abs/2401.14403 (2024). URL: https://api.semanticscholar.org/CorpusID:267212147.

[42] Haoru Xue et al. *Opening the Sim-to-Real Door for Humanoid Pixel-to-Action Policy Transfer*. 2025. arXiv: 2512.01061 [cs.RO]. URL: https://arxiv.org/abs/2512.01061.

[43] Seonghyeon Ye et al. *World Action Models are Zero-shot Policies*. 2026. arXiv: 2602.15922 [cs.RO]. URL: https://arxiv.org/abs/2602.15922.

[44] Yifan Yin et al. "PartInstruct: Part-level Instruction Following for Fine-grained Robot Manipulation". In: *ArXiv* abs/2505.21652 (2025). URL: https://api.semanticscholar.org/CorpusID:278959694.

[45] Lihan Zha et al. *LAP: Language-Action Pre-Training Enables Zero-shot Cross-Embodiment Transfer*. 2026. arXiv: 2602.10556 [cs.RO]. URL: https://arxiv.org/abs/2602.10556.

[46] Xiaohua Zhai et al. "Sigmoid Loss for Language Image Pre-Training". In: (2023). arXiv: 2303.15343 [cs.CV]. URL: https://arxiv.org/abs/2303.15343.

[47] Tianhao Zhang et al. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *IEEE International Conference on Robotics and Automation*. 2017. URL: https://api.semanticscholar.org/CorpusID:3720790.

[48] Tony Z Zhao et al. "Learning fine-grained bimanual manipulation with low-cost hardware". In: *arXiv preprint arXiv:2304.13705* (2023).

[49] Jinliang Zheng et al. "X-VLA: Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model". In: *ArXiv* abs/2510.10274 (2025). URL: https://api.semanticscholar.org/CorpusID:282057092.

[50] Jinliang Zheng et al. "X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model". In: *arXiv preprint arXiv:2510.10274* (2025).

APPENDIX

A. Full Author List

Abhay Deshpande[♥1*], Maya Guru[♥1*], Rose Hendrix[♥1*], Snehal Jauhri[♥1,4*],
Ainaz Eftekhar[♥1,2], Rohun Tripathi[♥1], Max Argus[♥1], Jordi Salvador[♥1],
Haoquan Fang[♥1,2], Matthew Wallingford[♥1], Wilbert Pumacay[♥1], Yejin Kim[♥1],
Quinn Pfeifer[2], Ying-Chun Lee[2], Piper Wolters[1], Omar Rayyan[3], Mingtong Zhang[5], Jiafei Duan[1,2],
Karen Farley[1], Winson Han[1], Eli Vanderbilt[1], Dieter Fox[1,2], Ali Farhadi[1,2], Georgia Chalvatzaki[4],
Dhruv Shah[♥†5], Ranjay Krishna[♥†1,2]

[1]Allen Institute for AI, [2]University of Washington, [3]University of California, Los Angeles,
[4]Technische Universität Darmstadt, [5]Princeton University
*denotes equal contribution in alphabetical order, ♥ marks core contributors, and † indicates equal advising.

B. Related Work

Imitation learning for manipulation. Imitation learning is the leading paradigm for robot manipulation. Initial methods focused on behavior cloning that map observations to actions [33, 47], while later work introduced hierarchical structures and temporal abstractions to address long-horizon tasks more effectively [24]. Recently, generative modeling techniques such as diffusion policies [8] have been introduced, demonstrating strong performance on manipulation benchmarks.

Recent developments have extended imitation learning to vision-language-action (VLA) models that integrate language understanding with perception and control within a unified architecture. Systems such as RT-1 [6] and RT-2 [7] showcase that increasing model capacity and utilizing multi-task robot datasets enable policies to perform hundreds of manipulation tasks conditioned on natural language instructions. More recently, π_0 and its subsequent variants [3, 4] applied a flow-matching action representation that enabled continuous action generation and supports generalist policies capable of cross-embodiment learning. Other recent works that explore cross-embodiment training using heterogeneous real-world robot datasets include X-VLA[49] that conditions a shared policy on embodiment-specific prompt tokens for multi-robot training, and LAP-VLA[45] that aligns robot control with languages by representing actions as language tokens. Although these systems exhibit impressive capabilities, they depend heavily on large-scale real-world robot demonstrations. In contrast, this work investigates training VLA policies exclusively from simulation-generated trajectories while preserving strong real-world performance.

Large-scale datasets and simulation. The advancement of generalist robot policies is closely associated with the availability of large-scale datasets. Several initiatives have gathered extensive real-world demonstrations spanning diverse tasks and embodiments, enabling learning from heterogeneous trajectories [29]. Datasets like DROID [18] offer large collection of manipulation demonstrations for training contemporary VLA models.

Owing to the high cost and logistical challenges of real-world data collection, recent research has increasingly emphasized simulation or synthetic datasets. GraspVLA [11] explores VLA policies trained on simulated grasping demonstrations, while the InternVLA family (InternVLA-M, InternVLA-A, InternVLA-H/N) [36] demonstrates large-scale pretraining for manipulation, action planning, navigation, and humanoid control using synthetic trajectories. Similarly, ArticuBot [38] leverages simulation at scale to generate and train on large demonstration datasets for articulated object manipulation. Additionally, work such as PartInstruct [44] and Infinigen-Articulated [17] illustrates the effectiveness of procedurally generated simulation datasets in supporting robot learning research.

Our work extends this line of research by introducing MolmoBot-Engine, a fully open-source pipeline that enables scalable data generation in simulation across different robots, tasks, and diverse environments, and MolmoBot-Data, a large-scale generated dataset of expert manipulation trajectories. By combining procedural scene generation with diverse rigid and articulated assets, our dataset enables training generalist policies that transfer to real-world deployment without any real-world demonstrations.

Articulated and mobile manipulation. Manipulating articulated objects such as doors, drawers, and cabinets remains challenging due to complex contact dynamics and partially observable object states. Mobile manipulation introduces additional complexity, requiring coordination among navigation, perception, and manipulation. Most large-scale manipulation systems concentrate on fixed-base manipulators operating in tabletop environments, where perception and workspace constraints are less complex [18, 7]. Several recent works that explore mobile manipulation typically address only a subset of the problem. For instance, some approaches focus on navigation relying on fixed-base manipulation skills for overall mobile manipulation tasks [40], or demonstrate only the feasibility of mobile manipulation platforms through real-world teleportation datasets [39] or real-world online adaptation strategies [41]. Other prior work has explored articulation-aware policies that incorporate object geometry and motion constraints. For example, FlowBot3D [14] learns manipulation flows to guide robot interaction with articulated objects.

Despite these advances, mobile manipulation remains underexplored within large-scale imitation learning frameworks. A recent work used simulations to collect a scalable dataset and demonstrated that sim-to-real transfer outperformed human teleoperators [42]. However, for particular articulated categories, such as door opening, solutions remain task-specific. This

study evaluates policies on both a tabletop manipulator and a mobile manipulator that performs multiple tasks such as mobile pick-and-place and door opening. The results demonstrate that large-scale simulation-generated data can produce policies that generalize to both articulated and mobile manipulation scenarios without real-world demonstrations.

C. Datagen — Robot Randomization

Initial joint-configuration randomization. At episode initialization, each move group's joint positions are sampled as $q_0 + \delta$, where q_0 is a nominal home configuration and $\delta_i \sim \mathcal{U}(-r_i, r_i)$ with per-joint noise magnitudes r_i. For both robots, the arm noise magnitudes are *graduated*: proximal joints receive smaller perturbations and distal joints larger ones. Concretely, the Franka arm uses $\mathbf{r}_{\text{arm}} = [0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175]$ rad (chosen via a Jacobian-weighted heuristic to bound TCP displacement to ≤ 10 cm), and each RB-Y1 arm uses $\mathbf{r}_{\text{arm}} = [0.05, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175]$ rad. The RB-Y1 additionally randomizes head pan and tilt (± 0.2 rad $\approx \pm 11.4^\circ$ each) and gripper aperture (± 0.01 rad). Torso and base initial joint positions are not perturbed.

Action noise injection. During data collection, noise is injected into expert actions to prevent policies from overfitting to exact action replay. The noise is *action-proportional*: its standard deviation scales with the magnitude of the commanded displacement, so stationary commands receive no noise and large motions receive proportionally more.

For arm move groups, noise is applied in TCP space and then mapped back to joint space via the Jacobian pseudo-inverse. Specifically, we compute the commanded TCP displacement $\Delta\mathbf{x} = J\Delta\mathbf{q}$ from the Jacobian J and the joint-space command $\Delta\mathbf{q}$. Position noise is sampled from a truncated Gaussian with $\sigma_{\text{pos}} = \alpha\|\Delta\mathbf{x}_{\text{pos}}\|$ and clipped to ± 2 cm, where $\alpha = 0.1$ is a scale factor. Rotation noise uses $\sigma_{\text{rot}} = 0.1 \cdot \sigma_{\text{pos}}$, clipped to ± 0.1 rad ($\approx 5.7^\circ$). The resulting 6-DoF TCP noise vector $\boldsymbol{\epsilon}_{\text{tcp}}$ is projected to joint space by solving $J\boldsymbol{\epsilon}_q = \boldsymbol{\epsilon}_{\text{tcp}}$ in the least-squares sense, and the noisy command is clipped to joint limits.

For the RB-Y1 base, planar noise is applied directly to (x, y, θ) commands using clipped Gaussians with $\sigma = 0.1 \cdot \|\Delta\mathbf{p}\|$, bounded to ± 2 cm in position and ± 0.05 rad ($\approx 2.8^\circ$) in heading. Action noise is disabled during simulated evaluation.

Gripper handling. Gripper close and open commands execute over fixed durations of 0.5s and 0.25s, respectively, followed by a settle period (`move_settle_time` = 0.1 s for the Franka; up to `max_grasping_timesteps` = 5 control steps for the RB-Y1) during which the arm is held stationary. This simulates real-world grasp settling time and ensures the object is stably grasped before subsequent arm motion resumes.

D. Datagen — Sensor Configuration

After placing objects and applying domain randomization, we configure the robot's sensors for the episode. We describe the camera systems for each platform, followed by additional sensor modalities.

1) FR3 camera system: We generate data for five camera viewpoints for the Franka FR3, to provide diverse viewpoints for tabletop manipulation. In this work, we only train with the wrist camera and one randomized exocentric camera; the rest are present in MolmoBot-Data to provide for future work.

Wrist camera. A gripper-mounted camera analogous to a ZED Mini, with 52° vertical FOV (±4° noise). Position is perturbed by ±1.5cm lateral, ±0.5cm vertical, and ±2cm in depth; orientation by ±8° in roll and ±4° in pitch and yaw.

Fixed shoulder camera. A robot-mounted exocentric camera positioned at a fixed offset from the robot base, with 71° FOV and light randomization (±5cm position, ±8° orientation). Placement is constrained to maintain visibility of task objects.

Randomized exocentric cameras. Three freely-placed cameras sample positions around the workspace center: two ZED2 analogues (64–72° FOV) and one GoPro analogue (137–140° FOV). For each camera, we sample distance (0.2–0.8m for ZED2, 0.2–0.5m for GoPro), height (0.05–0.6m above workspace), and azimuth (full 360°). Lookat target is the workspace center with ±10cm noise. Placement is rejected and resampled (up to 20 attempts) if task objects and gripper are not visible.

All FR3 cameras render at 624×352, chosen to be close to the real-world resolution of 640×360 while keeping both dimensions a multiple of 16 for video encoding.

2) RB-Y1 camera system: The RB-Y1 uses three cameras matching the real robot's sensor configuration, all of which are used for training in this work:

Head camera. A head-mounted camera analogous to a GoPro in wide mode, rendered at 1024×576 and cropped to 768×576 (4:3 aspect ratio) in post-processing. We use a vertical FOV of 139° with ±3° noise. Position is perturbed by ±1cm in each axis, orientation by ±4° around each axis, and randomized fisheye warping is applied per-frame during training.

Wrist cameras. Left and right wrist-mounted cameras analogous to Intel RealSense D405 sensors. These render at 1024×576 (16:9 aspect ratio) with 58° vertical FOV and ±4° FOV noise. Position noise is ±1.5cm lateral, ±0.5cm vertical, and ±1cm in depth; orientation noise is ±8° in roll and ±4° in pitch and yaw. Depth is recorded for the benefit of future dataset utility but unused during training.

3) Proprioception and additional sensors: Beyond visual observations, we record proprioceptive state and auxiliary information for analysis and potential future use:

Robot state. We record the joint positions and velocities, TCP poses for each gripper, and the robot base pose.

Action labels. We record actions in multiple representations: commanded joint positions (absolute and relative to current joint positions), end-effector twist relative to current pose, and absolute end-effector pose. This enables training with different action parameterizations from the same trajectories.

Task state. We record the object start and goal poses, grasp state indicators, policy phase, and retry counts from the expert policy.

Camera parameters. We record the intrinsic and extrinsic parameters for each camera, enabling projection between 2D and 3D coordinates and potential depth-based augmentation. We also record points in the image frame on objects of interest in all cameras.

Depth images are recorded for RGB-D camera analogues but were not used in any of the training runs reported in this work.

E. Datagen — Tasks

Below we detail specific task descriptions and success criteria for each task we generate data for and evaluate on.

1) Rigid object manipulation:

- *Pick:* Grasp a target object and lift it above its starting height. Success requires that the object is no longer supported by any non-robot surface and has been raised by at least 1 cm.
- *Pick-and-place:* Transport a target object to a specified receptacle. The task succeeds when at least 50% of the object's weight is supported by the receptacle, and the receptacle has not been displaced by more than 10 cm or rotated by more than 45°.
- *Pick-and-place-next-to:* Place a target object adjacent to a reference object on the same surface. Success requires the surface-to-surface distance in the XY plane to lie within $[0, 5]$ cm and the reference object to remain within 15 cm of its initial position.
- *Pick-and-place-color:* Place an object on a receptacle identified by color (e.g., "place on the red plate"). Two receptacles identical (except for color) are placed in the scene; success criteria match *pick-and-place*.

2) Articulated object manipulation:

- *Open:* Open a nearby articulated object (e.g., cabinet, drawer, oven, dishwasher) to at least 67% of its joint range.
- *Open-door:* Open a nearby hinged door to at least 67% of its hinge joint range. The instruction is conditioned on the robot's starting pose relative to the door, yielding either "push the door open" or "pull the door open."

F. Datagen — Expert Demonstrations

Grasp sampling and filtering. Rather than assuming a fixed grasp pose, we load a large set of pre-computed grasp candidates per object from MolmoSpaces' grasp dataset and progressively filter them:

1) **Candidate loading and ranking:** We load pre-computed 6-DoF grasps for each object, transform them into the world frame (including flipped variants), and rank them by a weighted cost that combines TCP proximity, rotation similarity, vertical alignment, and distance to the object center of mass.
2) **Collision filtering:** The top-ranked candidates are tested for gripper–scene collision by placing phantom collision bodies at each candidate pose in MuJoCo and running broadphase collision detection in batches of up to 128.
3) **IK feasibility:** Non-colliding candidates are checked for kinematic reachability via batch inverse kinematics (batches of up to 256). The highest-ranked feasible grasp is selected.

Phase-based trajectory generation. Each task is decomposed into a fixed sequence of phases, with motion planned independently per phase.

For *pick-and-place* tasks, the phases are: PREGRASP → GRASP → LIFT → PREPLACE→ PLACE → POSTPLACE → STOW. For the RB-Y1 demonstrator, the PREPLACE and PLACE phases are combined, and we omit the additional STOW phase. In other words, the policy will first move to a pregrasp pose offset along the grasp approach axis, move to the grasp pose and close the gripper, lift the object, move to a pose above the receptacle before lowering to the placement pose and opening the gripper, and finally moving back to the home position.

Pick-and-place-next-to and *pick-and-place-color* are the same as *pick-and-place*, differing only in placement pose.

Pick tasks proceed similarly to pick-and-place, but terminate after the LIFT phase.

For the *open* and *open-door* tasks, the phases are: PREGRASP → GRASP → ARTICULATE → POSTARTICULATE. After grasping the handle, articulation-specific end-effector waypoints are computed: a circular arc about the hinge axis for revolute joints (doors), or a linear path along the slide axis for prismatic joints (drawers). The planner solves for each waypoint sequentially using IK or trajectory optimization.

Retry behavior Each demonstrator is equipped with retry behavior. While executing a task, if the demonstrator detects a mistake (the object fell out of the grasp, the robot failed to acquire the grasp, etc.) it will reset to the first phase of the trajectory and try again. If more than 3 retries are triggered in an episode, it is terminated and discarded. This explicit retry behavior imbues policies with the ability to handle and recover from mistakes or disturbances.

TABLE II: MolmoBot-Data statistics by task. All episodes include RGB observations, proprioceptive state, action labels, and privileged information such as object visibility that the use of simulation affords. The number of assets reflects pickup objects and receptacles for pick-and-place and variants.

Task	Robot	Episodes	Frames	Assets (Obj. + Rec.)	Envs.	Avg. Length	Total Length
Door-open	RBY1	79.0k	15.4M	–	15.1k	19.6 s	429 h
Open	RBY1	46.6k	6.9M	217	10.5k	14.8 s	192 h
Pick	RBY1	62.3k	7.4M	4.2k	28.0k	10.7 s	184 h
Pick	Franka	781.8k	56.9M	10.7k	73.2k	4.8 s	1,042 h
Pick-and-Place	RBY1	14.8k	2.4M	2.4k + 183	9.3k	14.0 s	58 h
Pick-and-Place	Franka	554.2k	143.9M	7.0k + 494	61.6k	17.1 s	2,638 h
PnP Next-To	Franka	182.7k	54.7M	931 + 9.0k	44.9k	20.1 s	1,022 h
PnP Color	Franka	28.6k	7.5M	3.1k + 183	5.3k	17.4 s	138 h
Total		**1.7M**	**295.2M**	**11.4k + 9.4k**	**94.2k**	**11.7 s**	**5,704 h**

Motion planning for RB-Y1 with CuRobo. For the RB-Y1, we use CuRobo [34] for GPU-accelerated collision-aware trajectory optimization. We model the whole-body kinematics as a 23-DOF chain: a 3-DOF holonomic base (two virtual prismatic joints for planar translation and one continuous joint for yaw), a 6-DOF torso, and two 7-DOF arms. Given a target end-effector pose, CuRobo first solves inverse kinematics using 64 seeds, then computes a collision-free trajectory using 4 trajectory optimization seeds, and finally smooths and interpolates the result to match the simulation control frequency. The robot geometry is approximated by collision spheres, while scene obstacles use mesh-based collision checking with a 0.2m activation distance and self-collision optimization enabled. For phases requiring collision-free motion (e.g., pre-grasp approach, placement), multiple candidate goal poses are evaluated in parallel (default batch size of 4, up to 4 batches), and the trajectory with the least total joint displacement is selected. When a waypoint cannot be reached within a fixed number of control steps, the planner re-plans from the current configuration, up to a maximum of 5 re-planning attempts per phase.

G. Datagen — Referral Expressions

In order to sample referral expressions, we first consider task contexts. For example, in a task where the robot is in font of a workbench with some objects on top of it, the context should be the set of objects lying on the workbench surface). In that context, we seek to maximize the contrast between (1) the CLIP similarity between the normalized textual embedding of the referral expression and the normalized visual embedding of the target object, and (2) the CLIP similarity between the referral expression and any of the other objects on the workbench surface. In other words, we do not take into account whether the referral expression would be a better fit for any other object in the scene, as it is the robot's task to infer the correct context given the task setup and the instruction.

The set of possible referral expressions is composed of LLM-generated short descriptions (between 1 and 5 words), synset lemmas, and normalized object category names. We filter valid expressions that produce a CLIP-similarity contrast ≥ 0.03 while providing a CLIP similarity ≥ 0.1 and sample via a softmax distribution with temperature $2 \cdot 10^{-2}$ over the CLIP similarity contrasts for each filtered expression.

H. Datagen — Train-Time Task Prompt Randomization

To further boost the diversity of the language instructions, we procedurally randomize the task prompt during training.

Template Randomization There are many ways to give the same instruction to the policy. Therefore, at train-time, when sampling data, we sample one of several task prompt templates with varying wording and phrasing for use. For brevity, we defer the full list of task prompt templates to the released code.

Referral Expression Randomization MolmoBot-Data saves multiple valid referral expressions for each task-relevant object for each episode. During training, we then sample a referral expression for each object, biasing towards shorter expressions, which we then insert into the sampled prompt template.

I. Datagen — Dataset Statistics

Table II summarizes MolmoBot-Data. We generate 1.7M episodes comprising 295M frames across more than 11k unique target object assets, more than 9k receptacle object assets, and more than 94k environments, which are further modified by the procedural addition of task-relevant per-episode objects.

Comparison to prior datasets. Table III compares MolmoBot-Data to prior manipulation datasets.

Generation throughput. A key advantage of simulation-based data generation is scalability. Using 100 NVIDIA A100 80GB GPUs, we generate approximately 660 successful episodes per GPU-hour (with about 50% of the time spent on rollouts and

TABLE III: Comparison to prior manipulation datasets. MolmoBot-Data provides substantially more episodes and environment diversity through procedural generation.

Dataset	Source	Episodes	Hours	Unique Envs.	Tasks	Embod.	Mobile Manip.
DROID [18]	Real	76k	350	564	86	1	✗
Open X-Embodiment [30]	Real	1M+	–	–	527	22	✓
AgiBot-World [1]	Real	1M+	2,976	100+	217	1	✓
RoboMimic [26]	Sim	~1k	–	1	5	1	✗
MimicGen [25]	Sim	50k+	–	–	18	1	✗
InternData-A1 [36]	Sim	630k	7,433	227	70	4	✗
RoboCasa-365 [27]	Sim	500k	2,200	2,500	365	1	✓
MolmoBot-Data (Ours)	Sim	1.7M	5,704	94.2k	8	2	✓

the rest on scene setup and task sampling) or, equivalently, more than 88 hours of robot experience per hour of wall-clock time. The full MolmoBot-Data dataset was generated in approximately 6,500 GPU-hours. This represents a near 2.6× data throughput[1] compared to real-world data collection at equivalent scale with human demonstrators, enabling rapid iteration on data composition and task design as well as rapid adoption of new robotics platforms.

J. Models — MolmoBot: VLM-based manipulation policy

MolmoBot builds on Molmo2-4B [9], a vision-language model pretrained on large-scale image-text data. The architecture consists of three components: (1) a vision encoder that processes RGB observations from input camera views (2) a language model which jointly encodes visual features and task instructions, and (3) a DiT-based flow matching action head that predicts robot actions, as visualized in Fig. 3.

Vision encoder. Visual observations are encoded via SigLIP2 [37] and projected into the language model's embedding space. We freeze the vision encoder and the projector weights during training and train only the action head and the language model. We train MolmoBot to ingest up to $F = 3$ frames per view. We encode each image individually and image tokens for each 2×2 patch windows are pooled into a single vector using a multi-headed attention layer, where the mean of the patches serves as the query. Each image is encoded with 192 tokens. We concatenate image tokens from available camera views (head-mounted, external, and wrist cameras, depending on the platform), interleaved with text tokens encoding the image indices and view indices when appropriate. Optionally, we encode the corresponding initial-timestep images to provide context about the starting scene configuration.

LLM. The LLM takes as input the visual tokens interleaved with image indices jointly with the tokenized language instruction. For tasks requiring spatial grounding, we optionally condition on 2D point coordinates specifying target objects or placement locations; these are injected as special tokens in the instruction stream (e.g., `<point coords=> OBJECT </points>`). We use bi-directional attention for the vision tokens and causal attention for the text tokens during training and inference.

Action head. The action head is a DiT [31] which contains self-attention and cross-attention in each layer, where it attends to features of the Molmo2 backbone via cross-attention. Following recent work on flow matching for action prediction [3], the DiT iteratively denoises action chunks conditioned on a continuous timestep embedding $t \in [0, 1]$. The timestep embedding is used by each DiT block to modulate the embedding via adaptive layer normalization [4].

MolmoBot's action head has the same number of layers as the LLM encoder, and each action layer cross-attends to the hidden states of the input sequence (including the encoding of both vision and language) of the corresponding LLM layer. LLM and DiT have different hidden dimensions, so hidden states from the LLM are also projected to DiT's hidden dimension. We also encode robot states through a single-layer MLP, and concatenate them to the end of the VLM sequence before entering cross-attention at each layer. We train the action head to predict chunks of $H = 16$ actions and execute 8 before re-querying the policy following [48].

Action representation. We parameterize actions in joint space using two representations: absolute joint positions and joint position deltas. Both are continuous values representing the target configuration for each joint. At each timestep, the policy predicts targets for all actuated joints, including the gripper. For the RB-Y1's mobile base, we additionally predict base velocity commands (linear and angular) which are concatenated to the joint action. Joint-space control avoids the computational overhead and potential singularities of inverse kinematics at execution time. We train separate model variants on absolute and delta representations for the Franka FR3 task and compare their performance in Section IV. We only use delta policies for training the mobile manipulation task.

[1]Using ALOHA [48] as reference, where the effective real-time factor of a single human demonstrator is 1/3 for tasks of similar duration to ours, due to episode reset overhead or operator mistakes.

Single-frame training. We train MolmoBot with the behavior cloning objective. We train with a batch size of 1024 and train the model for 200k steps for the static manipulation task and for 100k steps for the mobile manipulation task. We use a learning rate of $1 \cdot 10^{-5}$, using a 2k step warm up for the LLM and a 200 step warm up for the action head. When sampling training examples from an expert roll-out, we up-sample steps with retry grasping behavior by 3×, steps with a successful pick by 2× and task completion behavior by 2×. The motivation is to improve the model's grasping behavior and avoid picking objects after task completion.

Our action head has a significantly lighter compute footprint than the VLM encoder. We leverage this during training by sampling multiple time steps T per example to denoise in parallel. This enables us to train the model at multiple time steps for a given observation and action pair. This in turn improves the convergence and the accuracy of the model. We use $T = 8$ to train all MolmoBots unless otherwise stated and report performance with various settings in section IV. We denote the single frame model MolmoBot-Img.

Multi-frame training. We train two multi-frame MolmoBots denoted as MolmoBot (F=2) and MolmoBot (F=3) with number of frames $F = 2$ and $F = 3$ respectively. For the multi-frame training, we initialize the model with the weights from MolmoBot-Img and train the model for 50k steps while keeping all the other training details the same as MolmoBot-Img. When using multiple frames, the model takes as input the frame from the cameras at the current state and the frames sampled D steps ago. We use $D = 8$ in all our experiments. Practically, the $F = 3$ model takes the current state, the state ~0.5 second before the current state and the state ~1 second before the current state.

K. Models — MolmoBot-Pi0

To isolate the effect of MolmoBot-Data on real-world VLA performance, we present MolmoBot-Pi0, a VLA with identical architecture as π_0 [3], trained entirely on our synthetic data from the initial Paligemma weights. This enables head-to-head comparisons with existing SOTA VLAs, controlling for modeling or architecture changes.

Architecture.

Following [3], MolmoBot-Pi0 uses the Paligemma 3B VLM with a flow-matching action expert. We use the openpi [32] codebase for all MolmoBot-Pi0 modeling code, ensuring equivalence with π_0.

Training protocol.

We train for 200k steps at a batch size of 1024 with a learning rate of $5 \cdot 10^{-5}$, using a 1k step warmup. To prevent overfitting to simulation rendering artifacts, we freeze the entirety of the SigLIP vision encoder. Robot actions are supervised as absolute joint positions, following findings from PolaRiS [16]. All other training parameters (flow matching timestep sampling, other optimizer hyperameters, etc.) are left as the default values.

L. Models — MolmoBot-SPOC: A lightweight transformer policy

Overview. SPOC [13] is a transformer-based architecture that demonstrated that imitation learning from shortest-path experts across hundreds of thousands of procedurally generated houses can produce navigation policies that transfer zero-shot to real-world environments. Inspired by this architecture, MolmoBot-SPOC introduces a lightweight transformer-based policy with several modifications that make it suitable for our static and mobile manipulation tasks. **Visual, Language, and Proprioceptive Encoding** Visual observations from all camera inputs are encoded using a SigLIP2-Base patch 16/256 image encoder [37], retaining the full set of patch tokens. Language goal instructions are encoded separately using the SigLIP text encoder [46]. The resulting token sequences consist of (1) visual patch embeddings, (2) language goal tokens, and (3) the robot's current joint state projected into the model's token dimension via a learned linear projection. These tokens are concatenated along the sequence dimension to form the cross-attention memory of the action decoder (Fig. 3). For tasks that provide spatial goal specifications, MolmoBot-SPOC optionally incorporates point-based goal encodings into the cross-attention memory. Depending on the task, one or two 2D pixel coordinates are provided: a single normalized image coordinate (x, y) for *pick*, *open*, and *door-open* tasks, or two coordinates (x_1, y_1, x_2, y_2) for *pick-and-place* tasks. Each coordinate is first passed through a sinusoidal positional encoder and then projected into the model's token dimension using a linear layer. A learned coordinate position embedding is added to each encoded point, and the resulting point tokens are concatenated with the other inputs in the cross-attention memory. MolmoBot-SPOC does not condition on any trajectory history; only the current timestep's observations and state are used. Optionally, we encode the corresponding initial-timestep images to provide context about the starting scene configuration when using point-based goal specification.

Action representation and quantile binning. MolmoBot-SPOC formulates action prediction as a discrete classification problem. Continuous action values are tokenized using a quantile binning strategy. Prior to binning, actions are normalized using the 1st and 99th percentiles of the training distribution, rescaling and clipping values to the $[-1, 1]$ range based on empirical quantiles. The normalized action space for each dimension is then divided into 256 bins, where bin boundaries correspond to equally spaced quantiles of the data (i.e., the $k/256$ quantile for $k = 1, \ldots, 256$). This produces data-adaptive bins that are approximately uniformly populated, yielding a well-calibrated discrete representation of the continuous action space. The

TABLE IV: Multitask data mixture for all MolmoBot Franka FR3 policies. The data mix is selected to ensure all coverage of each of the individual task's training set.

Task	Sampling Ratio
Pick	20%
Pick-and-place Fixed Height	10%
Pick-and-place Random Height	35%
Pick-and-place-next-to	20%
Pick-and-place-color	15%

decoder predicts a categorical distribution over the 256 bins independently for each action dimension at every timestep in the chunk and is trained using a standard cross-entropy loss.

Parallel action decoding. Following [48], MolmoBot-SPOC replaces the autoregressive decoder used in SPOC with a non-causal parallel decoder (Fig. 3). Instead of predicting actions sequentially, the decoder predicts an entire chunk of $D \times T$ action tokens in a single forward pass, where D is the number of robot action dimensions and $T = 16$ is the fixed chunk length. The decoder is provided with $D \times T$ learnable query embeddings—one for each $(\text{action dimension}, \text{timestep})$ pair in the chunk. Using bidirectional self-attention allows each query token to attend to all others within the chunk. Temporal structure is encoded using sinusoidal positional encodings applied over the flattened sequence of $D \times T$ positions, which are added to the learnable query embeddings before decoding.

TABLE V: Multitask data mixture for all MolmoBot and MolmoBot-SPOC RB-Y1 policies.

Model	Open	Door-open	Pick	Pick-and-place
MolmoBot Multitask	20%	20%	30%	30%
MolmoBot Door Specialist	-	100%	-	-
MolmoBot-SPOC Rigid	-	-	50%	50%
MolmoBot-SPOC Articulated	45%	55%	-	-

M. Models — Implementation details

Data mixing Table IV details the different training sets we use to train all the Franka FR3 policies. Pick-and-place Random Height comprises of pick and place tasks with the robot position initialized at random heights, while Pick-and-place Fixed Height initializes the model at the default droid position [18]. Training with randomized heights makes the model more robust to inference time variations. Pick-and-place-color trains the model to attend to the color attribute in input task and Pick-and-place-next-to training helps improve the models spatial understanding. Table V details the various data mixtures used to train RB-Y1 policies. Door-open is a specialized door opening task, while the Open task includes training samples to open cabinets and drawers.

Data augmentation We use image augmentation to improve our models' sim to real transfer. Specifically, we use ColorJitter, GaussianBlur, RandomPosterize, RandomSharpness and RandomGrayscale with different probabilities. Furthermore, we add prompt randomization during training to make the robot robust to inference time variation in language instruction. Further details on prompt randomization are detailed in Sec. H.

DROID Input/Output For all MolmoBot models trained for the DROID platform, we train on 2 of the cameras detailed in Sec. D3: the wrist ZED Mini camera, and one randomized exocentric ZED 2 camera. In addition to images, the current joint angles are also given to the model. All MolmoBot DROID models output actions as absolute joint position commands.

RB-Y1 Input/Output For all MolmoBot models trained for the RB-Y1 platform, we train on all 3 generated camera views: the head and both wrist cameras. MolmoBot-SPOC Articulated uses the repeated first frame of the trajectory and a normalized image point to ground the task spatially. The point is sampled from ten candidates that are on the desired task object to be manipulated. MolmoBot uses similar point conditioning for the door and articulated tasks. We also provide current joint angles. All RB-Y1 models output torso and arm actions as delta joint position commands, and mobile base commands as linear and angular offsets from the current pose.

N. Experiments — DROID Evaluation Environments

Below we describe each real-world environment used for DROID evaluations in the real world, pictured in Fig. 5. Specific task prompts for each task are detailed in Tab. VI. We also present the full results of every real-world evaluation trial for each policy on each task in each environment in Tab. VII.

Fig. 5: Real-world environments for our DROID evaluations. From left to right: kitchen, workroom, bedroom, office. Additional details in the Appendix.

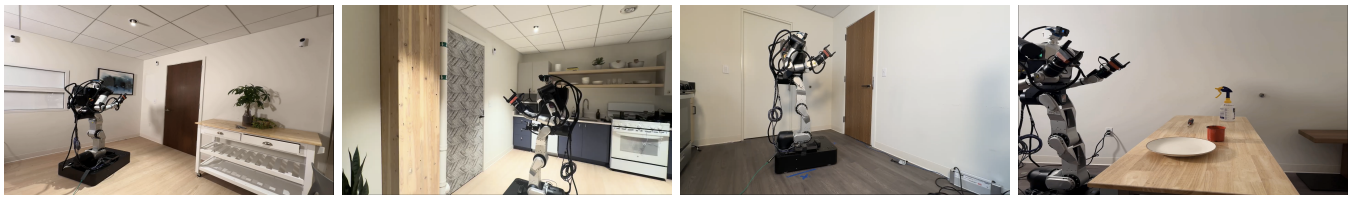

Fig. 6: Real-world environments for our RBY1 articulated and rigid object mobile manipulation evaluations.

For pick-and-place tasks, given a task prompt (e.g. "put the banana in the black bowl"), the policy must move the specified object to be stably in or on the given receptacle. If the policy accomplishes this for a reasonable amount of time within the episode horizon (900 steps), the trial is counted as a success. Failure to do so within the episode horizon, or exhibiting unsafe behavior (high-speed collisions, etc.) before success counts as a task failure.

Kitchen The kitchen environment consists of 4 objects (mug, computer mouse, apple, banana) and 2 receptacles (brown bowl, black bowl). The receptacles are placed to the left and right of the robot, while the objects are either placed in between the bowls ("easy" placement) or further from the workspace center, closer to the wrong bowl ("hard" placement). For each of these $4 \times 2 = 8$ tasks, the target receptacle is the brown bowl. For the final 2 tasks, all objects are placed onto the table, and the policy must put the mug into each of the receptacles.

Workroom The workroom environment consists of 5 objects (tape, wooden spoon, timer, copper mug, blue mug) and 2 receptacles (tray, box) on the left of the workspace. Each object must be placed into each receptacle for a total of 10 tasks. When evaluating on the mugs or timer, these objects are placed together on the table in the middle of the workspace. When evaluating on the tape or spoon, they are placed along with an additional spork (a distractor) in the middle of the workspace.

Bedroom The bedroom environment consists of 4 objects (pill bottle, lint roller, banana, tennis ball) and 2 receptacles (towel, basket). Each object is placed into each receptacle for $4 \times 2 = 8$ tasks, and for the final 2 tasks the policy must put the banana into each receptacle, but with a cluttered workspace. This environment notably does not feature a table as a support surface, but instead a bed, which tests robustness environment diversity.

Office The office environment features 8 objects (knife, banana, marker, scissors, carrot, screwdriver, computer mouse, mug) and 7 receptacles (cutting board, plate, mug, green bowl, blue bowl, basket, box), with multiple object configurations with varying amounts of clutter and distractors. Evaluations in this environment were conducted in an entirely different institution and geographical location, illustrating MolmoBot policies' ability to get up and running in completely new settings.

O. Experiments — Mobile Manipulation Experiments

We evaluate the MolmoBot Door Specialist policy on a door opening task in three real-world environments, pictured in Fig. 6, each featuring a distinct pull door with different visual textures, handle configurations, and surrounding scene context. Each environment also features distinct wall geometry and background clutter, testing the generalization of the policy across varied visual conditions. Episodes were executed at a 100ms inference timestep for safety reasons, which is slower than the timestep used during simulation evaluation.

Failure analysis. Aside from hardware failures, a recurring source of failure across Door 1 and Door 3 was difficulty grasping the handle. Both of these doors have handles positioned on the right side of the door, a configuration that is underrepresented in typical door interaction datasets and in our training data, which may explain the policy's reduced grasping reliability in these cases. In contrast, Door 2, whose handle configuration was more commonly represented, saw successful grasps in all three trials.

P. Experiments — Simulation Evaluation

Setup. We run our experiments with the following hyper-parameters. A policy_dt of $66ms$, task_horizon 303 steps / 20 seconds (pick tasks) and 606 steps / 40 seconds (for pick-and-place tasks). When using the filament renderer we set the

Benchmark	Task ID(s)	Task Prompt
Workroom	spoon_tray	"put the wooden spoon on the light blue tray"
	spoon_box	"put the wooden spoon in the wooden box"
	tape_tray	"put the blue tape on the light blue tray"
	tape_box	"put the blue tape in the wooden box"
	blue_mug_tray	"put the blue mug on the light blue tray"
	blue_mug_box	"put the blue mug in the wooden box"
	copper_mug_tray	"put the copper mug on the light blue tray"
	copper_mug_box	"put the copper mug in the wooden box"
	timer_tray	"put the green timer on the light blue tray"
	timer_box	"put the green timer in the wooden box"
Kitchen	apple_easy, apple_hard	"put the apple in the brown bowl"
	clutter_brown, mug_easy, mug_hard	"put the mug in the brown bowl"
	banana_easy, banana_hard	"put the banana in the brown bowl"
	mouse_easy, mouse_hard	"put the computer mouse in the brown bowl"
	clutter_black	"put the mug in the black bowl"
Bedroom	pills_towel	"put the pill bottle on the yellow towel"
	pills_basket	"put the pill bottle in the basket"
	roller_towel	"put the lint roller on the yellow towel"
	roller_basket	"put the lint roller in the basket"
	banana_towel, clutter_towel	"put the banana on the yellow towel"
	banana_basket, clutter_basket	"put the banana in the basket"
	ball_towel	"put the tennis ball on the yellow towel"
	ball_basket	"put the tennis ball in the basket"
Office	knife_board	"put the knife on the cutting board"
	banana_plate	"move the toy banana on the plate"
	marker_mug	"put the marker into the mug"
	scissors_bowl	"pick up the scissor and place it inside the bowl"
	carrot_basket	"put the carrot into the basket"
	knife_green_bowl	"pick up the knife and place it inside the green bowl"
	screwdriver_blue_bowl	"put the screwdriver in the blue bowl"
	mouse_blue_bowl	"place the mouse into the blue bowl"
	mug_bowl	"grasp the mug and put it inside the bowl"
	marker_box	"pick up the red marker and place it into the box"

TABLE VI: Task prompts for each task in each benchmark for our real-world DROID pick-and-place evaluations.

environment illumination to 12000 candela by default. Additionally, we note that StereoVLA was trained based on a front-on view of the robot; in order to run StereoVLA we modify our benchmark by moving cameras into this position, and we additionally filter out episodes where the target object is not visible. This yields 92 episodes for the Pick Classic task and 91 episodes for the Pick task. Despite these accommodations for StereoVLA, the performance remains low.

Tasks. We evaluate on a progression of increasingly difficult tasks (Tab. I) in 1000 episode benchmarks. We begin with a simple pick task in a controlled configuration and limited object diversity (*Pick MSProc*). The next set of tasks introduces more challenging object and viewpoint distributions in three variants: standard MuJoCo rendering (*Pick Classic*), photorealistic filament rendering (*Pick*) which is out of distribution for our training data, and heavily randomized camera viewpoints (*Pick Random-Cam*). Pick tasks are allotted 20 seconds for completion. We then evaluate pick-and-place variants including placing objects inside a receptacle (*Pick&Place*), next to a target (*PnP Next-To*), and in a receptacle of a specified color (*PnP Color*), all using filament rendering. We consider a Next-To episode as a success if the object to be moved is placed within 5 cm of the target object. Pick-and-place tasks are allotted 40 seconds due to their increased difficulty. We report only oracle success (task completed at any timestep) for pick tasks, as termination behavior is not well-defined for object lifting. For pick-and-place tasks, we report both final success rate and oracle success rate. The gap between these metrics reflects whether the policy can recognize task completion and disengage appropriately rather than continuing to manipulate the object after placement.

Results. Our models outperform all baselines across tasks. On the least-variation *Pick MSProc* task, MolmoBot (F=2) achieves 93.5% success compared to 48.0% for the strongest baseline ($\pi_{0.5}$-Finetune). The gap widens on more challenging distributions: on *Pick Random-Cam*, our models achieve 40–66% success while $\pi_{0.5}$ variants reach only 8–30%. Other VLA baselines (StereoVLA, LAP-VLA, X-VLA) fail almost entirely on our evaluation suite, with success rates below 7% on most tasks. On pick-and-place tasks, which require both grasping and placement, MolmoBot variants achieve 63–67% oracle success on *Pick&Place*. $\pi_{0.5}$-Finetune achieves 43.5% oracle success on this task. Notably, our models generalize to compositional instructions (*PnP Color*) where they must identify the correct receptacle by color, achieving 57–62% final success. Averaging

Benchmark	Policy	*Spoon Tray*	*Spoon Box*	*Tape Tray*	*Tape Box*	*Blue Mug Tray*	*Blue Mug Box*	*Copper Mug Tray*	*Copper Mug Box*	*Timer Tray*	*Timer Box*	**Avg**
Workroom	MolmoBot	3/3	2/3	3/3	3/3	3/3	3/3	3/3	1/3	3/3	3/3	**90%**
	MolmoBot-Img	3/3	3/3	3/3	3/3	3/3	2/3	2/3	0/3	3/3	1/3	77%
	MolmoBot-Pi0	1/3	0/3	3/3	3/3	3/3	3/3	2/3	1/3	0/3	2/3	60%
	$\pi_{0.5}$-DROID	2/3	2/3	2/3	0/3	1/3	1/3	0/3	0/3	0/3	0/3	27%
	π_0	0/3	0/3	1/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	3%
		Apple Easy	*Apple Hard*	*Mug Easy*	*Mug Hard*	*Banana Easy*	*Banana Hard*	*Mouse Easy*	*Mouse Hard*	*Clutter Brown*	*Clutter Black*	**Avg**
Kitchen	MolmoBot	1/3	2/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	0/3	70%
	MolmoBot-Img	3/3	3/3	2/3	3/3	3/3	3/3	3/3	2/3	3/3	1/3	**87%**
	MolmoBot-Pi0	2/3	3/3	2/3	0/3	3/3	1/3	3/3	2/3	0/3	0/3	53%
	$\pi_{0.5}$-DROID	3/3	0/3	2/3	1/3	3/3	2/3	3/3	1/3	2/3	2/3	63%
	π_0	0/3	1/3	0/3	0/3	1/3	0/3	3/3	1/3	0/3	0/3	20%
		Pills Towel	*Pills Basket*	*Roller Towel*	*Roller Basket*	*Banana Towel*	*Banana Basket*	*Ball Towel*	*Ball Basket*	*Clutter Towel*	*Clutter Basket*	**Avg**
Bedroom	MolmoBot	3/3	3/3	3/3	0/3	3/3	2/3	3/3	3/3	3/3	3/3	**87%**
	MolmoBot-Img	0/3	0/3	1/3	2/3	3/3	3/3	3/3	2/3	3/3	3/3	67%
	MolmoBot-Pi0	2/3	0/3	2/3	0/3	0/3	0/3	1/3	0/3	0/3	2/3	23%
	$\pi_{0.5}$-DROID	0/3	0/3	0/3	0/3	1/3	0/3	0/3	0/3	2/3	0/3	10%
	π_0	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0%
		Knife Board	*Banana Plate*	*Marker Mug*	*Scissors Bowl*	*Carrot Basket*	*Knife Green Bowl*	*Screwdriver Blue Bowl*	*Mouse Blue Bowl*	*Mug Bowl*	*Marker Box*	**Avg**
Office	MolmoBot	2/3	3/3	1/3	2/3	2/3	2/3	3/3	2/3	3/3	1/3	**70%**
	MolmoBot-Img	2/3	1/3	0/3	1/3	3/3	1/3	3/3	2/3	3/3	2/3	60%
	MolmoBot-Pi0	1/3	3/3	0/3	0/3	0/3	1/3	3/3	2/3	3/3	2/3	50%
	$\pi_{0.5}$-DROID	2/3	3/3	1/3	1/3	1/3	1/3	2/3	1/3	3/3	2/3	57%
	π_0	0/3	0/3	0/3	0/3	1/3	1/3	1/3	1/3	0/3	0/3	13%

TABLE VII: DROID real-world evaluation results on each task in each environment. Each cell shows successes out of 3 trials.

across simulation tasks, MolmoBot (F=2) achieves 64.1% compared to 10.1% for $\pi_{0.5}$ zero-shot. MB-Pi0, which uses the π_0 architecture trained on our data, achieves 41.8%—substantially higher than $\pi_{0.5}$ zero-shot and competitive with $\pi_{0.5}$-Finetune on pick tasks—demonstrating that much of the performance gain comes from MolmoBot-Data rather than architectural differences. On real-world evaluation, MolmoBot (F=2) achieves 79.2% success, compared to 31.3% for $\pi_{0.5}$. Interestingly, $\pi_{0.5}$-Finetune is competitive with our best policies on *PnP NextTo*, achieving best oracle success though not final success. This supports the notion that fine-tuning real-world-native policies on sim data to bridge the real-to-sim gap enables reasonable comparisons, further bolstered by our results in the real world.

Tab. IX evaluates models using only the fixed-shoulder camera, a more constrained setup that matches typical single-camera deployments. We test in simulation (*Pick MSProc*, 1000 episodes) and on 30 real-world trials in a kitchen environment (*Pick Kitchen*).

In simulation, our models maintain strong performance: MolmoBot variants achieve 91–93% success, while MolmoBot-SPOC reaches 70.4%. $\pi_{0.5}$-Finetune achieves 48.0% and $\pi_{0.5}$ zero-shot 18.1%. DreamZero performs zero-shot competitively at 44.3% while StereoVLA and LAP-VLA again perform poorly (6.6% and 19.4%).

On real-world evaluation, our models demonstrate successful zero-shot sim-to-real transfer. MolmoBot-Img peaks on this real world subset at 86.6% followed by MolmoBot (F=3) at 73.3% success and MolmoBot (F=2) at 70.0%. All variants

TABLE VIII: Door opening task results. Each door has a distinct visual texture. Trials differ in robot base position. †Hardware fault occurred during trial.

Door	Trial	Grasp	Opened	Failure Mode
1	1	✓†	✗	Grasping HW fault
	2	✗	✗	Base collision
	3	✗	✗	Joint limits
2	1	✓†	✗	Grasping HW fault
	2	✓	✓†	Opening HW fault
	3	✓	✓†	Opening HW fault
3	1	✗	✗	Base collision
	2	✗†	✗	Approach HW fault
	3	✗	✗	Grasping failure

TABLE IX: Simulation and real evaluation with restricted camera setup. Success rate averaged over 1000 episodes in simulation and 30 tasks in a real-world kitchen. All models evaluated zero-shot without task-specific finetuning.

Model	Pick MSProc (sim)	Pick Kitchen (real)
StereoVLA [12]	6.6	–
LAP-VLA [45]	19.4	–
π_0[3]	13.5	20.0
$\pi_{0.5}$[4]	18.1	63.3
$\pi_{0.5}$-Finetune	48.0	–
DreamZero [43]	44.3	–
MolmoBot-Pi0	66.2	53.3
MolmoBot-SPOC	70.4	36.6
MolmoBot-Img	92.2	**86.6**
MolmoBot (F=2)	**93.5**	70.0
MolmoBot (F=3)	91.3	73.3

perform better than $\pi_{0.5}$ zero-shot at 63.3%. The MolmoBot-SPOC achieves 36.6%, lower than the VLA variants but notable given its substantially smaller size and suitability for edge deployment. These results confirm that policies trained entirely on MolmoBot-Data transfer to real environments without fine-tuning.

External simulation benchmarks. We additionally evaluate on external benchmarks SIMPLER [22] and LIBERO [23], reimplemented for our DROID setup. These benchmarks were designed for in-domain evaluation of policies trained on specific demonstration datasets (RT-1/Bridge and LIBERO demonstrations, respectively) rather than zero-shot generalization. We find that while MolmoBot outperforms baselines on these benchmarks, the tight coupling between task specifications and benchmark-specific assets severely limits their utility for assessing generalist manipulation policies. Full results and discussion are provided in Appendix S.

Task Name	Objects	Episodes	Renderer	Camera
Pick-MSProc	Thor	1000	MuJoCo	Droid
Pick-Classic	Objaverse	200	MuJoCo	Droid-Light
Pick	Objaverse	200	Filament	Droid-Light
Pick-Rnd-Cam	Objaverse	200	Filament	Rnd.-Cam.
PnP-Next-To	Objaverse	200	Filament	Droid-Light
PnP-Color	Objaverse	200	Filament	Droid-Light

RB-Y1 Simulation Evaluation. Table X reports zero-shot simulation performance across all RB-Y1 policies. MolmoBot Multitask outperforms MolmoBot-SPOC across all shared tasks, which we attribute to several factors. First, MolmoBot's frozen VLM backbone provides rich visual representations that generalize well without task-specific finetuning, whereas MolmoBot-SPOC's smaller transformer architecture has more limited capacity. Second, MolmoBot Multitask was trained jointly

TABLE X: Simulation evaluation for RB-Y1 policies on held-out environments. All models evaluated zero-shot without task-specific finetuning.

Model	Pick	PnP	Open	Door Open
MB Multitask	44.8	22.5	25.2	70.2
MB Door	–	–	–	77.7
MB-SPOC Rigid	10.5	1.8	–	–
MB-SPOC Artic.	–	–	21.8	58.8

across all tasks, which may have enabled positive transfer between related manipulation behaviors. Although MolmoBot-SPOC demonstrates more modest performance in these evaluations, its compact scale enables future on-policy reinforcement learning in simulation, which has been shown to yield substantial performance gains [15].

Q. Experiments — Data Ablations

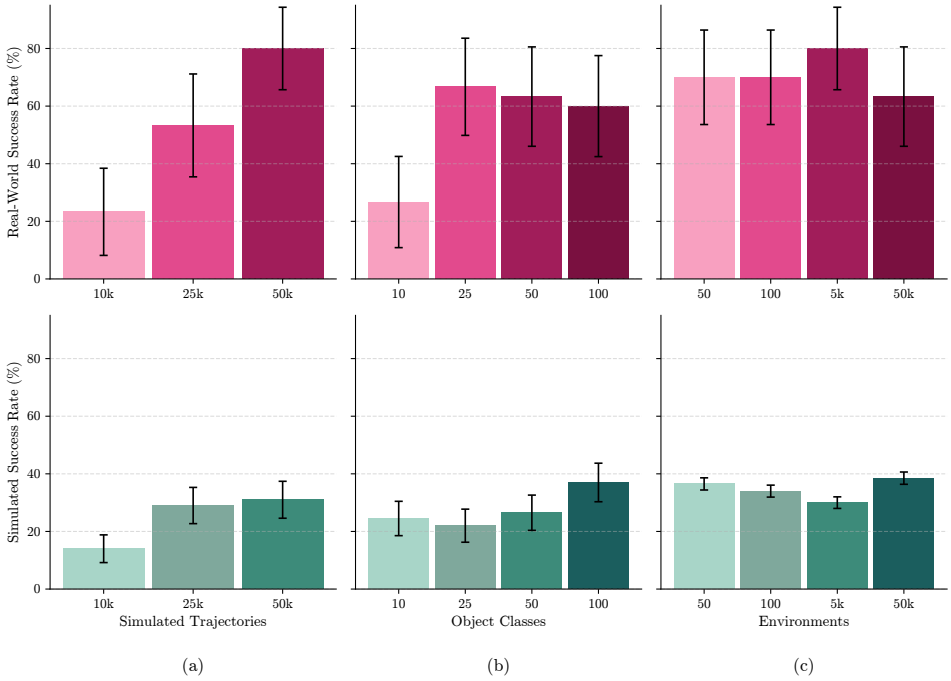

Fig. 7: Effect of scaling various dimensions of simulated training data, evaluated on DROID (real-world, top) and pick classic (simulation, bottom). **(a)** We vary trajectory count sampled from 5,000 houses; performance improves predictably with scale, particularly in real-world evaluations. **(b)** We control the number of unique object classes across 50,000 trajectories from 5,000 houses. Object diversity improves simulation performance but not real-world, possibly due to the evaluation using fewer, more general objects. **(c)** We vary the number of simulated house environments across 50,000 trajectories. Environment diversity does not improve performance in either setting, suggesting the local nature of pick makes background diversity unnecessary. Error bars: 95% binomial CIs.

In this section, we study how several properties of the training data affect performance, including data scale, the number of unique objects, the number of unique houses, and image augmentation. We observe some expected trends, such as performance improving monotonically as the amount of training data increases. We also find some surprising results, such as increasing the number of unique house environments having little effect on performance. All data ablations report performance on the pick task for the MolmoBot-Img model trained for $24K$ steps with batch size 512.

Evaluation Details Unlike Sec. IV-A, the data ablation real-world evaluations are on the *pick* task. The evaluations are performed with the DROID platform in the workroom environment (pictured in Fig. 5), but with different objects: a roll of blue tape, mug, aerosol can, banana, pill bottle, and wooden spoon. Evaluations are done in three groups of three objects, with five trials for each object. For object class ablations, we adjust the object list slightly such that two out of the six objects are strictly in unseen classes for all top 100: apple and screwdriver replace the tape and wooden spoon. Success is judged by the evaluator when the object is fully off the table by approximately 2cm or more. Our simulation results are for *Pick-Classic*, as detailed in Sec. IV-B. For each picking evaluation, the task prompt is formatted as: "pick up the `<object>`".

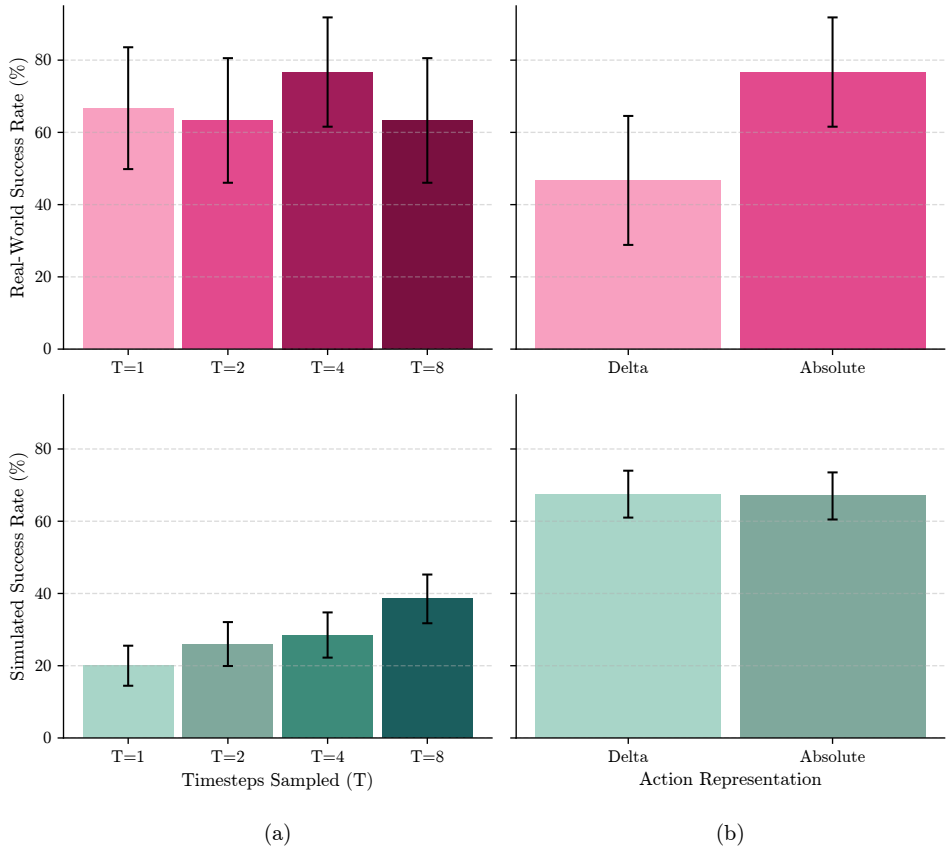

Fig. 8: Ablations on training and action parameterization. We evaluate MolmoBot-Img on both DROID (real-world, top row) and pick classic (simulation, bottom row). **(a)** We train while sampling multiple denoising timesteps T per example in parallel during training to improve convergence and final performance. We ablate $T \in \{1, 2, 4, 8\}$ and find that simulation performance improves steadily as T increases and peaks at $T = 8$, while real-world performance is less monotonic and peaks at $T = 4$. **(b)** We train using either absolute or delta action representations for 200K steps on Franka FR3 policies. The absolute action representation substantially improves real-world performance over delta actions, while the two representations perform similarly in simulation. Error bars denote 95% binomial proportion confidence intervals.

Scaling Number of Demonstrations

To study how performance changes with the data scale, we vary the total number of training demonstrations while keeping the number of house environments and object classes fixed. Concretely, we train MolmoBot-Img on datasets containing 10K, 25K, and 50K demonstrations sampled from the same set of 5K environments and 12.4K object categories. We evaluate the model for the pick task in both simulation and real. We observe predictable scaling trends as pick performance for both domains improves with the number of demonstrations (figure 7 (a)).

Scaling Environment Diversity For this ablation we vary the number of unique houses in the training set while keeping the total number of demonstrations fixed. Specifically, we contrast many demonstrations from fewer houses with fewer demonstrations spread across more houses. We fix the data scale at 50K trajectories. Unexpectedly, we find that increasing the number of unique training environments has little effect on downstream performance (Figure 7). This suggests that, for the pick task, performance is driven more by the total amount of interaction data than by scaling environment diversity.

Scaling Object Diversity We train MolmoBot-Img with a fixed number of 50K trajectories while sampling from 5 to 100 objects. We find that performance improves as expected for the simulated evaluation (Figure 7 (c)). However, the performance on DROID does not have a clear trend with respect to object diversity. We hypothesize that the number of objects beyond a small number does not improve performance on DROID because the number of objects in the evaluation is limited and semantically common such as apple and cup.

R. Experiments — Model Ablations

As in Sec. Q, our model ablation experiments are run in the real-world on the *pick* task in the workroom environment, and on *Pick-Classic* in simulation.

Timesteps sampled during training We sampled multiple time steps T per example and denoise in parallel during to improve the convergence and the accuracy of the model. We ablate the choice for $T \in \{1, 2, 4, 8\}$ during training and report the performance of MolmoBot-Img (Figure 8). Performance on simulation benchmarks improves as T increases and peaks at $T = 8$, suggesting the increase T helps. However, while the result on real subset of 30 examples is not as clear, with the performance peaking at $T = 4$.

Action representation. We compare MolmoBot-Img trained using absolute and delta representations on the complete multi-task data mix for $200K$ steps each for the Franka FR3 policies (Figure 8). On the Franka FR3 task, the absolute policy significantly outperforms the delta policy in real setting, while the simulation results are on-par for both policies. The significant gap in real across 3 of our benchmarks strongly suggests that absolute joint policy models transfer better to real world tasks.

S. Experiments — Zero-Shot Evaluation on External Simulation Benchmarks

Prior works often evaluate on popular existing benchmarks, including SIMPLER [22] and LIBERO [23]. Both benchmarks were designed to evaluate policies trained on specific demonstration datasets (RT-1/Bridge and LIBERO demonstrations, respectively) and correlate with real-world performance under those in-distribution conditions. Additionally, these benchmarks use specific robot embodiments (WidowX and Franka FR3 with Panda Hand, respectively), which further complicates zero-shot evaluation of generalist policies. To evaluate on these benchmarks without benchmark-specific finetuning, a policy would have to be not only generalizable, but also cross-embodiment, posing a very high bar.

Therefore, we introduce and evaluate on SIMPLER-DROID and LIBERO-DROID. We replace the WidowX in SIMPLER and the non-DROID Franka in LIBERO with the DROID platform, which enables zero-shot evaluation of DROID policies, improving unification and reproducibility within the community. Specifically, we remove the existing robot from these benchmarks and replace it with a DROID platform, offset by a fixed transformation to account for differing embodiment size. Cameras are also swapped out, mirroring the DROID camera setup (ZED-Mini wrist camera and ZED 2 exocentric camera).

We evaluate MolmoBot policies and baselines—using no in-domain data or task-specific fine-tuning—on these adapted benchmarks, and our findings suggest structural limitations in their suitability for assessing generalist policies.

SIMPLER. SIMPLER provides simulated evaluation environments designed to correlate with real-world performance for policies trained on RT-1 and BridgeData V2. We reimplemented four WidowX tasks for our Franka setup. Results are shown in Table XI.

TABLE XI: Zero-shot evaluation on SIMPLER tasks reimplemented for DROID Franka FR3.

Model	Carrot→Plate	Eggplant→Basket	Stack Cubes	Spoon→Cloth	Avg.
MolmoBot-Img	45.8%	0%	0%	0%	11.5%
MolmoBot-Pi0	0%	0%	0%	0%	0%
$\pi_{0.5}$-DROID	0%	0%	0%	0%	0%
π_0-DROID	0%	0%	0%	0%	0%

MolmoBot-Img achieves 45.8% on "put carrot on plate," but all models fail on the remaining tasks. We found that the prompts used in the dataset were out of distribution for most models and need fine-tuning on that type of task. As reported, the task with the prompt "put spoon on towel," led to 0% performance for all models but tweaking it to "put the spoon on the towel," led to more successful episodes. We leave prompt improvement to future work and report results with the exact prompt in the benchmark.

LIBERO. LIBERO is a lifelong learning benchmark with 130 manipulation tasks. We evaluate on LIBERO-Object, which requires picking specific grocery products (e.g., "alphabet soup," "bbq sauce") and placing them in a basket. To diagnose failure sources, we evaluate both with and without distractor objects. Results are shown in Table XII.

MolmoBot-Img outperforms all baselines (36.6% vs. 3.8% average in the no-distractor setting), preserving the rank ordering observed in our real-world evaluations. The no-distractor ablation shows that visual clutter contributes to failure, but even simplified scenes yield low performance—the task descriptors reference specific brand-name products whose visual appearance differs from objects in MolmoBot-Data. The high task-level variance (78% on "chocolate pudding" vs. 2% on "butter") further suggests performance depends on incidental overlap with benchmark-specific assets rather than general manipulation competence.

Discussion. These benchmarks measure familiarity with specific assets and scene configurations rather than zero-shot manipulation capability. While MolmoBot models preserve rank-order superiority over baselines—consistent with our real-world findings—the absolute performance is not indicative of the models' true manipulation competence. In contrast, our MolmoSpaces-based evaluation (Sec. IV-B) uses 94k+ procedurally generated environments with 11k+ object assets, providing sufficient diversity to assess generalization while maintaining validated correlation with real-world performance.

TABLE XII: Zero-shot evaluation on LIBERO-Object reimplemented for DROID Franka FR3. We evaluate with the standard set of distractors (std.) and without distractors (no dist.) to isolate failure sources.

	MolmoBot-Img		**MolmoBot-Pi0**		$\pi_{0.5}$**-DROID**		π_0**-DROID**	
Task	std.	no dist.	std.	no dist.	std.	no dist.	std.	no dist.
Alphabet soup	0%	16%	0%	2%	0%	2%	0%	0%
Cream cheese	0%	24%	0%	2%	2%	0%	0%	0%
Salad dressing	20%	40%	0%	2%	0%	0%	0%	0%
BBQ sauce	0%	42%	0%	14%	0%	2%	0%	0%
Ketchup	0%	42%	0%	6%	0%	4%	0%	0%
Tomato sauce	0%	52%	2%	6%	2%	8%	0%	0%
Butter	0%	2%	0%	0%	0%	2%	0%	0%
Milk	0%	26%	4%	2%	4%	0%	0%	0%
Choc. pudding	10%	78%	0%	2%	0%	6%	0%	0%
Orange juice	50%	44%	2%	2%	2%	0%	0%	2%
Average	8%	36.6%	0.8%	3.8%	1%	2.4%	0%	0.2%