Scaffolding Dexterous Manipulation with Vision-Language Models

Vincent de Bakker*†, Joey Hejna*, Tyler Ga Wei Lum*, Onur Celik†, Aleksandar Taranovic†,
Denis Blessing†, Gerhard Neumann†, Jeannette Bohg*, Dorsa Sadigh*
*Stanford University, †Karlsruhe Institute of Technology

***Abstract*—Dexterous robotic hands are essential for performing complex manipulation tasks, yet remain difficult to train due to the challenges of demonstration collection and high-dimensional control. Thus, contemporary works in dexterous manipulation have often bootstrapped from reference trajectories to trajectories specify target hand poses that guide the exploration of RL policies and object poses that enable dense, task-agnostic rewards. However, sourcing suitable trajectories—particularly for dexterous hands—remains a significant challenge. Our key insight is that modern vision-language models (VLMs) already encode the commonsense spatial and semantic knowledge needed to specify tasks and guide exploration effectively. Given a task description (e.g., "open the cabinet") and a visual scene, our method uses an off-the-shelf VLM to first identify task-relevant keypoints (e.g., handles, buttons) and then synthesize 3D trajectories for hand motion and object motion. Subsequently, we train a low-level residual RL policy in simulation to track these coarse trajectories or "scaffolds" with high fidelity.** [1]

I. Introduction

Dexterous manipulation is essential for a range of real-world tasks – such as using a power-drill or twisting a door knob – which require the fine-grained control offered by human-like hands [2]. Despite the intrinsic advantages of dexterous hands over simpler end-effectors, existing learning paradigms have struggled to cope with their complexity [50]. The prevailing approach for training generalist policies – imitation learning from demonstrations [5, 43] – has achieved limited success with robot hands, primarily due to the challenges of collecting accurate data with dexterous hardware, resulting in a scarcity of high-quality demonstrations [48, 60]. While alternative approaches attempt to re-target demonstrations from easier interfaces [27, 20, 65, 67, 51, 72, 47, 14], e.g., human hands, such approaches often induce irrecoverable errors for fine-grained tasks.

To avoid both data scarcity and the embodiment gap, a combination of reinforcement learning (RL) and sim-to-real transfer has emerged as a promising approach by enabling large-scale experience generation [3]. However, using RL simply shifts the burden from data collection to reward design. Standard RL approaches for dexterous manipulation necessitate hand-crafting complex, task-specific reward functions. A large amount of this complexity arises from the need to guide exploration; with large action spaces, dexterous hands need to be coaxed towards the correct part of the observation space to make progress on a task. Thus, various approaches have used demonstrations to bootstrap the RL process [50, 17, 41, 16, 49]. In dexterous manipulation, this is often done through trajectory tracking, where instead of designing a complex reward function, a policy is rewarded for tracking the exact wrist and object positions in a demonstration, leaving RL to only make adjustments [7, 73]. By re-framing dexterous manipulation as a trajectory tracking problem, such approaches can leverage dense, task-agnostic rewards and guide exploration by using residual policies [21, 13].

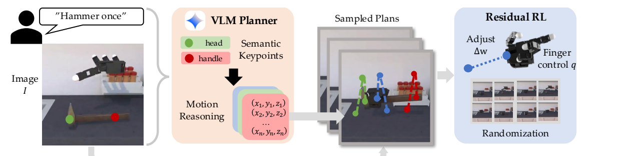

Fig. 1: Overview of our method: a VLM generates hand and object keypoint trajectories from a language command and scene image. A low-level residual RL policy is trained to track these trajectories in simulation.

Though demonstration tracking overcomes the design challenges associated with RL, it paradoxically re-introduces the same dependence on demonstrations we sought to avoid in the first place. For example, prior works [7, 73] that use tracking-based RL for dexterous manipulation often require large prior datasets with thousands of teleoperated demonstrations [11, 70], restricting the method to tasks for which data has already been collected.

Our key insight is that coarse motion plans ("scaffolds") can be sufficient for both of these criteria. Generating such plans only requires high-level spatial and semantic reasoning, the exact abilities afforded by new advancements in vision-language models (VLMs). Consequently, VLMs have the potential to supply the high-level reward signals and exploratory guidance needed for RL through coarse motions. So long as these motions generally encapsulate the desired behavior, RL can optimize per-timestep offsets and finger motions to maximize the tracking reward, ultimately surpassing human teleoperation in both performance and precision, eliminating the reliance on demonstrations.

Building upon this insight, we introduce a framework for learning manipulation policies for dexterous robot hands with VLM-generated motion plans and residual RL. Given a natural language instruction (e.g., "hammer once" Fig. 1) and image, an off-the-shelf VLM first identifies relevant object keypoints. Then, provided the initial keypoints and hand pose, the VLM generates the associated 3D trajectories for both object and hand motions to define the supervision targets for a "low-

[1] Videos at https://sites.google.com/view/vlm-scaffolding/home

level" residual RL policy trained in simulation.

We evaluate our method across a suite of challenging dexterous manipulation tasks in simulation requiring semantic understanding, human knowledge about concepts like "hammering", and precise manipulation for difficult or articulated objects.

II. Dexterous Manipulation via VLM Feedback

We focus on dexterous manipulation using robotic hands with visual observations and natural language instructions, with the aim of developing a general approach transferable across diverse applications and settings. Following prior work [7, 4], we adopt a hierarchical approach that naturally delineates planning and control. However, instead of centering plans around demonstrations, we leverage a VLM to produce coarse plans sufficient to "scaffold" low-level RL. We interface between these two components using 3D keypoints, as they provide sufficient precision for effective manipulation [62, 68], yet are abstract enough for VLM reasoning [12, 39] and often used during pre-training [58, 22].

A. Problem Formulation

Our goal is to learn a hierarchical policy for dexterous manipulation, where the high- and low-level policies interface via 3D keypoint-based plans or trajectory "scaffolds". While several prior works assume access to ground-truth states (often in simulation), such information is typically only partially observable in practice. For example, it is unrealistic to assume that one is able to precisely measure object poses and velocities in the real world. Only the dexterous hand's proprioceptive state $(\mathbf{w}, \mathbf{q}, \dot{\mathbf{q}})$ comprised of the current wrist pose $\mathbf{w} \in \mathrm{SE}(3)$, finger joint positions $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$ is exactly known. Instead of ground-truth states we assume access to RGB images I, depth D, and a language instruction L which communicates the task. Following standard practice in dexterous manipulation, we use an absolute action space comprised of desired wrist $\mathbf{w}^{\text{targ}}$ and finger joint positions $\mathbf{q}^{\text{targ}}$, i.e., $(\mathbf{w}^{\text{targ}}, \mathbf{q}^{\text{targ}}) \in \mathcal{A}$.

The high-level policy π^h produces a coarse, 3D keypoint-based plan τ from the language instruction L and an initial high-level observation o_1^h at time $t = 1$ containing the initial image I_1 and wrist position $\mathbf{w}_1$. As we instantiate π^h using a VLM, we assume the ability to project 2D keypoints $\mathbf{u}^{(i)} \in \mathbb{R}^2$ in image space to 3D keypoints $\mathbf{x}^{(i)} \in \mathbb{R}^3$ in world coordinates, which is easily accomplished in practice using depth information D and camera parameters (intrinsic and extrinsic). The number of 3D keypoints k in the final plan τ is specified through the instruction L. We enumerate these keypoints as $\mathbf{x}^{(1)}, \dots \mathbf{x}^{(k)}$ and abbreviate sequences of length T through time via the short-hand $1:T$. The final keypoint plan τ includes k 3D keypoint sequences $\mathbf{x}_{1:T}^{(1)}, \dots \mathbf{x}_{1:T}^{(k)}$ and a sequence of predicted wrist poses $\tilde{\mathbf{w}}_{1:T}$. This coarse plan encapsulates both information about the task via the k keypoint sequences which can capture object movements, and information to guide the agent's exploration via the wrist position $\mathbf{w}$.

The high-level policy can be written as:

$$\pi^h(\underbrace{\tilde{\mathbf{w}}_{1:T}, \mathbf{x}_{1:T}^{(1)}, \dots \mathbf{x}_{1:T}^{(k)}}_{\tau} \,|\, \underbrace{I_1, \mathbf{w}_1}_{o_1^h}, L) \tag{1}$$

The high-level policy only provides a coarse plan for the wrist $\mathbf{w}$ – not the finger joint positions $\mathbf{q}$ which will be learned by the low-level policy with RL.

The low-level policy π^l produces wrist and finger actions a_t to execute the keypoint plan τ. We assume access to a keypoint tracking model, which given an initial 3D keypoint $\mathbf{x}_1^{(i)}$ at time $t = 1$ is able to track its position over time to produce estimates $\hat{\mathbf{x}}_t^{(i)}$. The low-level policy π^l is then optimized via RL using a reward function that encourages consistency between the estimated 3D keypoints $\hat{\mathbf{x}}_t^{(i)}$ and those produced by the plan τ, $\mathbf{x}_t^{(i)}$. To accomplish this task, it takes as input both a low-level observation $o_t^l \in \mathcal{O}^l$, consisting of the proprioceptive state $(\mathbf{w}, \mathbf{q}, \dot{\mathbf{q}})$ and estimated keypoints $\hat{\mathbf{x}}^{(i)}, \dots, \hat{\mathbf{x}}^{(i)}$, and all future steps of the plan $\tau_{t:T}$. Succinctly,

$$\pi^l(\underbrace{\mathbf{w}_t^{\text{targ}}, \mathbf{q}_t^{\text{targ}}}_{a_t} \,|\, \underbrace{\mathbf{q}_t, \dot{\mathbf{q}}_t, \mathbf{w}_t}_{\text{proprio}}, \underbrace{\hat{\mathbf{x}}_t^{(1)}, \dots, \hat{\mathbf{x}}_t^{(k)}}_{\text{keypoint estimates}}, \underbrace{\tilde{\mathbf{w}}_{t:T}, \mathbf{x}_{t:T}^{(1)}, \dots, \mathbf{x}_{t:T}^{(k)}}_{\text{plan } \tau_{t:T}}). \tag{2}$$

Provided the high- and low-level decomposition of our approach, we now describe each component.

B. Trajectory Generation for High-Level Policies via VLMs

We implement the high-level policy π^h using a VLM, which must be able to effectively translate the task description L and initial image I_1 into a coarse motion plan τ for π^l to complete.

Keypoint Detection. First, the VLM identifies k 2D keypoints $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(k)}$ in the image I that are relevant to completing the task described via text L. The VLM is prompted with useful keypoints for the task. For example, the keypoints include both the handle and head of a hammer for the "hammering" task (Fig. 1) or the position of an object and its desired location (Fig. 2) for semantic pick-place. The full prompts used are included in Appendix H. Since the VLM operates in the 2D image plane, we lift 2D keypoints $\mathbf{u}$ to 3D world coordinates $\mathbf{x}$ using depth information.

Trajectory Generation. Second, provided the text description l, the VLM generates waypoint sequences of length $n < T$ for each of the initial 3D keypoints $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(k)}$ and the wrist position $\mathbf{w}_1$. In total, this results in $(k+1) \times n$ 3D waypoints which will serve as the basis of the plan τ. We include the full prompts used in Appendix H. While the first keypoint detection stage depends on the VLM's image understanding, this phase depends more on spatial understanding and reasoning – the VLM must translate semantic descriptions into motions, e.g., what "hammering" implies or how a door opens, while respecting the physical constraints between keypoints and proximity between the hand and manipulated objects. Note that we do not have the VLM produce keypoint trajectories of the full horizon T, as doing so might be more difficult and inaccurate. Instead, we posit the quality of each waypoint matters more than the number, as low-level RL can compensate

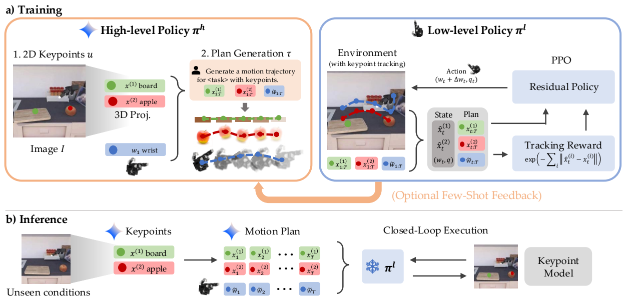

Fig. 2: a) Training: a high-level VLM predicts 3D keypoint plans, which a low-level policy learns to track via RL. b) Inference: new plans are generated by the VLM, which are executed by the frozen low-level policy.

for small mistakes in position but not large errors in reasoning. Afterwards, we interpolate the waypoints to length T.

Few-Shot Improvement. Though VLM-generated keypoint plans τ are often correct, they are not infallible. However, the accuracy of VLMs can often be improved by providing in-context examples [36, 9]. Provided m successful plans $\tau^{(1)}, \ldots, \tau^{(m)}$, we can prompt the high-level policy as $\pi^h(\tau | s_1, \tau^{(1)}, \ldots, \tau^{(m)})$ to produce better plans for the low-level policy.

C. Low-Level Control with Reinforcement Learning

The low-level policy π^l ensures that the keypoint plan τ provided by π^h is effectively tracked. We learn π^l using residual reinforcement learning [21, 13], which we formalize through a "plan" conditioned MDP on top of the low-level observation space $\mathcal{O}^l$ and action space $\mathcal{A}$ with horizon T. We assume the dynamics to be stochastic $p(o_{t+1}|o_t, a_t)$ to account for noise in keypoint estimation and that the initial state $o_1^l \sim p_\tau^{\text{init}}$ is always consistent with the high-level plan τ to ensure its validity. Naïvely, π^l is optimized to maximize the expected cumulative reward provided plans sampled from π^h, $\max_{\pi^l} \mathbb{E}_{\tau \sim \pi^h(\cdot|o_1^h)} \mathbb{E}_{o_{1:T}^l \sim \pi^l(\cdot|\tau)} [\sum_{t=1}^{T} r_\tau(o_t^l)]$ where $\pi^l(\cdot|\tau)$ represents the distribution of full trajectories of length T under π^l and p_τ^{init}. In this section, we describe how we use the plan τ to further guide the learning and exploration of π^l through the reward function, policy parameterization, and environment termination conditions (right half of Fig. 2).

Dense Keypoint Rewards. Standard RL based approaches for dexterous manipulation often require complex, hand-crafted reward functions. However, provided a high-level keypoint plan τ dictating how all objects should move and interact, we can simply reward the agent for following the plan via keypoint distances. Though similar ideas have been used for tracking reference demonstrations [7] with ground-truth object poses, we instead track keypoints, which do not require full observability. Our final reward function is given by

$$r_\tau(o_t) = \underbrace{\exp\left(\frac{-\beta}{k}\sum_{i=1}^{k}\|\hat{\mathbf{x}}_t^{(i)} - \tilde{\mathbf{x}}_t^{(i)}\|_2\right)}_{\text{Keypoint Tracking}} + \underbrace{\exp\left(-1/(N_{\text{contact}}(o_t) + \epsilon)\right)}_{\text{Maintaining Contact}}, \quad (3)$$

where the first term is a function of the mean Euclidean distance between the planned and observed keypoint positions, and the second term $N_{\text{contact}}(o_t)$ represents the number of finger tips in contact with the environment. This reward formulation is significantly simpler than those used in previous RL approaches that lack trajectory supervision [33, 54] and can be applied to any task sufficiently captured by keypoint trajectories.

Residual Policy. To guide the agent towards the objective specified by the high-level plan τ, we employ "residual" RL [21, 13] in the absolute pose action space $\mathcal{A}$. Specifically, the learned low-level policy π_θ^l predicts offsets $\Delta\mathbf{w}$ to the wrist plan $\tilde{\mathbf{w}}_t$ instead of absolute actions $\mathbf{w}^{\text{targ}}$. Mathematically, this can be written as follows:

$$a_t = (\tilde{\mathbf{w}}_t + \Delta\mathbf{w}, \mathbf{q}_t^{\text{targ}}), \text{ where } (\Delta\mathbf{w}, \mathbf{q}_t^{\text{targ}}) \sim \pi^l(\cdot|o_t). \quad (4)$$

This guarantees that the learned policy follows the plan's wrist trajectory $\tilde{\mathbf{w}}_{1:T}$ by default.

Termination Conditions. To improve learning efficiency, we terminate episodes early if the tracking error, $\frac{1}{k}\sum_{i=1}^{k}\|\hat{\mathbf{x}}_t^{(i)} - \tilde{\mathbf{x}}_t^{(i)}\|_2$, exceeds a threshold δ. This early stopping criterion serves as a strong supervisory signal, encouraging the policy to remain close to the intended trajectory. To further guide learning, we introduce a curriculum: the initial threshold δ_{init} is linearly annealed to $\delta_{\text{init}}/2$ over the course of training. This facilitates broad exploration in the early stages while promoting precise trajectory tracking later on. We select task-specific values for δ_{init}, provided in Appendix D.

III. Experiments

We conduct a comprehensive suite of experiments to assess the effectiveness, generality, and robustness of our method across a diverse range of dexterous manipulation tasks. Our evaluation is structured around four core questions: 1) Are VLM scaffolds effective for a broad range of dexterous tasks? 2) How much can iterative refinement improve performance? 3) What causes VLM scaffolds to fail? 4) Can our method successfully learn policies that transfer to the real world?

A. Experimental Setup

Task Suite We construct an evaluation suite using the ManiSkill simulator [56, 40] and Allegro Hand model designed to evaluate four core dexterous manipulation capabilities for which motion planning is difficult: i) semantic understanding, ii) unstructured motion, iii) articulated object manipulation, and iv) precise manipulation. Each of the eight tasks, two per category, is depicted in Fig. 6. Instead of reward functions, each task is specified by a language instruction L. For example, the instruction for the "Move Apple" task is "Move the apple to the cutting board". The high-level VLM π^h is additionally guided by a prompt to detect specified keypoints. Further details can be found in Appendix F. Crucially, the capabilities evaluated by our task set are difficult to design reward functions for (articulated object manipulation or requiring complex and unstructured motion) or are challenging to specify using classical motion planning (requiring semantic knowledge or precision).

Methods Given the novelty of our problem setting, there are few applicable baselines which are also language-conditioned,

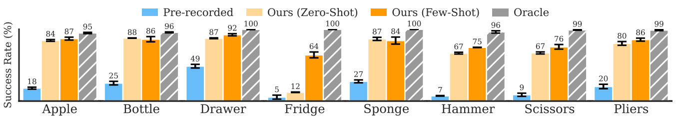

Fig. 3: Results on the simulation task suite. Success rate (in %) is averaged across 3 seeds and uncertainty is given by the standard error. Our method performs nearly as well as the oracle with perfectly scripted plans.

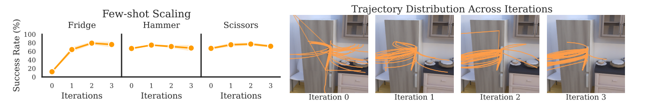

Fig. 4: (Left) The performance of our method as we iteratively refine the high-level policy π^h by providing successful plans τ in-context. (Right) The projected 3D plans on the evaluation set for each iteration.

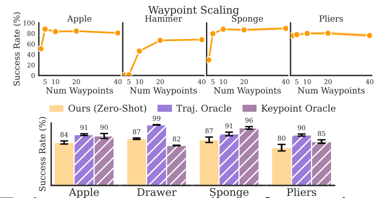

Fig. 5: (Left) Task success vs. number of waypoints in VLM plans. Most tasks saturate by 10 waypoints; only the hammer task benefits from denser trajectories. (Right) Ablation of VLM components. Replacing keypoints or trajectories with oracles highlights their relative impact across tasks.

demonstration-free, and do not require ground-truth state estimation. Thus, we mainly focus our experiments on comparison with a variety of oracles and ablations:

- **Oracle Keypoints and Trajectories:** This baseline uses fixed, manually defined keypoints and hard-coded trajectories for each task, representing an upper bound on performance with perfect semantic understanding and keypoint detection.
- **Reduced Waypoints:** We artificially constrain the VLM to produce shorter waypoint sequences, e.g., length three instead of $n = 20$, reducing the complexity of motion that can be expressed via the keypoints and wrist.
- **Pre-recorded Trajectories:** This method reuses pre-recorded trajectories from the training set at test-time, eliminating adaptability to new scenarios.

We evaluate two versions of our system: a zero-shot variant, where the vision-language model (VLM) receives no example plans, and a few-shot variant, where it is provided with three examples of successful plans τ in-context (Section II-B).

Architectures. We use Gemini 2.5 Flash Thinking [57] as the high-level policy with a thinking budget of 1000 tokens for plan generation. The low-level policy π^l is implemented as a 3-layer MLP with hidden dimensions of size 512 and ELU activations [8]. We sample 100 initial states and corresponding plans τ for training π^l with PPO [52].

Evaluation. For evaluation, we construct task-specific binary success metrics (e.g., object reaches target position, door opens to a minimum angle) to measure performance. All policy evaluations are conducted across 100 initial states with novel object configurations and hand poses. We run 20 trials for each configuration for a total of 2000 evaluation episodes and average results across three seeds.

B. How well do VLM scaffolds perform?

Simulation Results. Fig. 3 shows the success rates for the different simulation tasks. Our method with few-shot adaptation achieves consistently high success rates, often approaching the performance of the oracle with perfect scripted plans, indicating that modern VLMs are capable planners for scaffolding dexterous policies.

Iterative Refinement. We provide the VLM with successful trajectories from the training set as in-context examples to improve the proposed waypoints. We iterate this process up to three times in Fig. 4.

C. What Causes VLM Scaffolds To Fail?

Failure Modes. To comprehensively evaluate the failure modes of our pipeline across all tasks, we present a Sankey diagram in Fig. 8, categorizing errors into three primary sources: (i) incorrect keypoint detection, where keypoints do not lie on target objects, indicating deficiencies in VLM keypoint detection; (ii) incomplete trajectory tracking by the low-level policy, suggesting either inaccuracies in the low-level policy or unsuitable trajectories; and (iii) tracked trajectories that nonetheless fail to achieve success, revealing shortcomings in VLM trajectory generation.

Number of Waypoints. In Fig. 5 (Left), we evaluate the performance of our method using 3, 5, 10, 20, and 40 waypoints for plan generation. The results show that planning fidelity is typically not a large source of error.

VLM Components. To ablate the impact of using a VLM for keypoint detection and plan generation, we replace each component with an oracle in Fig. 5 (Right). For the Keypoint oracle, we use hand-specified keypoints for generating τ. For the Traj. oracle, we use VLM keypoints but script plans for τ.

D. Real-World Results

To evaluate sim-to-real transfer, we deploy our system on a real robot using the same inference pipeline as in simulation. From a single real-world RGB-D image and a natural-language command, the vision-language planner generates wrist and keypoint trajectories. The low-level policy is trained entirely in simulation using a digital twin of the real-world environment.

We perform initial experiments on two tasks with a Kuka robot and an Allegro hand: **Place Bottle onto Plate** and **Slide Box to Bottle**, which demonstrate semantic placement and non-prehensile manipulation (Fig. 7). We perform 20 policy rollouts per task, and achieve 90% success rate on the **Place Bottle onto Plate** task and 85% success rate on the **Slide Box to Bottle** task. These results suggest that our modular trajectory-based approach not only generalizes well within simulation but also scales to real-world deployment, reinforcing the practical viability of the proposed system. See Appendix E for details on the hardware experiments.

References

[1] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In Karen Liu, Dana

Kulic, and Jeff Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 403–415. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/agarwal23a.html.

[2] Shan An, Ziyu Meng, Chao Tang, Yuning Zhou, Tengyu Liu, Fangqiang Ding, Shufang Zhang, Yao Mu, Ran Song, Wei Zhang, Zeng-Guang Hou, and Hong Zhang. Dexterous manipulation through imitation learning: A survey, 2025. URL https://arxiv.org/abs/2504.03515.

[3] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39 (1):3–20, 2020.

[4] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems (RSS), Virtual Event/Corvalis, July*, pages 12–16, 2020.

[5] 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 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. In *arXiv preprint arXiv:2212.06817*, 2022.

[6] Yuanpei Chen, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuan Jiang, Zongqing Lu, Stephen McAleer, Hao Dong, Song-Chun Zhu, and Yaodong Yang. Towards human-level bimanual dexterous manipulation with reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 5150–5163. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/217a2a387f52c30755c37b0a73430291-Paper-Datasets_and_Benchmarks.pdf.

[7] Yuanpei Chen, Chen Wang, Yaodong Yang, and Karen Liu. Object-centric dexterous manipulation from human motion data. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=KAzku0Uyh1.

[8] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). arxiv 2015. *arXiv preprint arXiv:1511.07289*, 10, 2020.

[9] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[10] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.

[11] Zicong Fan, Omid Taheri, Dimitrios Tzionas, Muhammed Kocabas, Manuel Kaufmann, Michael J. Black, and Otmar Hilliges. ARCTIC: A dataset for dexterous bimanual hand-object manipulation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[12] Kuan Fang, Fangchen Liu, Pieter Abbeel, and Sergey Levine. Moka: Open-world robotic manipulation through mark-based visual prompting. *Robotics: Science and Systems (RSS)*, 2024.

[13] Siddhant Haldar, Jyothish Pari, Anant Rai, and Lerrel Pinto. Teach a robot to fish: Versatile imitation from one minute of demonstrations. In *Robotics: Science and Systems*, 2023.

[14] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan Ratliff, and Dieter Fox. Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9164–9170. IEEE, 2020.

[15] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[16] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[17] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation Bootstrapped Reinforcement Learning. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.056.

[18] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Conference on Robot Learning*, pages 540–562. PMLR, 2023.

[19] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[20] Aadhithya Iyer, Zhuoran Peng, Yinlong Dai, Irmak Guzey, Siddhant Haldar, Soumith Chintala, and Lerrel Pinto. OPEN TEACH: A versatile teleoperation system for robotic manipulation. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=cvAIaS6V2I.

[21] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[22] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. In *European Conference on Computer Vision*, pages 18–35. Springer, 2024.

[23] Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

[24] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: rapid motor adaptation for legged robots. *CoRR*, abs/2107.04034, 2021. URL https://arxiv.org/abs/2107.04034.

[25] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=10uNUgI5Kl.

[26] Teyun Kwon, Norman Di Palo, and Edward Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, 2024. doi: 10.1109/LRA.2024.3410155.

[27] Marion Lepert, Jiaying Fang, and Jeannette Bohg. Phantom: Training robots without robots using only human videos, 2025. URL https://arxiv.org/abs/2503.00779.

[28] Kailin Li, Puhao Li, Tengyu Liu, Yuyang Li, and Siyuan Huang. Maniptrans: Efficient dexterous bimanual manipulation transfer via residual learning. *arXiv preprint arXiv:2503.21860*, 2025.

[29] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[30] Toru Lin, Zhao-Heng Yin, Haozhi Qi, Pieter Abbeel, and Jitendra Malik. Twisting lids off with two hands. *arXiv preprint arXiv:2403.02338*, 2024.

[31] Haichao Liu, Sikai Guo, Pengfei Mai, Jiahang Cao, Haoang Li, and Jun Ma. Robodexvlm: Visual language model-enabled task planning and motion control for dexterous robot manipulation. *arXiv preprint arXiv:2503.01616*, 2025.

[32] Xueyi Liu, Jianibieke Adalibieke, Qianwei Han, Yuzhe Qin, and Li Yi. Dextrack: Towards generalizable neural tracking control for dexterous manipulation from human references. *arXiv preprint arXiv:2502.09614*, 2025.

[33] Tyler Ga Wei Lum, Martin Matak, Viktor Makoviychuk, Ankur Handa, Arthur Allshire, Tucker Hermans, Nathan D. Ratliff, and Karl Van Wyk. DextrAH-g: Pixels-to-action dexterous arm-hand grasping with geometric fabrics. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=S2Jwb0i7HN.

[34] Tyler Ga Wei Lum, Olivia Y. Lee, C. Karen Liu, and Jeannette Bohg. Crossing the human-robot embodiment gap with sim-to-real rl using one human demonstration, 2025. URL https://arxiv.org/abs/2504.12609.

[35] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[36] Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, et al. Vision language models are in-context value learners. In *The Thirteenth International Conference on Learning Representations*, 2024.

[37] Parsa Mahmoudieh, Deepak Pathak, and Trevor Darrell. Zero-shot reward specification via grounded natural language. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14743–14752. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/mahmoudieh22a.html.

[38] Priyanka Mandikal and Kristen Grauman. Dexvip: Learning dexterous grasping with human hand pose priors from video. In *Conference on Robot Learning (CoRL)*, 2021.

[39] Suvir Mirchandani, Fei Xia, Pete Florence, brian ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large language models as general pattern machines. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=RcZMI8MSyE.

[40] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=zQIvkXHS_U5.

[41] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299, 2018. doi:

10.1109/ICRA.2018.8463162.

[42] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[43] Open X-Embodiment Collaboration, Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, Antonin Raffin, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Brian Ichter, Cewu Lu, Charles Xu, Chelsea Finn, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Chuer Pan, Chuyuan Fu, Coline Devin, Danny Driess, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Federico Ceola, Fei Xia, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Giulio Schiavi, Hao Su, Hao-Shu Fang, Haochen Shi, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jaehyung Kim, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jiajun Wu, Jialin Wu, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jitendra Malik, Jonathan Tompson, Jonathan Yang, Joseph J. Lim, João Silvério, Junhyek Han, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Zhang, Keyvan Majd, Krishan Rana, Krishnan Srinivasan, Lawrence Yunliang Chen, Lerrel Pinto, Liam Tan, Lionel Ott, Lisa Lee, Masayoshi Tomizuka, Maximilian Du, Michael Ahn, Mingtong Zhang, Mingyu Ding, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Pannag R Sanketi, Paul Wohlhart, Peng Xu, Pierre Sermanet, Priya Sundaresan, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Sherry Moore, Shikhar Bahl, Shivin Dass, Shuran Song, Sichun Xu, Siddhant Haldar, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Sudeep Dasari, Suneel Belkhale, Takayuki Osa, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xuanlin Li, Yao Lu, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yueh hua Wu, Yujin Tang, Yuke Zhu, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zhuo Xu, and Zichen Jeff Cui. Open X-Embodiment: Robotic learning datasets and RT-X models, 2023.

[44] Shivansh Patel, Xinchen Yin, Wenlong Huang, Shubham Garg, Hooshang Nayyeri, Li Fei-Fei, Svetlana Lazebnik, and Yunzhu Li. A real-to-sim-to-real approach to robotic manipulation with vlm-generated iterative keypoint rewards. *arXiv preprint arXiv:2502.08643*, 2025.

[45] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *Acm transactions on graphics (tog)*, 36(4):1–13, 2017.

[46] Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-Hand Object Rotation via Rapid Motor Adaptation. In *Conference on Robot Learning (CoRL)*, 2022.

[47] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *IEEE Robotics and Automation Letters*, 7(4):10873–10881, 2022.

[48] Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. In *Robotics: Science and Systems*, 2023.

[49] Ilija Radosavovic, Xiaolong Wang, Lerrel Pinto, and Jitendra Malik. State-only imitation learning for dexterous manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7865–7871, 2021. doi: 10.1109/IROS51168.2021.9636557.

[50] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.049.

[51] Juntao Ren, Priya Sundaresan, Dorsa Sadigh, Sanjiban Choudhury, and Jeannette Bohg. Motion tracks: A unified representation for human-robot transfer in few-shot imitation learning, 2025.

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

[53] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[54] Ritvik Singh, Arthur Allshire, Ankur Handa, Nathan Ratliff, and Karl Van Wyk. Dextrah-rgb: Visuomotor policies to grasp anything with dexterous hands. *arXiv preprint arXiv:2412.01791*, 2024.

[55] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa

Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[56] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[57] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[58] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[59] David Venuto, Mohammad Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and Ankit Anand. Code as reward: Empowering reinforcement learning with VLMs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=6P88DMUDvH.

[60] Chen Wang, Haochen Shi, Weizhuo Wang, Ruohan Zhang, Li Fei-Fei, and C Karen Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. *arXiv preprint arXiv:2403.07788*, 2024.

[61] Jun Wang, Ying Yuan, Haichuan Che, Haozhi Qi, Yi Ma, Jitendra Malik, and Xiaolong Wang. Lessons from learning to spin "pens". In *CoRL*, 2024.

[62] Qianxu Wang, Congyue Deng, Tyler Ga Wei Lum, Yuanpei Chen, Yaodong Yang, Jeannette Bohg, Yixin Zhu, and Leonidas Guibas. Neural attention field: Emerging point relevance in 3d scenes for one-shot dexterous grasping. *arXiv preprint arXiv:2410.23039*, 2024.

[63] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. RL-VLM-f: Reinforcement learning from vision language foundation model feedback. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=YSoMmNWZZx.

[64] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17868–17879, 2024.

[65] Chuan Wen, Xingyu Lin, John Ian Reyes So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point Trajectory Modeling for Policy Learning. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.092.

[66] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

[67] Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. Flow as the cross-domain manipulation interface. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=cNI0ZkK1yC.

[68] Fan Yang, Dongsheng Luo, Wenrui Chen, Jiacheng Lin, Junjie Cai, Kailun Yang, Zhiyong Li, and Yaonan Wang. Multi-keypoint affordance representation for functional dexterous grasping, 2025. URL https://arxiv.org/abs/2502.20018.

[69] Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning, 2023.

[70] Lixin Yang, Kailin Li, Xinyu Zhan, Fei Wu, Anran Xu, Liu Liu, and Cewu Lu. OakInk: A large-scale knowledge repository for understanding hand-object interaction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[71] Weirui Ye, Yunsheng Zhang, Haoyang Weng, Xianfan Gu, Shengjie Wang, Tong Zhang, Mengchen Wang, Pieter Abbeel, and Yang Gao. Reinforcement learning with foundation priors: Let embodied agent efficiently learn on its own. In *8th Annual Conference on Robot Learning*.

[72] Zhao-Heng Yin, Changhao Wang, Luis Pineda, Krishna Bodduluri, Tingfan Wu, Pieter Abbeel, and Mustafa Mukadam. Geometric retargeting: A principled, ultrafast neural hand retargeting algorithm. *arXiv preprint arXiv:2503.07541*, 2025.

[73] Zhao-Heng Yin, Changhao Wang, Luis Pineda, Francois Hogan, Krishna Bodduluri, Akash Sharma, Patrick Lancaster, Ishita Prasad, Mrinal Kalakrishnan, Jitendra Malik, et al. Dexteritygen: Foundation controller for unprecedented dexterity. *arXiv preprint arXiv:2502.04307*, 2025.

[74] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, brian ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=SgTPdyehXMA.

APPENDIX

A. Related Work

Planning with Vision-Language Models. Recent advances have demonstrated the potential of VLMs to guide robotic planning through their powerful semantic and spatial reasoning capabilities. One family of approaches directly synthesizes policies by translating natural language instructions into executable code using low-level perception and control APIs [29, 53, 18, 66]. To extend this to dexterous manipulation, [31] integrates predefined skill libraries, at the expense of limiting generalization and behavioral diversity. Other efforts propose using VLMs to plan actions by generating spatial keypoint constraints [19] or directly producing waypoints [12, 44]. However, such methods operate in an open-loop fashion and lack the closed-loop feedback necessary for fine-grained, adaptive control in dexterous tasks. Aside from directly learning policies, several works use VLMs to code dense reward functions [35, 74, 59], but these approaches often require privileged access to environment internals and result in opaque and sometimes hard-to-interpret reward structures. Other approaches more directly leverage the vision capabilities of VLMs to act as success detectors [10, 25, 69], reward functions [37], or value functions [36, 71] for RL. Oftentimes, these quantities can be learned from VLM generated preferences [63, 26, 23]. However, all of these approaches are often too imprecise to produce the dense optimization signals required for dexterous manipulation and are less efficient than using the VLM to simply produce a plan.

Learned Dexterous Manipulation. Though early works demonstrated the feasibility of deploying in-hand manipulation policies trained in simulation on real robots [3, 15, 6, 30], they relied on carefully crafted reward functions for each task. Such approaches have proven most successful in locomotion [4, 24, 1], where rewards are more easily designed and terrain can be replicated, unlike object dynamics in manipulation. More recent efforts scale to full-arm dexterity and multi-object grasping [54, 33], while others incorporate human priors to improve sample efficiency [38]. Despite these advances, most approaches are still limited to only a set task, e.g. object grasping or rotation [46, 61], where manually, task specific rewards can be designed. However, this approach remains inherently unscalable to more complex and non-cyclic tasks.

Dexterous Manipulation by Tracking Motions. When framing dexterous manipulation as a tracking problem, dense rewards are easy to obtain via tracking error [4, 45]. Some systems leverage motion capture data to extract object and wrist trajectories from human demonstrations, which are then used to train tracking policies in simulation via residual RL [7, 28]. Other approaches improve robustness by iteratively adding successful rollouts to the training dataset [32]. Recent work also shows that a single demonstration can bootstrap effective policy learning [34]. However, all of these methods depend on human demonstrations, which are expensive to collect and difficult to scale.

B. Conclusion

We presented a new framework for dexterous robotic manipulation that combines VLMs with reinforcement learning to generate and execute semantically meaningful hand-object trajectories. By casting manipulation as a trajectory-tracking problem using VLM-generated keypoint plans, our method eliminates the need for human demonstrations or handcrafted reward functions, while enabling generalization across diverse objects, goals, and scene configurations.

Our experiments in both simulation and the real world show that this approach reliably solves a variety of complex manipulation tasks, including articulated objects, semantic reasoning, and fine finger control. The system exhibits strong generalization to novel keypoints and configurations, and transfers effectively to physical hardware without additional tuning or data collection.

Limitations and Future work. Our method currently assumes rigid-body object interaction, which simplifies simulation keypoint tracking. Extending to deformable objects would require improved simulators and the ability to track keypoints on non-rigid surfaces (e.g., point tracking models [22]). Additionally, high-level trajectory generation is not directly aware of the low-level controller's capabilities, limiting the system's adaptiveness; closing this loop by feeding execution feedback back into the VLM planner is a promising direction. Finally, trajectory generation with current reasoning VLMs takes 1-2 minutes, limiting responsiveness and motivating faster VLMs.

C. Hyperparameters

PPO The hyperparameters of our PPO training are detailed in Table I.

TABLE I: PPO Hyperparameters

Hyperparameter	Value
Normalize Advantage per Mini-Batch	True
Value Loss Coefficient	1.0
Clip Parameter	0.2
Use Clipped Value Loss	True
Desired KL	0.01
Entropy Coefficient	0.01
Discount Factor (Gamma)	0.99
GAE Lambda (Lam)	0.95
Max Gradient Norm	1.0
Learning Rate	0.0003
Number of Learning Epochs	5
Number of Mini-Batches	16
Schedule	Adaptive
Policy Class Name	ActorCritic
Activation Function	ELU
Actor Hidden Dimensions	[512, 512, 512]
Critic Hidden Dimensions	[512, 512, 512]
Initial Noise Std	1.0
Noise Std Type	Scalar
Number of Steps per Environment	24
Max Iterations	2000
Empirical Normalization	True
Number of Environments	2048

Simulation We use Maniskill3 [56] for our simulations. Our hyperparameters are listed in Table II. We situate our tasks in simulated scenes from the ReplicaCAD dataset [55]. Some of the objects in the tasks are from the RoboCasa project [42].

TABLE II: Simulation and Control Settings

Setting	Value
Action Exponential Average Gamma	0.9
Simulation Frequency	120 Hz
Control Frequency	60 Hz
Max Rigid Contact Count	$2048 \times 2048 \times 8$
Max Rigid Patch Count	$2048 \times 2048 \times 2$
Found Lost Pairs Capacity	2^{27}
Gravity	[0, 0, -9.81]
Bounce Threshold	2.0
Solver Position Iterations	8
Solver Velocity Iterations	0
Default Dynamic Friction	1.0
Default Static Friction	1.0
Restitution	0
Finger Static Friction	2.0
Dummy Joint Stiffness	2000
Dummy Joint Damping	100
Dummy Joint Force Limits	1000
Finger Joint Stiffness	10
Finger Joint Damping	0.3
Finger Joint Force Limit	10
Controller Type	PD Joint Targets

VLM

We detail our query settings in Table III.

TABLE III: VLM Configuration

Hyperparameter	Value
Image Size	800×800
Trajectory Query Thinking Budget	1000
Keypoint Query Temperature	0.5
Trajectory Query Code Execution	Enabled

D. Environment

Observation Space

Table IV details the components of our observation space. Importantly, our policy does not rely on privileged information such as contact forces during training, making the observation space more amenable to real-world deployment.

TABLE IV: Observation Space Configuration

Observation Type	Dimension
Joint Position (Dummy Joints + Fingers)	22
Joint Velocity (Dummy Joints + Fingers)	22
Exponential Average Action	22
Finger Poses	$4 \times 7 = 28$
Initial Keypoint Positions	$3 \times k$
Current Keypoint Positions	$3 \times k$
Planned Future Keypoint Positions	$3 \times 15 \times k$
Current Wrist Pose	6
Planned Future Wrist pose	$6 \times 15 = 30$
Total	$130 + 6 \times k + 3 \times 15 \times k$

Action Space

We use a residual action space on the wrist pose, and directly control the fingers. We normalized the action space to the range [-1, 1]. A ”zero” action corresponds to following the reference trajectory precisely with an entirely open hand.

We only control the fingers individually for the scissors and pliers tasks since we did not see any benefit for the other tasks. For the scissors and pliers task we have one action for every finger (instead of every joint). This makes hand action space four dimensional for the allegro hand. For all other tasks we control all fingers with one action, only opening or closing the entire hand.

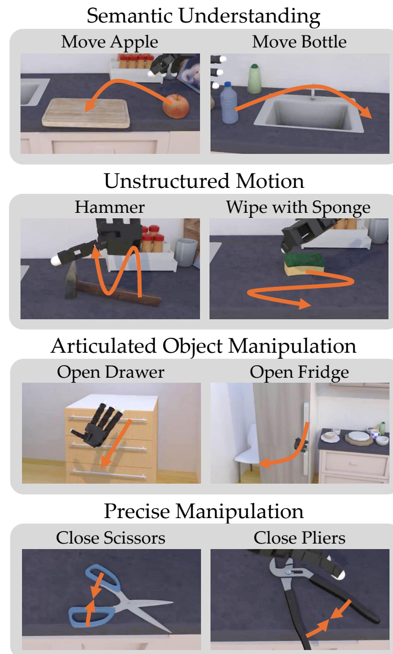

Fig. 6: A depiction of the eight tasks used for evaluation. Each task belongs to one of four overarching categories.

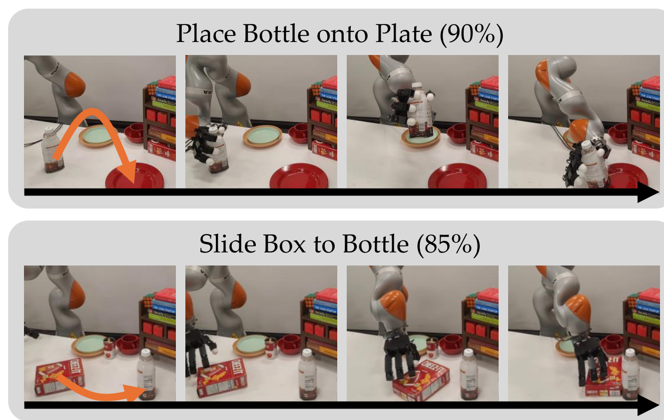

Fig. 7: Real-world rollouts of **Place Bottle onto Plate** and **Slide Box to Bottle**.

Termination Thresholds

We detail our initial termination thresholds per task in Table V. The initial thresholds are linearly reduced to half of their initial value over the course of training.

E. Hardware Experiment Details

Inference-Time Pipeline Details

TABLE V: Initial Termination Thresholds for Manipulation Tasks

Object	Threshold (cm)
Apple	10
Bottle	10
Hammer	8
Drawer	15
Sponge	8
Plier	5
Scissors	3
Fridge	20

At inference-time, we run our policy in the real-world as follows:

1) Capture an RGB-D image of the scene.
2) Query the VLM for 2D keypoints, given the RGB image and natural language command.
3) Backproject the 2D keypoints into 3D keypoints using the depth camera and camera intrinsics.
4) Transform the 3D keypoints from camera frame to world frame using camera extrinsics.
5) Query the VLM for a wrist pose trajectory and keypoint trajectories, given the initial wrist pose and 3D keypoints.
6) Run FoundationPose [64] to track the objects, which allows us to track their associated keypoints (we assume the keypoint does not move relative to the object frame)
7) Run the low-level policy, given the base wrist pose trajectory and tracked keypoints.

Mapping Wrist Actions to Arm Joints

In simulation-only experiments, we control a *floating* (non-physical) hand whose wrist pose can be commanded directly. To train a policy in simulation that can be executed on a real robot, the wrist is attached to a 7-DoF KUKA LBR iiwa 14 arm, so the residual wrist-pose action produced by the policy must be converted into incremental arm joint commands. We perform this conversion with damped–least–squares inverse kinematics (DLS-IK).

Let $J \in \mathbb{R}^{6\times N_{\text{arm}}}$ be the analytical Jacobian of the arm ($N_{\text{arm}} = 7$ in our setup), evaluated at the current joint configuration $\boldsymbol{\theta} \in \mathbb{R}^{N_{\text{arm}}}$, and let $\mathbf{e} \in \mathbb{R}^6$ be the 6-D spatial error twist (concatenated position and orientation error) between the current wrist pose and the target wrist pose. The arm joint update $\Delta\boldsymbol{\theta} \in \mathbb{R}^{N_{\text{arm}}}$ is computed as:

$$\Delta\boldsymbol{\theta} = J^\top \left(JJ^\top + \lambda^2 I_6\right)^{-1} \mathbf{e}, \tag{5}$$

where $\lambda = 0.5$ is a constant damping factor. Equation (5) implements the damped pseudoinverse $J_\lambda^\dagger = J^\top(JJ^\top + \lambda^2 I_6)^{-1}$, yielding the minimum-norm solution to $J\Delta\boldsymbol{\theta} = \mathbf{e}$ while regularising the update near kinematic singularities. Lastly, we compute a joint position target $\boldsymbol{\theta}^{\text{target}} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$, clamp this to stay within the joint limits, and then send this as the target to a low-level joint-position PD controller running at 200 Hz.

By default, the target wrist pose is specified by the wrist pose trajectory generated by the VLM. The policy outputs a residual wrist pose action that modifies this target, allowing fine-grained corrections. The resulting target is then used to compute the spatial error $\mathbf{e}$. By construction, if the residual wrist pose action is $\mathbf{0}$, the error $\mathbf{e}$ corresponds exactly to the difference between the current wrist pose and the original VLM-generated trajectory, so the arm will simply follow the given wrist pose trajectory.

Tasks

We evaluate our system on two real-world manipulation tasks:

- **Slide Box to Bottle**: The goal is to push the box to the bottle. The box starts from a face-down orientation approximately 35cm away from the bottle. A trial is considered successful if the box makes contact with the bottle.
- **Place Bottle onto Plate**: The goal is to grasp the bottle and place it onto a plate. The bottle starts from an upright orientation approximately 42cm away from the plate. The task is considered successful if the bottle is lifted and makes contact with the top surface of the plate.

For each task, we run 20 trials across 4 VLM-generated trajectories. The procedure is as follows: We first initialize the objects in random positions with the same range as used in simulation training. Next, we capture an RGB-D image of the scene, and then query the VLM to generate a wrist trajectory and keypoint trajectories based on the image and a natural language instruction. Each generated trajectory is tested in 5 repeated trials, resetting the objects to similar initial poses before each attempt. This process is repeated 4 times with new randomized object positions and new trajectory queries, resulting in 20 total trials per task.

Domain Randomization

We improve the policy's ability to transfer to the real world in a zero-shot setting through domain randomization. This enables robustness to physical parameters that are unknown, noisy, or inaccurately modeled in the real environment. Specifically, we apply the following randomizations during training:

- **Joint stiffness and damping** are multiplied from their default values by a factor sampled from a uniform distribution: $\mathcal{U}(0.3,\ 3.0)$. These parameters are sampled once at the start of training (independently for each parallel environment) and remain fixed throughout training.
- **Observation noise** is added to each of the robot proprioception observations, sampled independently from a normal distribution: $\mathcal{N}(0,\ 0.05^2)$. This is uncorrelated noise that is sampled at every control timestep.
- **Action noise** is added to exponential average action, sampled from: $\mathcal{N}(0,\ 0.05^2)$. This is uncorrelated noise that is sampled at every control timestep.

Additional Adjustments

- The observation space is nearly identical to that described in Table IV, except that the dummy joints used to control the floating-hand wrist pose are replaced with the arm's actual joints (for joint positions, velocities, and exponentially averaged actions).
- We increase the exponential smoothing factor for the action average to $\gamma = 0.98$ to produce smoother motions and reduce jitter in the executed actions.

- We adjusted the trajectories to be twice as long for real-world experiments to effectively slow the robot motion down. We found that higher-speed motions typically resulted in less reliable policies, as this likely worsened the sim-to-real gap.
- To prevent significant collisions between the hand and the table, we clamp the z-coordinate of the target wrist pose to remain above the table height.

Digital Twin Construction

Our digital twin simulation environment consisted of a robot, table, and two objects per task. The robot URDF and physics parameters were acquired by standard open-source repositories. We measured the dimensions of the table and its position relative to the robot with a measuring tape, which took about 5 minutes. The objects were scanned using an off-the-shelf 3D LiDAR scanning app called Kiri Engine, which took about 3 minutes per object.

Additional Qualitative Analysis

- We found that VLM keypoint detection worked significantly better on real world images, as they are more likely to be in-distribution than simulation images.
- As the low-level policy operates in a closed-loop fashion, we find it to be robust to dynamics differences between simulation and reality.
- The low-level RL policy appeared to optimize the task objective (moving the object keypoint along the generated keypoint trajectory) very well. For example, on the **Slide Box to Bottle** task, when the predicted box keypoint was on the bottom side of the box, the policy would not only push the box to the bottle, but rotate the box so that the bottom side of the box would be as close as possible to the bottle. On the **Place Bottle onto Plate** task, when the predicted bottle keypoint was on the upper half of the bottle, the policy would often place the bottle on its side so that the keypoint would be as close to the plate as possible.
- The most common failure mode came from the keypoint tracking errors. While the initial predicted keypoints were accurate, our pose tracker was only reliable when the object was completely unoccluded. The pose predictions got worse when the object was occluded and occasionally got very bad when highly occluded, which degraded policy performance.
- We performed preliminary experiments testing our policy on unseen objects with different but similar geometry (e.g., replacing the bottle with a mustard bottle or tall cup, replacing the plate with a different sized plate). The policy still worked reasonably well on these unseen objects due to the VLM's common-sense understanding to select good keypoints and the RL policy's state-based observations.

F. Tasks

We provide brief descriptions of the eight simulated tasks we evaluated:

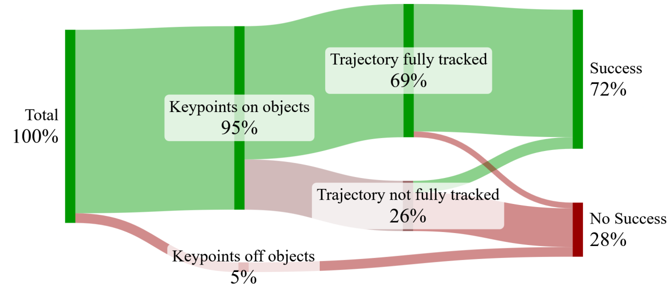

Fig. 8: Error decomposition across failure cases. Most errors stem from incomplete trajectory tracking, followed by keypoint detection issues.

- **Move Apple:** An apple and a cutting board are placed on a kitchen counter. The keypoints are the apple and the cutting board. The agent's objective is to pick up the apple and place it on top of the cutting board.
- **Move Bottle:** A bottle is positioned on a kitchen counter next to a sink. The keypoints are the bottle and a target point on the counter across the sink. The goal is for the agent to pick up the bottle and move it to the other side of the sink.
- **Open Drawer:** A closed cupboard with multiple drawers is located in a living room. The handle of the top drawer serves as the sole keypoint. The objective is to open this drawer by at least 20 cm.
- **Open Fridge:** A closed refrigerator is situated in a kitchen. The handle of the fridge is the only keypoint. The agent's task is to open the fridge door by at least 60 degrees.
- **Hammer:** A hammer rests on a kitchen counter, with the head and handle defined as keypoints. The goal is for the agent to pick up the hammer and perform a hammering motion with at least three swings. A swing is defined as an upward and downward movement of at least 5 cm.
- **Wipe with Sponge:** A sponge is located on a kitchen counter, acting as the sole keypoint. The task is to perform a wiping motion on the counter, with success defined as moving the sponge at least 30 cm on the counter.
- **Close Scissors:** An open pair of scissors is situated on a kitchen counter, with the handles serving as keypoints. The goal is to close the scissors until the blades form an angle of less than 5 degrees.
- **Close Pliers:** An open pair of pliers is positioned on a kitchen counter, with the handles defined as keypoints. The objective is to close the pliers until the handles form an angle of less than 5 degrees.

G. Compute Resources

Our training is performed on NVIDIA GPUs, ranging from A5000s to L40s. Depending on the specific task and hardware configuration, training durations vary between 1.5 and 6 hours. For real-world inference, we utilize two RTX 4090 GPUs.

H. Prompt Examples

Move Apple
Keypoint Prompt

```
Point to the apple and the cutting board in
the image.
The answer should follow the json format: [{"
name": "apple", "point": [...]}, {"name": "
cutting board", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to pick up
an apple and put it on a cutting board.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the apple.
Then grasp the apple and lift it up.
Finally move the apple on the cutting board
and put it down.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the apple is [0.00,
0.00, 0.00].
The initial position of the cutting board is
[-0.01, -0.38, -0.05].
The initial position of the hand is [-0.07,
-0.09, 0.26].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"apple": {"x": float, "y": float, "z": float},
"cutting board": {"x": float, "y": float, "z":
 float},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Move Bottle

Keypoint Prompt

```
Point to the water bottle on the kitchen
counter, and pinpoint a point on the kitchen
counter to the right of the kitchen sink in
the image.
The answer should follow the json format: [{"
name": "bottle", "point": [...]}, {"name": "
point", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to move a
bottle to the target position called "point"
on the kitchen counter.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the bottle.
Then grasp the bottle and lift it up.
Finally move the bottle to the target position
 called "point" and put it down.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the bottle is [0.00,
0.00, 0.00].
The initial position of the point is [-0.22,
0.80, -0.13].
The initial position of the hand is [0.25,
-0.08, 0.20].
Use the following json format for the
trajectory:
[\{
"waypoint_num": 0,
"bottle": {"x": float, "y": float, "z": float
},
"point": {"x": float, "y": float, "z": float},
"hand": {"x": float, "y": float, "z": float}
\} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Open Drawer

Keypoint Prompt

```
Point to the handle of the top cabinet drawer
in the image.
The answer should follow the json format: [{"
name": "handle", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to pull open
 a cabinet drawer.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the handle
of the drawer.
Then grasp the handle.
Finally pull the drawer open by at least 30cm.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the handle is [0.00,
0.00, 0.00].
The initial position of the hand is [0.32,
-0.05, 0.12].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"handle": {"x": float, "y": float, "z": float
},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Open Fridge

Keypoint Prompt

```
Point to the top handle of the refrigerator
door in the image.
The answer should follow the json format: [{"
name": "handle", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to open a
refrigerator door.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
The refrigerator faces in x direction.
The y axis points to the right, and the z axis
 points up.
First figure out how large the door is.
Then describe how the x and y coordinates of
the handle change as the door is opened.
Now move the robot hand towards the handle.
Then grasp the handle.
Finally fully open the door.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the handle is [0.00,
0.00, 0.00].
The initial position of the hand is [0.50,
0.00, -0.22].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"handle": {"x": float, "y": float, "z": float
},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Hammer

Keypoint Prompt

```
Point to the brown handle and the metal head
of the hammer in the image.
The answer should follow the json format: [{"
name": "handle", "point": [...]}, {"name": "
head", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to make a
hammering motion.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the handle.
Then grasp the handle.
Finally hit on the kitchen counter 3 times.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the handle is [0.00,
0.00, 0.00].
The initial position of the head is [-0.02,
-0.15, 0.03].
The initial position of the hand is [0.01,
0.06, 0.27].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"handle": {"x": float, "y": float, "z": float
},
"head": {"x": float, "y": float, "z": float},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Wipe with Sponge

Keypoint Prompt

```
Point to the green yellow sponge on the
kitchen counter in the image.
The answer should follow the json format: [{"
name": "sponge", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to wipe a
kitchen counter with a sponge.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the sponge.
Then grasp the sponge.
Finally wipe the kitchen counter with the
sponge.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the sponge is [0.00,
0.00, 0.00].
The initial position of the hand is [0.26,
0.03, 0.29].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"sponge": {"x": float, "y": float, "z": float
},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Close Scissors

Keypoint Prompt

```
Point to the two loops of the scissors in the
image.
The answer should follow the json format: [{"
name": "loop 1", "point": [...]}, {"name": "
loop 2", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to close a
pair of scissors.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the scissors
.
```

```
Then grasp the two loops and entirely close
the scissors.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the loop 1 is [0.00,
0.00, 0.00].
The initial position of the loop 2 is [-0.07,
0.07, 0.01].
The initial position of the hand is [0.03,
-0.06, 0.33].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"loop 1": {"x": float, "y": float, "z": float
},
"loop 2": {"x": float, "y": float, "z": float
},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Close Pliers

Keypoint Prompt

```
Point to the left and right handles of the
plier in the image.
The answer should follow the json format: [{"
name": "handle left", "point": [...]}, {"name
": "handle right", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to close a
plier.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the plier.
Then grasp the left and right handles and
entirely close the plier.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the handle left is
[0.00, 0.00, 0.00].
The initial position of the handle right is
[-0.05, 0.16, 0.00].
The initial position of the hand is [0.01,
-0.08, 0.31].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"handle left": {"x": float, "y": float, "z":
float},
"handle right": {"x": float, "y": float, "z":
float},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Place Bottle onto Plate

Keypoint Prompt

```
Point to the middle of the bottle and the
plate on the table in the image.
The answer should follow the json format: [{"
name": "bottle", "point": [...]}, {"name": "
plate", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to move a
bottle onto a plate.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the bottle.
Then grasp the bottle and lift it up.
Then place the bottle on to the plate.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the bottle is [0.00,
0.00, 0.00].
The initial position of the plate is [0.30,
0.38, -0.15].
The initial position of the hand is [-0.15,
-0.29, 0.09].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"bottle": {"x": float, "y": float, "z": float
},
"plate": {"x": float, "y": float, "z": float},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```

Slide Box to Bottle

Keypoint Prompt

```
Point to the box and the bottle on the table
in the image.
The answer should follow the json format: [{"
name": "box", "point": [...]}, {"name": "
bottle", "point": [...]}]
The points are in [y, x] format normalized to
0-1000.
```

Trajectory Prompt

```
Your are controlling a robot hand to slide the
 box over the table to the bottle.
The coordinates are measured in meters.
The x axis is forward, the y axis is left and
the z axis is up.
First move the robot hand towards the box.
Then slide the box over the table to the
bottle.
Describe a very realistic trajectory of
exactly 20 waypoints.
Use code to generate the output.
The initial position of the box is [0.00,
0.00, 0.00].
```

```
The initial position of the bottle is [0.17,
0.23, 0.08].
The initial position of the hand is [-0.22,
-0.29, 0.19].
Use the following json format for the
trajectory:
[{
"waypoint_num": 0,
"box": {"x": float, "y": float, "z": float},
"bottle": {"x": float, "y": float, "z": float
},
"hand": {"x": float, "y": float, "z": float}
} ...]
**Only** print the json output. Do **not**
print anything else with the code.
```