3D Diffuser Actor: Policy Diffusion with 3D Scene Representations

Tsung-Wei Ke†, Nikolaos Gkanatsios†, Katerina Fragkiadaki

3d-diffuser-actor.github.io

***Abstract*— We marry diffusion policies and 3D scene representations for robot manipulation. Diffusion policies learn the action distribution conditioned on the robot and environment state using conditional diffusion models. They have recently shown to outperform both deterministic and alternative state-conditioned action distribution learning methods. 3D robot policies use 3D scene feature representations aggregated from a single or multiple camera views using sensed depth. They have shown to generalize better than their 2D counterparts across camera viewpoints. We unify these two lines of work and present 3D Diffuser Actor, a neural policy architecture that, given a language instruction, builds a 3D representation of the visual scene and conditions on it to iteratively denoise 3D rotations and translations for the robot's end-effector. At each denoising iteration, our model represents end-effector pose estimates as 3D scene tokens and predicts the 3D translation and rotation error for each of them, by featurizing them using 3D relative attention to other 3D visual and language tokens. 3D Diffuser Actor sets a new state-of-the-art on RLBench with an absolute performance gain of 18.1% over the current SOTA on a multi-view setup and an absolute gain of 13.1% on a single-view setup. On the CALVIN benchmark, it outperforms the current SOTA in the setting of zero-shot unseen scene generalization by being able to successfully run 0.2 more tasks, a 7% relative increase. It also works in the real world from a handful of demonstrations. We ablate our model's architectural design choices, such as 3D scene featurization and 3D relative attentions, and show they all help generalization. Our results suggest that 3D scene representations and powerful generative modeling are keys to efficient robot learning from demonstrations.**

I. Introduction

Many robot manipulation tasks are inherently multimodal: at any point during task execution, there may be multiple actions which yield task-optimal behavior. A natural choice is then to treat policy learning as a distribution learning problem conditioned on the current robot state [1], [2], [3], [4]. Recent works use diffusion models for learning action distributions from demonstrations [5], [6], [7] and outperform deterministic or other alternatives [8], [9], [4], [10]. They exhibit better action distribution coverage and higher fidelity (less mode hallucination) than alternative formulations [5]. They have so far been used with low-dimensional engineered state representations [5] or 2D image encodings [6].

Lifting features from perspective views to a bird's eye view (BEV) or 3D robot workspace map has shown strong results in robot learning [13], [14], [15], [16]. Our conjecture is that this improved performance comes from the fact that the 3D visual scene content and the robot's end-effector poses live in a common 3D space. BEV or 3D policy formulations e.g., Transporter Networks [17], [18], C2F-ARM [14], PerAct [13], Act3D [16] and RVT [15], discretize the robot's workspace for localizing the robot's end-effector. Such 3D policies have not been combined yet with diffusion objectives.

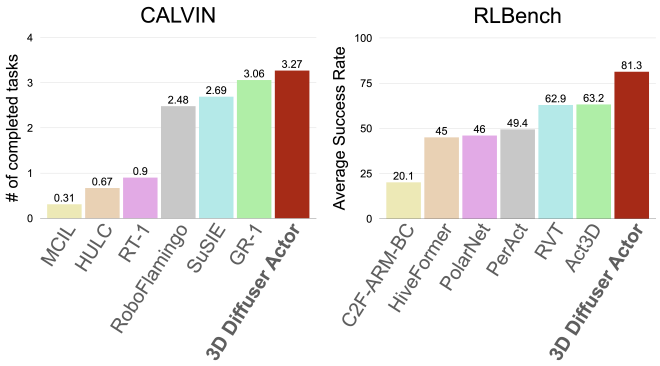


Fig. 1: **3D Diffuser Actor sets a new state-of-the-art on RLBench [11] on a multi-view setup (PerAct) and on CALVIN [12] on a zero-shot long-horizon setup (ABC-D).**

In this paper, we propose 3D Diffuser Actor, a model that marries diffusion policies for handling action multimodality and 3D scene encodings for effective spatial reasoning. 3D Diffuser Actor is a denoising neural network that takes as input a 3D scene, the current estimate of the end-effector's future trajectory (3D location and orientation), as well as the diffusion iteration index, and predicts the error in 3D translation and rotation. Our model achieves translation equivariance in prediction by representing the current estimate of the robot's end-effector trajectory as 3D scene tokens and featurizing them jointly with the visual tokens using relative-position 3D attentions [19], [20], as shown in Figure 2.

We test 3D Diffuser Actor in learning from demonstrations on the simulation benchmarks of RLBench [11] and CALVIN [12], as well as in the real world. Our model sets a new state-of-the-art on RLBench, outperforming existing 3D policies and 2D diffusion policies with an 18.1% absolute gain. On CALVIN, it outperforms the current SOTA in the setting of zero-shot unseen scene generalization by a 7% relative gain (Figure 1). We additionally show that 3D Diffuser Actor outperforms all existing policy formulations that either do not use 3D scene representations or do not use action diffusion. We further show 3D Diffuser Actor can learn multi-task manipulation in the real world from few demonstrations. Videos, code and checkpoints are available on our website.

†Equal contribution
Carnegie Mellon University
{tsungwek,ngkanats,katef}@cs.cmu.edu

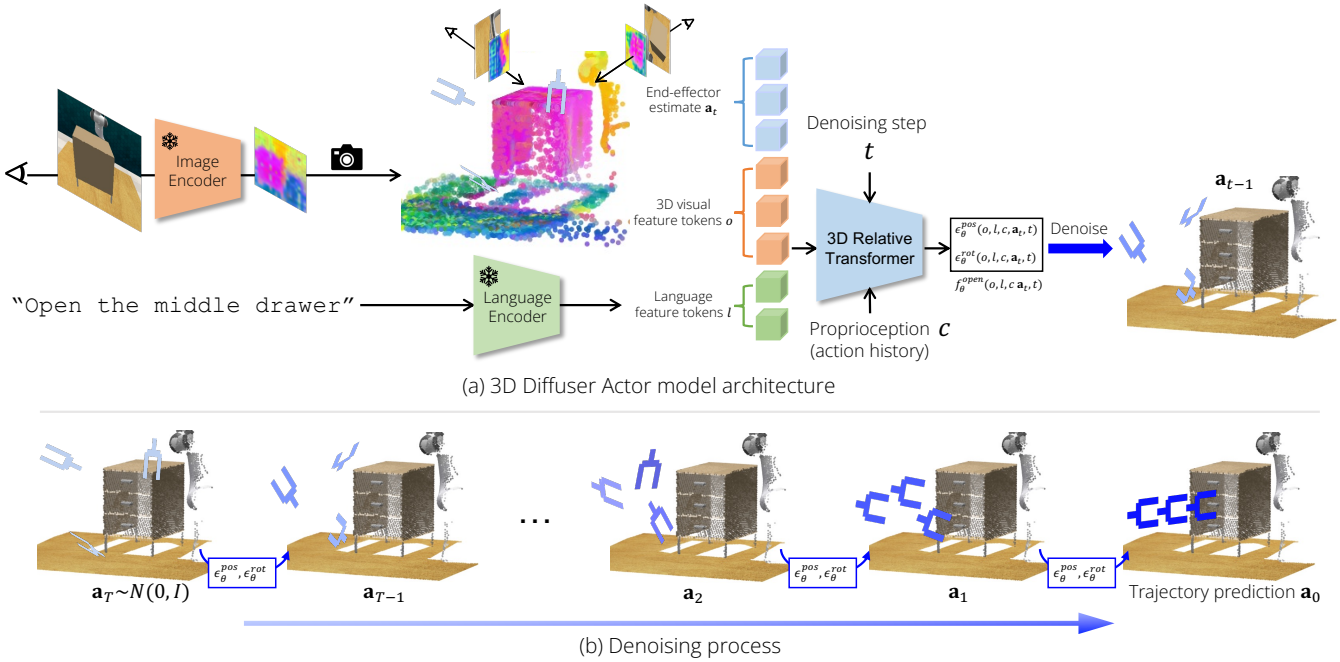


Fig. 2: **3D Diffuser Actor** is a diffusion model of the robot 3D trajectory conditioned on sensory input, language goals and action history. It outputs position and rotation residuals for denoising, as well as the end-effector's state (open/close). During inference, it iteratively denoises the current estimate for the robot's future trajectory, starting from pure noise.

II. Method

3D Diffuser Actor (Figure 2) is a conditional diffusion model that takes as input visual observations, a language instruction, a short history of the robot's end-effectors and the current estimate for the robot's future action trajectory, and predicts the error in the end-effector's 3D translations and 3D orientations for each predicted timestep.

Observation space and action representation 3D Diffuser Actor is trained on demonstrations of successful trajectories in the form of $\{(o_1, \mathbf{a}_1), (o_2, \mathbf{a}_2), ...\}$, accompanied with a task language description [21], [13], [16], [22]. Each observation o comprises a set of posed RGB-D images. Each action $\mathbf{a}$ describes an end-effector pose and is decomposed into 3D position, 3D orientation and a binary open/closed state: $\mathbf{a} = \{\mathbf{a}^{\text{pos}} \in \mathbb{R}^3, \mathbf{a}^{\text{rot}} \in \mathbb{R}^6, \mathbf{a}^{\text{open}} \in \{0, 1\}\}$. We represent rotations using the 6D rotation representation of [23] to avoid the discontinuities of the quaternion representation.

Keyposes 3D Diffuser Actor inherits the temporal abstraction of demonstrations into end-effector *keyposes* [21], [13], [24] (Appendix, Section C). During inference, 3D Diffuser Actor can either predict and execute the full trajectory of actions up to the next keypose (including the keypose), or just predict the next keypose and use a sampling-based motion planner to reach it, similar to previous works [13], [25], [16]. In the rest of this section we use the term "trajectory" to refer to 3D Diffuser Actor's output. Keypose prediction is a special case where the trajectory contains only one future pose.

Scene and language encoder We use a scene and language encoder similar to [16], [26]. The input is a set of posed RGB-D images. We first extract multi-scale visual tokens for each camera view using a pre-trained 2D feature extractor and a feature pyramid network [27]. Next, we associate every 2D feature grid location in the 2D feature maps with a depth value, by averaging the depth values of the image pixels that correspond to it. We use camera intrinsics and the pinhole camera equation to map a pixel location and depth value (x, y, d) to a 3D location (X, Y, Z), and "lift" the 2D feature tokens to 3D, to obtain a 3D feature cloud. The language encoder maps language task descriptions or instructions into language feature tokens. We use the pre-trained CLIP ResNet50 2D image encoder [28] to encode each RGB image into a 2D feature map and the pre-trained CLIP language encoder to featurize the language task instruction.

3D Relative Position Denoising Transformer 3D Diffuser Actor iteratively denoises an estimate of the end-effector's future trajectory. Every trajectory step token is a 10D action, as defined earlier. We project each such action into a high-dimensional representation through an MLP. Each action token comes with the positional embeddings of its corresponding $\mathbf{a}^{\text{pos}}$. This enables relative cross-attention with the scene and other entities. We incorporate proprioceptive information as a short history of predicted end-effector keyposes. These are represented with learnable feature vectors and their 3D positions are used to compute positional embeddings.

We contextualize all tokens (visual o, language l, proprioception c and current action estimate $\mathbf{a}_t^{pos}, \mathbf{a}_t^{rot}$) using 3D relative position attention layers [29], [30]. The updated action feature tokens are fed to MLPs to predict the position error $\epsilon_\theta^{\text{pos}}(o, l, c, \mathbf{a}_t^{\text{pos}}, \mathbf{a}_t^{\text{rot}}, t)$, rotation error $\epsilon_\theta^{\text{rot}}(o, l, c, \mathbf{a}_t^{\text{pos}}, \mathbf{a}_t^{\text{rot}}, t)$ and whether the end-effector should

be open or closed $f_\theta^{\text{open}}(o, l, c, \mathbf{a}_t^{\text{pos}}, \mathbf{a}_t^{\text{rot}}, t)$. We update the current estimate of each element of the end-effector's trajectory:

$$\mathbf{a}_{t-1}^{\text{pos}} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{a}_t^{\text{pos}} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta^{\text{pos}}(o, l, c, \mathbf{a}_t^{\text{pos}}, \mathbf{a}_t^{\text{rot}}, t) \right) + \frac{1-\bar{\alpha}_{t+1}}{1-\bar{\alpha}_t} \beta_t \mathbf{z}^{pos} \quad (1)$$

$$\mathbf{a}_{t-1}^{\text{rot}} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{a}_t^{\text{rot}} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta^{\text{rot}}(o, l, c, \mathbf{a}_t^{\text{rot}}, \mathbf{a}_t^{\text{rot}}, t) \right) + \frac{1-\bar{\alpha}_{t+1}}{1-\bar{\alpha}_t} \beta_t \mathbf{z}^{rot} \quad (2)$$

where $\mathbf{z}^{pos}, \mathbf{z}^{rot} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ variables of appropriate dimension. We found using a scaled-linear noise schedule to denoise thet end-effector's 3D positions and a squared cosine noise schedule for the end-effector's 3D orientations to converge much faster than using squared cosine noise for both.

Training 3D Diffuser Actor is trained on a dataset of RGB-D observations o, proprioception information c, action trajectories $\mathbf{a} = [\mathbf{a}^{pos}, \mathbf{a}^{rot}]$ and language goals $\mathcal{D} = \{(o_1, c_1, \mathbf{a}_1, l_1), (o_2, c_2, \mathbf{a}_2, l_2), ...\}$. During training, we randomly sample a diffusion step t and add noise $\epsilon = (\epsilon_t^{\text{pos}}, \epsilon_t^{\text{rot}})$ to the ground-truth action. Our model learns to reconstruct the clean actions by predicting the pose residual with respect to the current estimate. We adopt the $L1$ loss for reconstructing the 3D position and 3D rotation error. We use binary cross-entropy loss to supervise end-effector opening f_θ^{open}:

$$\mathcal{L}_\theta = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \text{BCE}(f_\theta^{\text{open}}(o_i, l_i, c_i, \mathbf{a}_{t,i}^{\text{pos}}, \mathbf{a}_{t,i}^{\text{rot}}, t), \mathbf{a}_i^{\text{open}}) + w_1 \cdot \|(\epsilon_\theta^{\text{pos}}(o_i, l_i, c_i, \mathbf{a}_{t,i}^{\text{pos}}, \mathbf{a}_{t,i}^{\text{rot}}, t) - \epsilon_t^{\text{pos}}\| + w_2 \cdot \|(\epsilon_\theta^{\text{rot}}(o_i, l_i, c_i, \mathbf{a}_{t,i}^{\text{pos}}, \mathbf{a}_{t,i}^{\text{rot}}, t) - \epsilon_t^{\text{rot}}\|, \quad (3)$$

where w_1, w_2 are hyperparameters.

III. Experiments

We test 3D Diffuser Actor on RLBench [11] and CALVIN [12] against the current state-of-the-art and ablative versions of our model. We also test our model in the real world.

A. Evaluation on RLBench

Setup A Franka Panda Robot is used to manipulate the scene. Our model is trained to predict the next end-effector keypose and employs the motion planner BiRRT [31] to reach it pose [13], [16]. We train and evaluate on the *PerAct setup* [13]. This uses a suite of 18 manipulation tasks with 2-60 variations per task, specified by language descriptions. There are 100 training demonstrations available per task and 25 unseen test episodes for each task. Four cameras are available. A more detailed discussion of the setup, baselines and results on a single-camera setup is included in our appendix (Section D).

Results In Figure 1 right, 3D Diffuser Actor achieves an average 81.3% success rate among all 18 tasks, an absolute improvement of $+18.1\%$ over the previous state-of-the-art.

Ablations In Table I. 3D Diffuser Actor largely outperforms its 2D counterpart, 2D Diffuser Actor, underlining the importance of 3D scene representations. Using absolute 3D attentions (3D Diffuser Actor w/o Rel. Attn.) is significantly worse, indicating the importance of translation equivariance. Notably, this baseline already outperforms all prior arts in Figure 1, proving the effectiveness of marrying 3D representations and diffusion policies.

	Avg. Success.
2D Diffuser Actor	47.0
3D Diffuser Actor w/o Rel. Attn.	71.3
3D Diffuser Actor (ours)	**81.3**

TABLE I: **Ablation study** on 18 RLBench tasks.

close box	put duck	insert peg into hole	insert peg into torus	put mouse	open pen
100	100	50	30	80	100
press stapler	**put grapes**	**sort rectangle**	**stack blocks**	**stack cups**	**put block in triangle**
90	90	50	20	40	90

TABLE II: **Multi-Task performance on real-world tasks.**

B. Evaluation on CALVIN

Setup A Franka Panda Robot arm manipulates the scene. CALVIN consists of 34 tasks and 4 different environments (A, B, C and D), differing mostly on texture and object placements. We train 3D Diffuser Actor on the environments A, B and C and evaluate on 1000 unique instruction chains on environment D. Each instruction chain includes five language instructions that need to be executed sequentially. We extract keyposes (Section C) and train our model to predict both the end-effector keypose and the corresponding trajectory to reach it (see Section E for more).

Results 3D Diffuser Actor achieves state-of-the-art results (Figure 1 left), completing on average 3.27 tasks in a row.

C. Evaluation in the real world

We use a Franka Emika robot and capture visual observations with a Azure Kinect RGB-D sensor at a front view. We use 12 tasks and 15 demonstrations per task where we record the keyposes. We refer to the appendix (Section G) for more details. 3D Diffuser Actor is trained to predict the next end-effector keypose and uses the BiRRT [31] planner provided by the MoveIt! ROS package [32] to reach it. We measure the success rate on 10 episodes per task in Table II.

IV. Conclusion

We present 3D Diffuser Actor, a 3D robot manipulation policy with action diffusion that sets a new state-of-the-art on RLBench and CALVIN by a large margin and learns in the real-world from a handful of demonstrations. Our future work will attempt to train 3D Diffuser Actor in domain-randomized simulation environments at a large scale, to help transfer to the real world.

REFERENCES

[1] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016. [Online]. Available: http://arxiv.org/abs/1606.03476

[2] Y. Tsurumine and T. Matsubara, "Goal-aware generative adversarial imitation learning from imperfect demonstration for robotic cloth manipulation," 2022.

[3] K. Hausman, Y. Chebotar, S. Schaal, G. S. Sukhatme, and J. J. Lim, "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets," *CoRR*, vol. abs/1705.10479, 2017. [Online]. Available: http://arxiv.org/abs/1705.10479

[4] N. M. M. Shafiullah, Z. J. Cui, A. Altanzaya, and L. Pinto, "Behavior transformers: Cloning k modes with one stone," 2022.

[5] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Z. Tan, I. Momennejad, K. Hofmann, and S. Devlin, "Imitating human behaviour with diffusion models," 2023.

[6] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *arXiv preprint arXiv:2303.04137*, 2023.

[7] M. Reuss, M. Li, X. Jia, and R. Lioutikov, "Goal-conditioned imitation learning using score-based diffusion policies," *arXiv preprint arXiv:2304.02532*, 2023.

[8] A. Mandlekar, F. Ramos, B. Boots, L. Fei-Fei, A. Garg, and D. Fox, "IRIS: implicit reinforcement without interaction at scale for learning control from offline robot manipulation data," *CoRR*, vol. abs/1911.05321, 2019. [Online]. Available: http://arxiv.org/abs/1911.05321

[9] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '07. New York, NY, USA: Association for Computing Machinery, 2007. [Online]. Available: https://doi.org/10.1145/1329125.1329407

[10] P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson, "Implicit behavioral cloning," *CoRR*, vol. abs/2109.00137, 2021. [Online]. Available: https://arxiv.org/abs/2109.00137

[11] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[12] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7327–7334, 2022.

[13] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 785–799.

[14] S. James, K. Wada, T. Laidlow, and A. J. Davison, "Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 739–13 748.

[15] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox, "Rvt: Robotic view transformer for 3d object manipulation," *arXiv preprint arXiv:2306.14896*, 2023.

[16] T. Gervet, Z. Xian, N. Gkanatsios, and K. Fragkiadaki, "Act3d: Infinite resolution action detection transformer for robotic manipulation," *arXiv preprint arXiv:2306.17817*, 2023.

[17] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2021, pp. 726–747.

[18] H. Huang, O. Howell, X. Zhu, D. Wang, R. Walters, and R. Platt, "Fourier transporter: Bi-equivariant robotic manipulation in 3d," in *ICLR*, 2024.

[19] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," 2018.

[20] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *arXiv preprint arXiv:2104.09864*, 2021.

[21] S. James and A. J. Davison, "Q-attention: Enabling efficient learning for vision-based robotic manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1612–1619, 2022.

[22] H. Wu, Y. Jing, C. Cheang, G. Chen, J. Xu, X. Li, M. Liu, H. Li, and T. Kong, "Unleashing large-scale video generative pre-training for visual robot manipulation," *arXiv preprint arXiv:2312.13139*, 2023.

[23] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," 2020.

[24] H. Liu, L. Lee, K. Lee, and P. Abbeel, "Instruction-following agents with jointly pre-trained vision-language models," *arXiv preprint arXiv:2210.13431*, 2022.

[25] P.-L. Guhur, S. Chen, R. G. Pinel, M. Tapaswi, I. Laptev, and C. Schmid, "Instruction-driven history-aware policies for robotic manipulations," in *Conference on Robot Learning*. PMLR, 2023, pp. 175–187.

[26] Z. Xian, N. Gkanatsios, T. Gervet, T.-W. Ke, and K. Fragkiadaki, "Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 2323–2339.

[27] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[28] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[29] Y. Li and T. Harada, "Lepard: Learning partial point cloud matching in rigid and deformable scenes," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[30] N. Gkanatsios, M. K. Singh, Z. Fang, S. Tulsiani, and K. Fragkiadaki, "Analogy-forming transformers for few-shot 3d parsing," *ArXiv*, vol. abs/2304.14382, 2023.

[31] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[32] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[33] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Proceedings of (NeurIPS) Neural Information Processing Systems*, D. Touretzky, Ed. Morgan Kaufmann, December 1989, pp. 305 – 313.

[34] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[35] C. Lynch and P. Sermanet, "Grounding language in play," *CoRR*, vol. abs/2005.07648, 2020. [Online]. Available: https://arxiv.org/abs/2005.07648

[36] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, "Goal-conditioned imitation learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/c8d3a760ebab631565f8509d84b3b3f1-Paper.pdf

[37] Z. J. Cui, Y. Wang, N. M. M. Shafiullah, and L. Pinto, "From play to policy: Conditional behavior generation from uncurated robot data," *ArXiv*, vol. abs/2210.10047, 2022.

[38] D.-N. Ta, E. Cousineau, H. Zhao, and S. Feng, "Conditional energy-based models for implicit policies: The gap between theory and practice," 2022.

[39] N. Gkanatsios, A. Jain, Z. Xian, Y. Zhang, C. Atkeson, and K. Fragkiadaki, "Energy-based models as zero-shot planners for compositional scene rearrangement," *arXiv preprint arXiv:2304.14391*, 2023.

[40] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," 2015.

[41] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *CoRR*, vol. abs/2006.11239, 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[42] S. Singh, S. Tu, and V. Sindhwani, "Revisiting energy based models as policies: Ranking noise contrastive estimation and interpolating energy models," 2023.

[43] T. Salimans and J. Ho, "Should EBMs model the energy or the score?" in *Energy Based Models Workshop - ICLR 2021*, 2021. [Online]. Available: https://openreview.net/forum?id=9AS-TF2jRNb

[44] H. Ryu, J. Kim, J. Chang, H. S. Ahn, J. Seo, T. Kim, J. Choi, and R. Horowitz, "Diffusion-edfs: Bi-equivariant denoising generative

modeling on se (3) for visual robotic manipulation," *arXiv preprint arXiv:2309.02685*, 2023.

[45] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki, "Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5923–5930.

[46] Z. Wang, J. J. Hunt, and M. Zhou, "Diffusion policies as an expressive policy class for offline reinforcement learning," *arXiv preprint arXiv:2208.06193*, 2022.

[47] U. A. Mishra and Y. Chen, "Reorientdiff: Diffusion model based reorientation for object manipulation," *arXiv preprint arXiv:2303.12700*, 2023.

[48] W. Liu, T. Hermans, S. Chernova, and C. Paxton, "Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects," *arXiv preprint arXiv:2211.04604*, 2022.

[49] A. Simeonov, A. Goyal, L. Manuelli, L. Yen-Chen, A. Sarmiento, A. Rodriguez, P. Agrawal, and D. Fox, "Shelving, stacking, hanging: Relational pose diffusion for multi-modal rearrangement," *arXiv preprint arXiv:2307.04751*, 2023.

[50] X. Fang, C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Dimsam: Diffusion models as samplers for task and motion planning under partial observability," *arXiv preprint arXiv:2306.13196*, 2023.

[51] I. Kapelyukh, V. Vosylius, and E. Johns, "Dall-e-bot: Introducing web-scale diffusion models to robotics," *IEEE Robotics and Automation Letters*, 2023.

[52] Y. Dai, M. Yang, B. Dai, H. Dai, O. Nachum, J. Tenenbaum, D. Schuurmans, and P. Abbeel, "Learning universal policies via text-guided video generation," *arXiv preprint arXiv:2302.00111*, 2023.

[53] A. Ajay, S. Han, Y. Du, S. Li, G. Abhi, T. Jaakkola, J. Tenenbaum, L. Kaelbling, A. Srivastava, and P. Agrawal, "Compositional foundation models for hierarchical planning," *arXiv preprint arXiv:2309.08587*, 2023.

[54] K. Black, M. Nakamoto, P. Atreya, H. Walke, C. Finn, A. Kumar, and S. Levine, "Zero-shot robotic manipulation with pretrained image-editing diffusion models," *arXiv preprint arXiv:2310.10639*, 2023.

[55] H. Chen, C. Lu, C. Ying, H. Su, and J. Zhu, "Offline reinforcement learning via high-fidelity generative behavior modeling," 2023.

[56] B. Yang, H. Su, N. Gkanatsios, T.-W. Ke, A. Jain, J. Schneider, and K. Fragkiadaki, "Diffusion-es: Gradient-free planning with diffusion for autonomous driving and zero-shot instruction following," *ArXiv*, vol. abs/2402.06559, 2024.

[57] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine, "Idql: Implicit q-learning as an actor-critic method with diffusion policies," *arXiv preprint arXiv:2304.10573*, 2023.

[58] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," *arXiv preprint arXiv:2205.09991*, 2022.

[59] H. He, C. Bai, K. Xu, Z. Yang, W. Zhang, D. Wang, B. Zhao, and X. Li, "Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning," *arXiv preprint arXiv:2305.18459*, 2023.

[60] Z. Wang, T. Oba, T. Yoneda, R. Shen, M. R. Walter, and B. C. Stadie, "Cold diffusion on the replay buffer: Learning to plan from known good states," *ArXiv*, vol. abs/2310.13914, 2023.

[61] Z. Liang, Y. Mu, M. Ding, F. Ni, M. Tomizuka, and P. Luo, "Adaptdiffuser: Diffusion models as adaptive self-evolving planners," *arXiv preprint arXiv:2302.01877*, 2023.

[62] Z. Chen, S. Kiami, A. Gupta, and V. Kumar, "Genaug: Retargeting behaviors to unseen situations via generative augmentation," *arXiv preprint arXiv:2302.06671*, 2023.

[63] Z. Mandi, H. Bharadhwaj, V. Moens, S. Song, A. Rajeswaran, and V. Kumar, "Cacti: A framework for scalable multi-task multi-scene visual imitation learning," *arXiv preprint arXiv:2212.05711*, 2022.

[64] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[65] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," 2023.

[66] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, *et al.*, "A generalist agent," *arXiv preprint arXiv:2205.06175*, 2022.

[67] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 991–1002.

[68] A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, A. Raffin, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Ichter, C. Lu, C. Xu, C. Finn, C. Xu, C. Chi, C. Huang, C. Chan, C. Pan, C. Fu, C. Devin, D. Driess, D. Pathak, D. Shah, D. Büchler, D. Kalashnikov, D. Sadigh, E. Johns, F. Ceola, F. Xia, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Schiavi, H. Su, H. Fang, H. Shi, H. B. Amor, H. I. Christensen, H. Furuta, H. Walke, H. Fang, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Kim, J. Schneider, J. Hsu, J. Bohg, J. Bingham, J. Wu, J. Wu, J. Luo, J. Gu, J. Tan, J. Oh, J. Malik, J. Tompson, J. Yang, J. J. Lim, J. Silvério, J. Han, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Zhang, K. Majd, K. Rana, K. P. Srinivasan, L. Y. Chen, L. Pinto, L. Tan, L. Ott, L. Lee, M. Tomizuka, M. Du, M. Ahn, M. Zhang, M. Ding, M. K. Srirama, M. Sharma, M. J. Kim, N. Kanazawa, N. Hansen, N. M. O. Heess, N. J. Joshi, N. Suenderhauf, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, P. R. Sanketi, P. Wohlhart, P. Xu, P. Sermanet, P. Sundaresan, Q. H. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Mendonca, R. Shah, R. Hoque, R. C. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Moore, S. Bahl, S. Dass, S. Song, S. Xu, S. Haldar, S. O. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Dasari, S. Belkhale, T. Osa, T. Harada, T. Matsushima, T. Xiao, T. Yu, T. Ding, T. Davchev, T. Zhao, T. Armstrong, T. Darrell, V. Jain, V. Vanhoucke, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Wang, X. Zhu, X. Li, Y. Lu, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Xu, Y. Wang, Y. Bisk, Y. Cho, Y. Lee, Y. Cui, Y. hua Wu, Y. Tang, Y. Zhu, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Xu, and Z. J. Cui, "Open x-embodiment: Robotic learning datasets and rt-x models," *ArXiv*, vol. abs/2310.08864, 2023.

[69] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu, J. Luo, T. Kreiman, Y. Tan, D. Sadigh, C. Finn, and S. Levine, "Octo: An open-source generalist robot policy," https://octo-models.github.io, 2023.

[70] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng, "Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks," 2021.

[71] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 894–906.

[72] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, "Perceiver: General perception with iterative attention," 2021.

[73] C. Lynch and P. Sermanet, "Language conditioned imitation learning over unstructured data," *arXiv preprint arXiv:2005.07648*, 2020.

[74] O. Mees, L. Hermann, and W. Burgard, "What matters in language conditioned robotic imitation learning over unstructured data," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, pp. 11 205–11 212, 2022.

[75] O. Mees, J. Borja-Diaz, and W. Burgard, "Grounding language with visual affordances over unstructured data," in *ICRA*, 2023.

[76] X. Li, M. Liu, H. Zhang, C. Yu, J. Xu, H. Wu, C. Cheang, Y. Jing, W. Zhang, H. Liu, *et al.*, "Vision-language foundation models as effective robot imitators," *arXiv preprint arXiv:2311.01378*, 2023.

[77] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 392–18 402.

[78] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," *arXiv preprint arXiv:2210.08402*, 2022.

[79] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ international*

conference on intelligent robots and systems. IEEE, 2013, pp. 1321–1326.

[80] Y. Ze, G. Yan, Y.-H. Wu, A. Macaluso, Y. Ge, J. Ye, N. Hansen, L. E. Li, and X. Wang, "Gnfactor: Multi-task real robot learning with generalizable neural feature fields," *arXiv preprint arXiv:2308.16891*, 2023.

[81] S. Chen, R. G. Pinel, C. Schmid, and I. Laptev, "Polarnet: 3d point clouds for language-guided robotic manipulation," *ArXiv*, vol. abs/2309.15596, 2023.

[82] G. Qian, Y. Li, H. Peng, J. Mai, H. A. A. K. Hammoud, M. Elhoseiny, and B. Ghanem, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies," in *NeurIPS*, 2022.

[83] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[84] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[85] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

APPENDIX

A. Acknowledgements

This work is supported by Sony AI, NSF award No 1849287, DARPA Machine Common Sense, an Amazon faculty award, and an NSF CAREER award. The authors would like to thank Moritz Reuss for useful training tips on CALVIN; Zhou Xian for help with the real-robot experiments; Brian Yang for discussions, comments and efforts in the early development of this paper.

B. Related Work

Learning robot manipulation policies from demonstrations Though state-to-action mappings are typically multimodal, earlier works on learning from demonstrations train deterministic policies with behavior cloning [33], [34]. To better handle action multimodality, other approaches discretize action dimensions and use cross entropy losses [35], [13], [17]. However, the number of bins needed to approximate a continuous action space grows exponentially with increasing dimensionality. Generative adversarial networks [1], [2], [36], variational autoencoders [8] and combined Categorical and Gaussian distributions [4], [25], [37] have been used to learn from multimodal demonstrations. Nevertheless, these models tend to be sensitive to hyperparameters, such as the number of clusters used [4]. Implicit behaviour cloning represents distributions over actions by using Energy-Based Models (EBMs) [10], [38]. Optimization in EBMs amounts to searching the energy landscape for minimal-energy action, given a state. EBMs are inherently multimodal, since the learned landscape can have more than one minima. EBMs are also composable, which makes them suitable for combining action distributions with additional constraints during inference [39]. Diffusion models [40], [41] are a powerful class of generative models related to EBMs in that they model the score of the distribution, else, the gradient of the energy, as opposed to the energy itself [42], [43]. The key idea behind diffusion models is to iteratively transform a simple prior distribution into a target distribution by applying a sequential denoising process. They have been used for modeling state-conditioned action distributions in imitation learning [44], [45], [5], [46], [7], [47] from low-dimensional input, as well as from visual sensory input, and show both better mode coverage and higher fidelity in action prediction than alternatives. They have not been yet combined with 3D scene representations.

Diffusion models in robotics Beyond policy representations in imitation learning, diffusion models have been used to model cross-object and object-part arrangements [48], [49], [47], [50], [39] and visual image subgoals [51], [52], [53], [54]. They have also been used successfully in offline reinforcement learning [55], [56], [57], where they model the state-conditioned action trajectory distribution [57], [55] or state-action trajectory distribution [58], [59], [60]. ChainedDiffuser [26] proposes to replace motion planners, commonly used for keypose to keypose linking, with a trajectory diffusion model that conditions on the 3D scene feature cloud and the predicted target 3D keypose to denoise a trajectory from the current to the target keypose. 3D Diffuser Actor instead predicts the next 3D keypose for the robot's end-effector alongside the linking trajectory, which is a much harder task than linking two given keyposes. Lastly, image diffusion models have been used for augmenting the conditioning images input to robot policies to help the latter generalize better [61], [62], [63].

2D and 3D scene representations for robot manipulation End-to-end image-to-action policy models, such as RT-1 [64], RT-2[65], GATO [66], BC-Z [67], RT-X [68], Octo [69] and InstructRL [24] leverage transformer architectures for the direct prediction of 6-DoF end-effector poses from 2D video input. However, this approach comes at the cost of requiring thousands of demonstrations to implicitly model 3D geometry and adapt to variations in the training domains. Another line of research is centered around Transporter networks [17], [70], [71], [39], demonstrating remarkable few-shot generalization by framing end-effector pose prediction as pixel classification. Nevertheless, these models are usually confined to top-down 2D planar environments with simple pick-and-place primitives. Direct extensions to 3D, exemplified by C2F-ARM [14] and PerAct [13], involve voxelizing the robot's workspace and learning to identify the 3D voxel containing the next end-effector keypose. However, this becomes computationally expensive as resolution requirements increase. Consequently, related approaches resort to either coarse-to-fine voxelization or efficient attention operations [72] to mitigate computational costs. Act3D [16] foregoes 3D scene voxelization altogether; it instead computes a 3D action map of variable spatial resolution by sampling 3D points in the empty workspace and featurizing them using cross-attentions to the 3D physical points. Robotic View Transformer (RVT) [15] re-projects the input RGB-D image to alternative image views, featurizes those and lifts the predictions to 3D to infer 3D locations for the robot's end-effector. Both Act3D and RVT show currently the highest performance on RLBench [11]. Our model outperforms them by a big margin, as we show in the experimental section.

Instruction-conditioned long-horizon policies To generate actions based on language instructions, early works [73], [74], [64] employ a text encoder to map language instructions to latent features, then they deploy 2D policies that condition on these language latents to predict actions. HULC++ [75] alternates between a free-space policy that reaches subgoals (next object to interact with) and a local policy [74] for low-level interaction. However, these models struggle to infer 3D actions precisely from 2D observations and language instructions and do not generalize to new environments with different texture. Recent works [54], [76], [22] explore the use of large-scale pre-training to boost their performance. SuSIE [54] proposes to synthesize visual subgoals using InstructPix2Pix [77], an instruction-conditioned image generative model pre-trained on a subset of 5 billion images [78]. Actions are then predicted by a low-level diffusion policy conditioned on both current observations and synthesized visual goals. RoboFlamingo [76] fine-tunes existing vision-language models, pre-trained for solving vision and language tasks, to

predict the robot's actions. GR-1 [22] pre-trains a GPT-style transformer for video generation on a massive video corpus, then jointly trains the model for predicting robot actions and future observations. We show that 3D Diffuser Actor can exceed the performance of these policies thanks to effective use of 3D representations.

C. Our heuristics for keypose discovery

For RLBench we use the heuristics from [21], [14]: a pose is a keypose if (1) the end-effector state changes (grasp or release) or (2) the velocity's magnitude approaches zero (often at pre-grasp poses or a new phase of a task). For our real-world experiments we maintain the above heuristics and record pre-grasp poses as well as the poses at the beginning of each phase of a task, e.g., when the end-effector is right above an object of interest. We report the number of keyposes per real-world task in this Appendix (Section G). Lastly, for CALVIN we adapt the above heuristics to devise a more robust algorithm to discover keyposes. Specifically, we track end-effector state changes and significant changes of motion, i.e. both velocity and acceleration. For reference, we include our Python code for discovering keyposes in CALVIN here:

```python
"""Utility functions for computing keyposes."""
import numpy as np
from scipy.signal import argrelextrem

def motion_changed(trajectories, buffer_size):
    """Select keyposes where motion changes
    significantly. The chosen poses shall be sparse
    .

    Args:
        trajectories: a list of 1D array
        buffer_size: an integer indicates the
           minimum distance of waypoints

    Returns:
        key_poses: a list of integers indicates
           the time steps when motion of the end
           effector changes significantly.
    """
    # compute velocity
    trajectories = np.stack(
        [trajectories[0]] + trajectories, axis=0
    )
    velocities = (
        trajectories[1:] - trajectories[:-1]
    )

    # compute acceleration
    velocities = np.concatenate(
        [velocities, [velocities[-1]]],
        axis=0
    )
    accelerations = velocities[1:] - velocities
    [:-1]

    # compute the magnitude of acceleration
    A = np.linalg.norm(
        accelerations[:, :3], axis=-1
    )

    # local maximas of acceleration indicates
    # significant motion change of the end effector
    local_max_A = argrelextrema(A, np.greater)[0]

    # consider the top 20% of local maximas
    K = int(A.shape[0] * 0.2)
    topK = np.sort(A)[::-1][K]
    large_A = A[local_max_A] >= topK
    local_max_A = local_max_A[large_A].tolist()

    # select waypoints sparsely
    key_poses = [local_max_A.pop(0)]
    for i in local_max_A:
        if i - key_poses[-1] >= buffer_size:
            key_poses.append(i)

    return key_poses

def gripper_state_changed(trajectories):
    """Select keyposes where the end-effector
    opens/closes.

    Args:
        trajectories: a list of 1D array

    Returns:
        key_poses: a list of integers indicates
           the time steps when the end-effector
           opens/closes.
    """
    trajectories = np.stack(
        [trajectories[0]] + trajectories, axis=0
    )
    openess = trajectories[:, -1]
    changed = openess[:-1] != openess[1:]

    key_poses = np.where(changed)[0].tolist()

    return key_poses

def keypoint_discovery(trajectories, buffer_size=5)
    :
    """Select keyposes where motion changes
    significantly. The chosen poses shall be sparse
    .

    Args:
        trajectories: a list of 1D array
        buffer_size: an integer indicates the
           minimum distance of waypoints

    Returns:
        key_poses: an Integer array indicates the
           indices of keyposes
    """
    motion_changed = motion_changed(
        trajectories, buffer_size
    )

    gripper_changed = (
        gripper_state_changed(trajectories)
    )
    one_frame_before_gripper_changed = [
        i - 1 for i in gripper_changed if i > 1
    ]

    last_frame = [len(trajectories) - 1]

    key_pose_inds = (
        moition_changed +
        gripper_changed.tolist() +
        one_frame_before_gripper_changed.tolist() +
        last_frame
    )
    key_pose_inds = np.unique(key_pose_inds)

    return key_pose_inds
```

D. Evaluation on RLBench

Setup RLBench is built atop the CoppelaSim [79] simulator, where a Franka Panda Robot is used to manipulate the scene. Our 3D Diffuser Actor is trained to predict the next end-effector keypose and we employ the low-level motion planner BiRRT [31], native to RLBench, to reach the predicted pose, following previous works [13], [16]. Our model does not perform collision checking for our experiments. We train and evaluate 3D Diffuser Actor on two experimental setups of multi-task manipulation:

1) *PerAct setup* introduced in [13]: This uses a suite of 18 manipulation tasks, each task has 2-60 variations, which concern scene variability across object poses, appearance and semantics. The tasks are specified by language descriptions. There are four cameras available (front, wrist, left shoulder, right shoulder). There are 100 training demonstrations available per task, evenly split across task variations and 25 unseen test episodes for each task. Due to the randomness of the sampling-based motion planner, we test our model across 5 random seeds. During evaluation, models are allowed to predict a maximum of 25 keyposes, unless they receive earlier task-completion indicators from the simulation environment.
2) *GNFactor setup* introduced in [80]: This uses a suite of 10 manipulation tasks (a subset of PerAct's task set). Only one RGB-D camera view is available (front camera). There are 20 training demonstrations per task, evenly split across variations. The evaluation setup mirrors that of PerAct, with the exception that we test the final checkpoint using 3, rather than 5, random seeds on the test set.

Baselines We consider the following baselines:

1) C2F-ARM-BC [14], a 3D policy that iteratively voxelizes RGB-D images and predicts actions in a coarse-to-fine manner. Q-values are estimated within each voxel and the translation action is determined by the centroid of the voxel with the maximal Q-values.
2) PerAct [13], a 3D policy that voxelizes the workspace and detects the next voxel action through global self-attention.
3) Hiveformer [25], a 3D policy that enables attention between features of different history time steps.
4) PolarNet [81], a 3D policy that computes dense point representations for the robot workspace using a PointNext backbone [82].
5) RVT [15], a 3D policy that deploys a multi-view transformer to predict actions and fuses those across views by back-projecting to 3D.
6) Act3D [16], a 3D policy that featurizes the robot's 3D workspace using coarse-to-fine sampling and featurization.
7) GNFactor [80], a 3D policy that co-optimizes a neural field for reconstructing the 3D voxels of the input scene and a PerAct module for predicting actions based on voxel representations.

We report results for RVT, PolarNet and GNFactor based on their respective papers. Results for CF2-ARM-BC and PerAct are presented as reported in RVT [15]. Results for Hiveformer are copied from PolarNet [81].

We observed that Act3D [16] does not follow the same setup as PerAct on RLBench. Specifically, Act3D uses different 1) 3D object models, 2) success conditions, 3) training/test episodes and 4) maximum numbers of keyposes during evaluation. For fair comparison, we retrain and test Act3D on the same setup.

We also compare to the following ablative versions of our model:

1) *2D Diffuser Actor*, our implementation of [6]. We remove the 3D scene encoding from 3D Diffuser Actor and instead use per-image 2D representations by average-pooling features within each view. We add learnable embeddings to distinguish between different views. We use standard attention layers for joint encoding the action estimate and 2D image features.
2) *3D Diffuser Actor w/o Rel. Attn.*, an ablative version of our model that uses standard (non-relative) attentions to featurize the current rotation and translation estimate with the 3D scene feature cloud. This version of our model is not translation-equivariant.

Evaluation metrics Following previous work [16], [13], we evaluate policies by task completion success rate, which is the proportion of execution trajectories that achieve the goal conditions specified in the language instructions.

Results on the PerAct setup We show quantitative results in Table III. Our 3D Diffuser Actor outperforms all baselines on most tasks by a large margin. It achieves an average 81.3% success rate among all 18 tasks, an absolute improvement of $+18.1\%$ over Act3D, the previous state-of-the-art. In particular, 3D Diffuser Actor achieves big leaps on tasks with multiple modes, such as *stack blocks*, *stack cups* and *place cups*, which most baselines fail to complete. We obtain substantial improvements of $+39.5\%$, $+18.4\%$, $+41.6\%$ and $+20.8\%$ on *stack blocks*, *put in cupboard*, *insert peg* and *stack cups* respectively.

Results on the GNFactor setup We show quantitative results in Table IV. We train Act3D on this setup using its publicly available code. 3D Diffuser Actor outperforms both GNFactor and Act3D by a significant margin, achieving absolute performance gains of $+46.4\%$ and $+13.1\%$ respectively. Notably, 3D Diffuser Actor and Act3D utilize similar 3D scene representations—sparse 3D feature tokens—while GNFactor featurizes a scene with 3D voxels and learns to reconstruct them. Even with a single camera view, both Act3D and 3D Diffuser Actor outperform GNFactor significantly. This suggests that the choice of 3D scene representation is a more crucial factor than the completion of 3D scenes in developing efficient 3D manipulation policies.

E. Evaluation on CALVIN

The CALVIN benchmark is build on top of the PyBullet [83] simulator and involves a Franka Panda Robot arm

	Avg. Success ↑	open drawer	slide block	sweep to dustpan	meat off grill	turn tap	put in drawer	close jar	drag stick	stack blocks
C2F-ARM-BC [14]	20.1	20	16	0	20	68	4	24	24	0
PerAct [13]	49.4	88.0$_{\pm 5.7}$	74.0$_{\pm 13.0}$	52.0$_{\pm 0.0}$	70.4$_{\pm 2.0}$	88.0$_{\pm 4.4}$	51.2$_{\pm 4.7}$	55.2$_{\pm 4.7}$	89.6$_{\pm 4.1}$	26.4$_{\pm 3.2}$
HiveFormer [25]	45	52	64	28	**100**	80	68	52	76	8
PolarNet [81]	46	84	56	52	**100**	80	32	36	92	4
RVT [15]	62.9	71.2$_{\pm 6.9}$	81.6$_{\pm 5.4}$	72.0$_{\pm 0.0}$	88.0$_{\pm 2.5}$	93.6$_{\pm 4.1}$	88.0$_{\pm 5.7}$	52.0$_{\pm 2.5}$	99.2$_{\pm 1.6}$	28.8$_{\pm 3.9}$
Act3D [16]	63.2	78.4$_{\pm 11.2}$	96.0$_{\pm 2.5}$	**86.4**$_{\pm 6.5}$	95.2$_{\pm 1.6}$	94.4$_{\pm 2.0}$	91.2$_{\pm 6.9}$	**96.8**$_{\pm 3.0}$	80.8$_{\pm 6.4}$	4.0$_{\pm 3.6}$
3D Diffuser Actor (ours)	**81.3**	**89.6**$_{\pm 4.1}$	**97.6**$_{\pm 3.2}$	84.0$_{\pm 4.4}$	96.8$_{\pm 1.6}$	**99.2**$_{\pm 1.6}$	**96.0**$_{\pm 3.6}$	**96.0**$_{\pm 2.5}$	**100.0**$_{\pm 0.0}$	**68.3**$_{\pm 3.3}$

	screw bulb	put in safe	place wine	put in cupboard	sort shape	push buttons	insert peg	stack cups	place cups
C2F-ARM-BC [14]	8	12	8	0	8	72	4	0	0
PerAct [13]	17.6$_{\pm 2.0}$	86.0$_{\pm 3.6}$	44.8$_{\pm 7.8}$	28.0$_{\pm 4.4}$	16.8$_{\pm 4.7}$	92.8$_{\pm 3.0}$	5.6$_{\pm 4.1}$	2.4$_{\pm 2.2}$	2.4$_{\pm 3.2}$
HiveFormer [25]	8	76	80	32	8	84	0	0	0
PolarNet [81]	44	84	40	12	12	96	4	8	0
RVT [15]	48.0$_{\pm 5.7}$	91.2$_{\pm 3.0}$	91.0$_{\pm 5.2}$	49.6$_{\pm 3.2}$	36.0$_{\pm 2.5}$	**100.0**$_{\pm 0.0}$	11.2$_{\pm 3.0}$	26.4$_{\pm 8.2}$	4.0$_{\pm 2.5}$
Act3D [16]	32.8$_{\pm 6.9}$	95.2$_{\pm 4.0}$	59.2$_{\pm 9.3}$	67.2$_{\pm 3.0}$	29.6$_{\pm 3.2}$	93.6$_{\pm 2.0}$	24.0$_{\pm 8.4}$	9.6$_{\pm 6.0}$	3.2$_{\pm 3.0}$
3D Diffuser Actor (ours)	**82.4**$_{\pm 2.0}$	**97.6**$_{\pm 2.0}$	**93.6**$_{\pm 4.8}$	**85.6**$_{\pm 4.1}$	**44.0**$_{\pm 4.4}$	98.4$_{\pm 2.0}$	**65.6**$_{\pm 4.1}$	**47.2**$_{\pm 8.5}$	**24.0**$_{\pm 7.6}$

TABLE III: **Multi-Task performance on RLBench.** We report success rates on 18 RLBench tasks of the same test split as PerAct [13]. Four camera views are used for all experiments. We evaluate the final checkpoint over 5 random seeds with the same 3D object models and success conditions of RLBench [11] as PerAct. We show the mean and standard deviation of success rates average across all random seeds. Our 3D Diffuser Actor outperforms all prior arts among most tasks by a large margin. Variances are included when available.

	Avg. Success.	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
GNFactor [80]	31.7	25.3	76.0	28.0	57.3	50.7	20.0	0.0	37.3	18.7	4.0
Act3D [16]	65.3	52.0	84.0	80.0	66.7	64.0	**100.0**	54.7	86.7	64.0	0.0
3D Diffuser Actor	**78.4**	**82.7**	**89.3**	**94.7**	**88.0**	**80.0**	92.0	**77.3**	**98.7**	**69.3**	**12.0**

TABLE IV: **Multi-Task performance on RLBench with single camera view.** Following GNFactor [80], we report sucess rates on 10 RLBench tasks using only the *front* camera view. All models are trained with 20 demonstrations per task. We evaluate the final checkpoints across 3 seeds with 25 episodes for each task in the test set. Our 3D Diffuser Actor outperforms prior state-of-the-art baselines–GNFactor and Act3D–among most tasks by a large margin.

	Train episodes	Task completed in a row 1	2	3	4	5	Avg. Len
MCIL [73]	All	30.4	1.3	0.2	0.0	0.0	0.31
HULC [74]	All	41.8	16.5	5.7	1.9	1.1	0.67
RT-1 [64]	Lang	53.3	22.2	9.4	3.8	1.3	0.90
RoboFlamingo [76]	Lang	82.4	61.9	46.6	33.1	23.5	2.48
SuSIE [54]	All	87.0	69.0	49.0	38.0	26.0	2.69
GR-1 [22]	Lang	85.4	71.2	59.6	49.7	40.1	3.06
3D Diffuser Actor (ours, 60 keyposes)	Lang	**92.2**	**78.7**	**63.9**	**51.2**	**41.2**	**3.27**
3D Diffuser Actor (ours, 360 keyposes)	Lang	95.7	88.1	78.6	65.7	55.3	3.83

TABLE V: **Zero-shot long-horizon performance on CALVIN.** We report success rates and average length of completed task sequences. All models are trained on play trajectories on environments A, B, C, and tested on 1000 unique instruction chains on environment D. MCIL, HULC and SuSIE are trained with all play trajectories in the training scenes, whereas, RT-1, RoboFlamingo, GR-1 and our 3D Diffuser Actor are trained with the subset of play data annotated with language descriptions. Our 3D Diffuser Actor achieves a new state-of-the-art. For reference, we include our model's performance when allowed for more predictions, in which case the performance significantly increases.

that manipulates the scene. CALVIN consists of 34 tasks[1] and 4 different environments (A, B, C and D). All environments are equipped with a desk, a sliding door, a drawer, a button that turns on/off an LED, a switch that controls a lightbulb

[1]The definition of "task" varies between different benchmarks: CALVIN considers the manipulation of variations (e.g., same objects with different colors) as different tasks, whereas, RLBench considers those the same task.

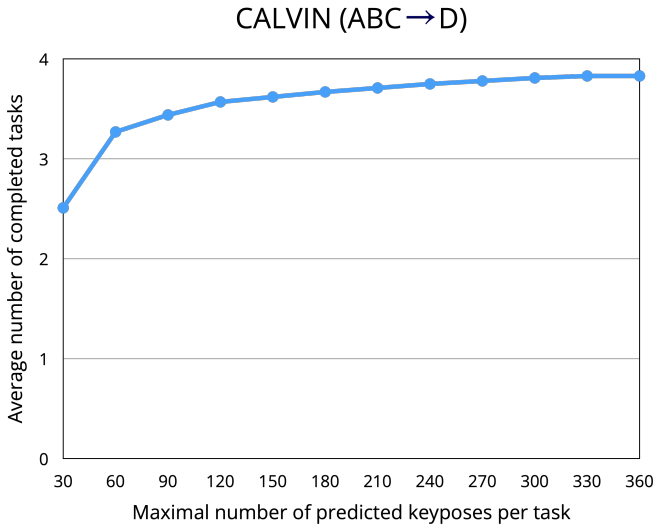


Fig. 3: **Zero-shot long-horizon performance on CALVIN with varying number of keyposes allowed at test time.**

and three different colored blocks (red, blue and pink). These environments differ from each other in the texture of the desk and positions of the objects. CALVIN provides 24 hours of tele-operated unstructured play data, 35% of which are annotated with language descriptions.

We train 3D Diffuser Actor on the subset of play data annotated with language descriptions on the environments A, B and C and evaluate it on 1000 unique instruction chains on environment D, following prior works [76], [22]. Each instruction chain includes five language instructions that need to be executed sequentially. We devise an algorithm (Section C) to extract keyposes on CALVIN, since prior works do not use keyposes on this benchmark. We train our 3D Diffuser Actor to predict both the end-effector pose for each keypose and the corresponding trajectory to reach the predicted pose, instead of using a motion planner. Prior works [74], [54], [76] predict a maximum of 360 actions to complete each instructional task, while, on average, it takes only 60 actions to complete each task using ground-truth trajectories. Our model predicts both keyposes and corresponding trajectories, and, on average, it takes 10 keyposes of each demonstration to complete an instructional task. We thus allow our model to predict a maximum of 60 keyposes for each task. For reference, we also allow our model to predict 360 times and show the influence of this hyperparameter in Section F.

Baselines. We consider the following baselines:

1) MCIL [73], a multi-modal goal-conditioned 2D policy that maps three types of goals–goal images, language instructions and task labels–to a shared latent feature space, and conditions on such latent goals to predict actions.
2) HULC [74], a 2D policy that uses a variational autoencoder to sample a latent plan based on the current observation and task description, then conditions on this latent to predict actions.
3) RT-1 [64], a 2D transformer-based policy that encodes the image and language into a sequence of tokens and employs a Transformer-based architecture that contextualizes these tokens and predicts the arm movement or terminates the episode.
4) RoboFlamingo [76], a 2D policy that adapts existing vision-language models, which are pre-trained for solving vision and language tasks, to robot control. It uses frozen vision and language foundational models and learns a cross-attention between language and visual features, as well as a recurrent policy that predicts the low-level actions conditioned on the language latents.
5) SuSIE [54], a 2D policy that deploys an large-scale pre-trained image generative model [77] to synthesize visual subgoals based on the current observation and language instruction. Actions are then predicted by a low-level goal-conditioned 2D diffusion policy that models inverse dynamics between the current observation and the predicted subgoal image.
6) GR-1 [22], a 2D policy that first pre-trains an autoregressive Transformer on next frame prediction, using a large-scale video corpus without action annotations. Each video frame is encoded into an 1d vector by average-pooling its visual features. Then, the same architecture is fine-tuned in-domain to predict both actions and future observations.

We report results for HULC, RoboFlamingo, SuSIE and GR-1 from the respective papers. Results from MCIL are borrowed from [74]. Results from RT-1 are copied from [22].

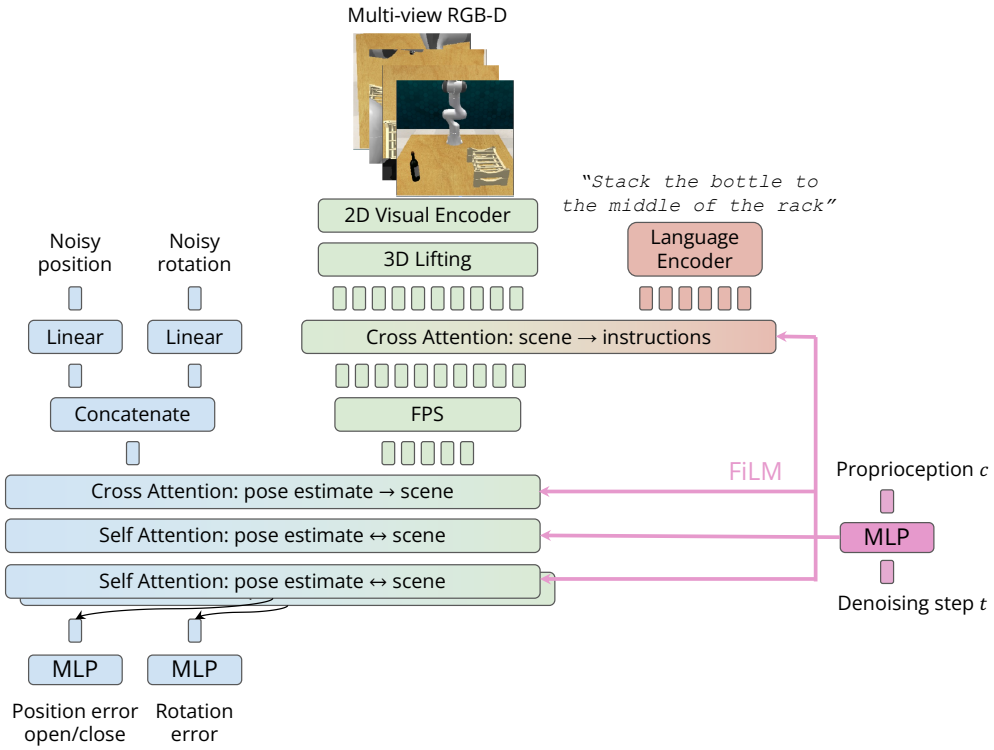
Evaluation metrics Following previous works [74], [22], we report the success rate and the average number of completed sequential tasks.

F. Evaluation on CALVIN with varying number of maximum keyposes allowed at test time

We show zero-shot long-horizon performance on CALVIN with varying number of keyposes allowed at test time in Figure 3. The performance saturates with more than 300 keyposes.

G. Evaluation in the real world

We validate 3D Diffuser Actor in learning manipulation tasks from real-world demonstrations. We use a Franka Emika robot and capture visual observations with a Azure Kinect RGB-D sensor at a front view. Images are originally captured at 1280×720 resolution and downsampled to a resolution of 256×256. Camera extrinsics are calibrated w.r.t the robot base. We use 12 tasks: 1) *close a box*, 2) *put ducks in bowls*, 3) *insert a peg vertically into the hole*, 4) *insert a peg horizontally into the torus*, 5) *put a computer mouse on the pad*, 6) *open the pen*, 7) *press the stapler*, 8) *put grapes in the bowl*, 9) *sort the rectangle*, 10) *stack blocks with the same shape*, 11) *stack cups* and 12) *put block in a triangle on the plate*. All tasks take place in a cluttered scene with distractors (random objects that do not participate in the task) which are not mentioned in the descriptions below.



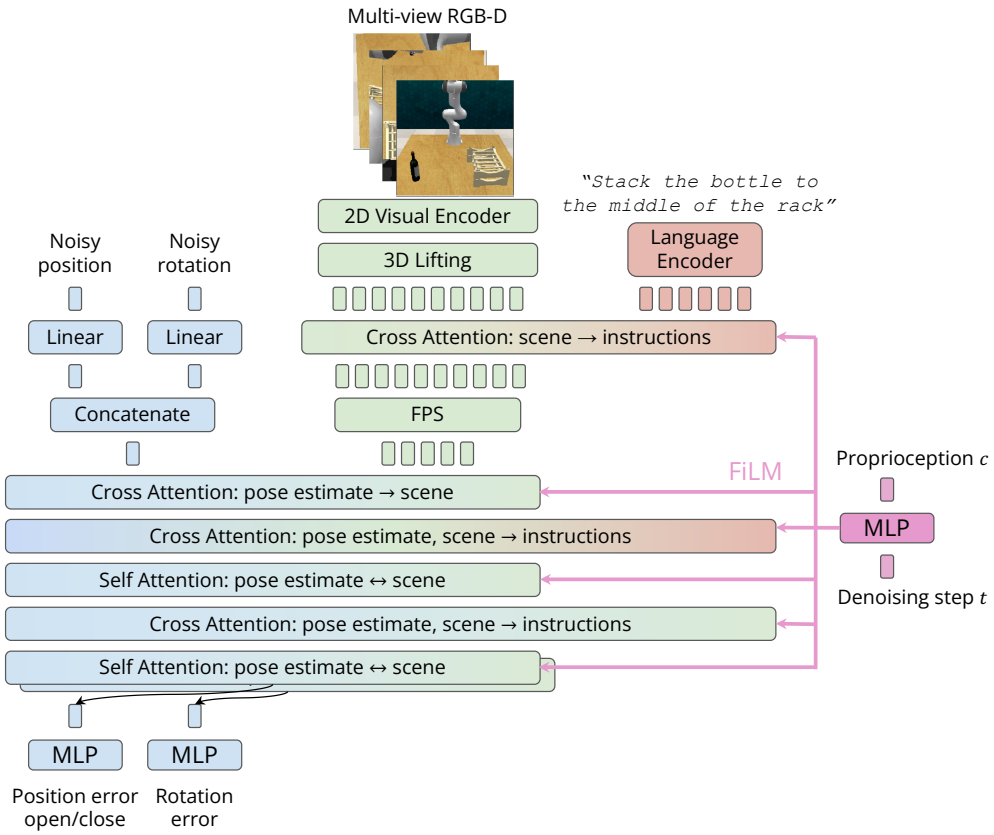


Fig. 4: **3D Diffuser Actor architecture** in more detail. **Top**: Standard version: the encoded inputs are fed to attention layers that predict the position and rotation error for each trajectory timestep. The language information is fused to the visual stream by allowing the encoded visual feature tokens to attend to language feature tokens. There are two different attention and output heads for position and rotation error respectively. **Bottom**: version with enhanced language conditioning: cross-attention layers from visual and pose estimate tokens to language tokens are interleaved between pose estimate-visual token attention layers.

1) Close a box: The end-effector needs to move and hit the lid of an open box so that it closes. The agent is successful if the box closes. The task involves two keyposes.
2) Put a duck in a bowl: There are two toy ducks and two bowls. One of the ducks have to be placed in one of the bowls. The task involves four keyposes.
3) Insert a peg vertically into the hole: The agent needs to detect and grasp a peg, then insert it into a hole that is placed on the ground. The task involves four keyposes.
4) Insert a peg horizontally into the torus: The agent needs to detect and grasp a peg, then insert it into a torus that is placed vertically to the ground. The task involves four keyposes.
5) Put a computer mouse on the pad: There two computer mice and one mousepad. The agent needs to pick one mouse and place it on the pad. The task involves four keyposes.
6) Open the pen: The agent needs to detect a pen that is attached vertically to the table, grasp its lid and pull it to open the pen. The task involves three keyposes.
7) Press the stapler: The agent needs to reach and press a stapler. The task involves two keyposes.
8) Put grapes in the bowl: The scene contains three vines of grapes of different color and one bowl. The agent needs to pick one vine and place it in the bowl. The task involves four keyposes.
9) Sort the rectangle: Between two rectangle cubes there is space for one more. The task comprises detecting the rectangle to be moved and placing it between the others. It involves four keyposes.
10) Stack blocks with the same shape: The scene contains of several blacks, some of which have rectangular and some cylindrical shape. The task is to pick the same-shape blocks and stack them on top of the first one. It involves eight keyposes.
11) Stack cups: The scene contains three cups of different colors. The agent needs to successfully stack them in any order. The task involves eight keyposes.
12) Put block in a triangle on the plate: The agent needs to detect three blocks of the same color and place them inside a plate to form an equilateral triangle. The task involves 12 keyposes.

The above tasks examine different generalization capabilities of 3D Diffuser Actor, for example multimodality in the solution space (5, 8), order of execution (10, 11, 12), precision (3, 4, 6) and high noise/variance in keyposes (1).

We collect 15 demonstrations per task where we record the keyposes, most of which naturally contain noise and multiple modes of human behavior. For example, we pick one of two ducks to put in the bowl, we insert the peg into one of two holes and we put one of three grapes in the bowl, as shown in Figure 5. 3D Diffuser Actor conditions on language descriptions and is trained to predict the next end-effector keypose. During inference, we utilize the BiRRT [31] planner provided by the MoveIt! ROS package [32] to reach the predicted poses. We evaluate 10 episodes for each task are report the success rate.

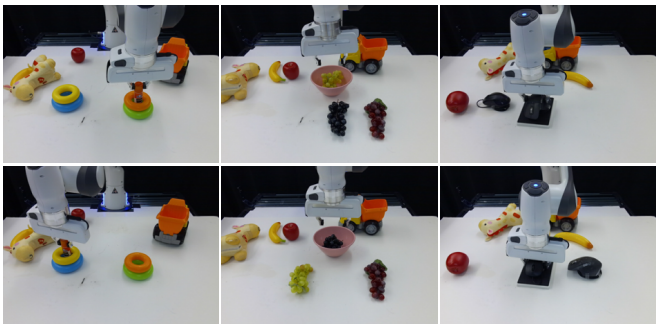


Fig. 5: **Visualized results of our real-world manipulation.**

H. Run time

We measure the latency of our 3D Diffuser Actor on CALVIN in simulation, using an NVIDIA GeForce 2080 Ti graphic card. The wall-clock time of 3D Diffuser Actor is 600 ms. Notably, our model predicts end-effector keyposes sparsely, resulting in better efficiency than methods that predict actions densely at each time step. On CALVIN, on average, it takes 10 keyposes / 60 actions to complete each task using ground-truth trajectories. Our 3D Diffuser Actor can thus perform manipulation efficiently.

I. Limitations

Our framework currently has the following limitations: **1.** Our model conditions on 3D scene representations, which require camera calibration and depth information. **2.** All tasks in RLBench and CALVIN are quasi-static. Extending our method to dynamic tasks and velocity control is a direct avenue of future work.

J. RLBench tasks under PerAct's setup

We provide an explanation of the RLBench tasks and their success conditions under the PerAct setup for self-completeness. All tasks vary the object pose, appearance and semantics, which are not described in the descriptions below. For more details, please refer to the PerAct paper [13].

1) Open a drawer: The cabinet has three drawers (top, middle and bottom). The agent is successful if the target drawer is opened. The task on average involves three keyposes.
2) Slide a block to a colored zone: There is one block and four zones with different colors (red, blue, pink, and yellow). The end-effector must push the block to the zone with the specified color. On average, the task involves approximately 4.7 keyposes
3) Sweep the dust into a dustpan: There are two dustpans of different sizes (short and tall). The agent needs to sweep the dirt into the specified dustpan. The task on average involves 4.6 keyposes.

4) Take the meat off the grill frame: There is chicken leg or steck. The agent needs to take the meat off the grill frame and put it on the side. The task involves 5 keyposes.
5) Turn on the water tap: The water tap has two sides of handle. The agent needs to rotate the specified handle $90°$. The task involves 2 keyposes.
6) Put a block in the drawer: The cabinet has three drawers (top, middle and bottom). There is a block on the cabinet. The agent needs to open and put the block in the target drawer. The task on average involves 12 keyposes.
7) Close a jar: There are two colored jars. The jar colors are sampled from a set of 20 colors. The agent needs to pick up the lid and screw it in the jar with the specified color. The task involves six keyposes.
8) Drag a block with the stick: There is a block, a stick and four colored zones. The zone colors are sampled from a set of 20 colors. The agent is successful if the block is dragged to the specified colored zone with the stick. The task involves six keyposes.
9) Stack blocks: There are 8 colored blocks and 1 green platform. Each four of the 8 blocks share the same color, while differ from the other. The block colors are sampled from a set of 20 colors. The agent needs to stack N blocks of the specified color on the platform. The task involves 14.6 keyposes.
10) Screw a light bulb: There are 2 light bulbs, 2 holders, and 1 lamp stand. The holder colors are sampled from a set of 20 colors. The agent needs to pick up and screw the light bulb in the specified holder. The task involves 7 keyposes.
11) Put the cash in a safe: There is a stack of cash and a safe. The safe has three layers (top, middle and bottom). The agent needs to pick up the cast and put it in the specified layer of the safe. The task involves 5 keyposes.
12) Place a wine bottle on the rack: There is a bottle of wine and a wooden rack. The rack has three slots (left, middle and right). The agent needs to pick up and place the wine at the specified location of the wooden rack. The task involves 5 keyposes.
13) Put groceries in the cardboard: There are 9 YCB objects and a cupboard. The agent needs to grab the specified object and place it in the cupboard. The task involves 5 keyposes.
14) Put a block in the shape sorter: There are 5 blocks of different shapes and a sorter with the corresponding slots. The agent needs to pick up the block with the specified shape and insert it into the slot with the same shape. The task involves 5 keyposes.
15) Push a button: There are 3 buttons, whose colors are sampled from a set of 20 colors. The agent needs to push the colored buttons in the specified sequence. The task involves 3.8 keyposes.
16) Insert a peg: There is 1 square, and 1 spoke platform with three colored spoke. The spoke colors are sampled from a set of 20 colors. The agent needs to pick up the square and put it onto the spoke with the specified color. The task involves 5 keyposes.
17) Stack cups: There are 3 cups. The cup colors are sampled from a set of 20 colors. The agent needs to stack all the other cups on the specified one. The task involves 10 keyposes.
18) Hang cups on the rack: There are 3 mugs and a mug rack. The agent needs to pick up N mugs and place them onto the rack. The task involves 11.5 keyposes.

K. Detailed Model Diagram

We present a more detailed architecture diagram of our 3D Diffuser Actor in Figure 4a. We also show a variant of 3D Diffuser Actor with enhanced language conditioning in Figure 4b, which achieves SOTA results on CALVIN.

The inputs to our network are i) a stream of RGB-D views; ii) a language instruction; iii) proprioception in the form of end-effector's history poses; iv) the current noisy estimates of position and rotation; v) the denoising step t. The images are encoded into visual tokens using a pretrained 2D backbone. The depth values are used to "lift" the multi-view tokens into a 3D feature cloud. The language is encoded into feature tokens using a language backbone. The proprioception is represented as learnable tokens with known 3D locations in the scene. The noisy estimates are fed to linear layers that map them to high-dimensional vectors. The denoising step is fed to an MLP.

The visual tokens cross-attend to the language tokens and get residually updated. The proprioception tokens attend to the visual tokens to contextualize with the scene information. We subsample a number of visual tokens using Farthest Point Sampling (FPS) in order to decrease the computational requirements. The sampled visual tokens, proprioception tokens and noisy position/rotation tokens attend to each other. We modulate the attention using adaptive layer normalization and FiLM [84]. Lastly, the contextualized noisy estimates are fed to MLP to predict the error terms as well as the end-effector's state (open/close).

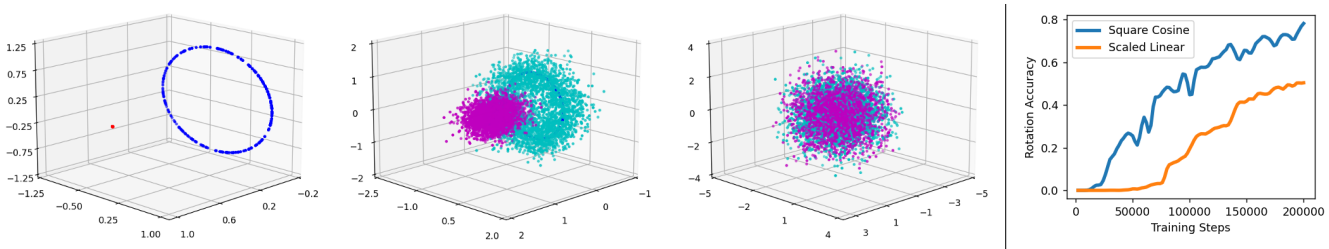
L. Hyper-parameters for experiments

We lift 2D image features to 3D by calculating xyz-coordinates of each image pixel, using the sensed depth and camera parameters. We augment RGB-D observations with random rescaling and cropping. Nearest neighbor interpolation is used for rescaling RGB-D observations. To reduce the memory footprint in our 3D Relative Transformer, we use Farthest Point Sampling to sample a subset of the points in the input 3D feature cloud. We use FiLM [84] to inject conditional input, including the diffusion step and proprioception history, to every attention layer in the model.

Table VI summarizes the hyper-parameters used for training/evaluating our model. On CALVIN we observed that our model overfits the training data, resulting in lower test performance. We use higher $\mathrm{weight\_decay}$ and shorter $\mathrm{total\_epoch}$ on CALVIN compared to RLBench.

	PerAct	GNFactor	CALVIN
Model			
image_size	256	256	200
embedding_dim	120	120	192
camera_views	4	1	2
action_history_length	3	3	3
FPS : % of sampled tokens	20%	20%	33%
diffusion_timestep	100	100	25
noise_scheduler : position	scaled_linear	scaled_linear	scaled_linear
noise_scheduler : rotation	squaredcos	squaredcos	squaredcos
action_space	absolute pose	absolute pose	relative displacement
Training			
batch_size	240	240	1080
learning_rate	$1e^{-4}$	$1e^{-4}$	$3e^{-4}$
weight_decay	$5e^{-4}$	$5e^{-4}$	$5e^{-3}$
total_epochs	$1.6e^{4}$	$8e^{5}$	450
optimizer	Adam	Adam	Adam
loss weight : w_1	30	30	30
loss weight : w_2	10	10	10
Evaluation			
maximal # of keyposes	25	25	60

TABLE VI: **Hyper-parameters of our experiments.** We list the hyper-parameters used for training/evaluating our model on RLBench and CALVIN simulated benchmarks. On RLBench we conduct experiments under two setups: PerAct and GNFactor.



(a) Clean (b) Scaled Linear (c) Square Cosine (d) Rotation Accuracy

Fig. 6: **Visualization of noised rotation based on different schedulers.** We split the 6 DoF rotation representations into 2 three-dimension unit-length vectors, and plot the first/second vector as a point in 3D. The noised counterparts are colorized in magenta/cyan. We visualize the rotation of all keyposes in RLBench *insert_peg* task. From left to right, we visualize the (a) clean rotation, (b) noisy rotation with a scaled-linear scheduler, and (c) that with a square cosine scheduler. Lastly, we compare (d) the denoising performance curve of two noise schedulers. Here, accuracy is defined as the percentage of times the absolute rotation error is lower than a threshold of 0.025. Using the square cosine scheduler helps our model to denoise from the pure noise better.

M. The importance of noise scheduler

We use the following two noise schedulers:

1) a scaled-linear noise scheduler $\beta_t = (\beta_{\max} - \beta_{\min})t + \beta_{\min}$, where $\beta_{\max}, \beta_{\min}$ are hyperparameters, set to 0.02 and 0.0001 in our experiments,
2) a squared cosine noise scheduler $\beta_t = \frac{1-\cos\left(\frac{(t+1)/T+0.008}{1.008} * \frac{\pi}{2}\right)^2}{\cos\left(\frac{t/T+0.008}{1.008} * \frac{\pi}{2}\right)^2}$.

We visualize the clean/noised 6D rotation representations as two three-dimensional unit-length vectors in Figure 6. We plot each vector as a point in the 3D space. We can observe that noised rotation vectors generated by the squared linear scheduler cover the space more completely than those by the scaled linear scheduler.

N. Background on Denoising Diffusion Probabilistic Models

A diffusion model learns to model a probability distribution $p(x)$ by inverting a process that gradually adds noise to a sam-

ple x. For us, x represents a sequence of 3D translations and 3D rotations for the robot's end-effector. The diffusion process is associated with a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$, which defines how much noise is added at each time step. The noisy version of sample x at time t can then be written as $x_t = \sqrt{\bar{\alpha}_t}x + \sqrt{1-\bar{\alpha}_t}\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, is a sample from a Gaussian distribution (with the same dimensionality as x), $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The denoising process is modeled by a neural network $\hat{\epsilon} = \epsilon_\theta(x_t; t)$ that takes as input the noisy sample x_t and the noise level t and tries to predict the noise component ϵ.

Diffusion models can be easily extended to draw samples from a distribution $p(x|\mathbf{c})$ conditioned on input $\mathbf{c}$, which is added as input to the network ϵ_θ. For us $\mathbf{c}$ is the visual scene captured by one or more calibrated RGB-D images, a language instruction, as well as a short history of the robot's end-effector's poses. Given a collection of $\mathcal{D} = \{(x^i, \mathbf{c}^i)\}_{i=1}^{N}$ of end-effector trajectories x^i paired with observation and robot history context $\mathbf{c}^i$, the denoising objective becomes:

$$\mathcal{L}_{\text{diff}}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x^i, \mathbf{c}^i \in \mathcal{D}} ||\epsilon_\theta(\sqrt{\bar{\alpha}_t}x^i + \sqrt{1-\bar{\alpha}_t}\epsilon, \mathbf{c}^i, t) - \epsilon||. \tag{4}$$

This loss corresponds to a reweighted form of the variational lower bound for $\log p(x|\mathbf{c})$ [41].

In order to draw a sample from the learned distribution $p_\theta(x|\mathbf{c})$, we start by drawing a sample $x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Then, we progressively denoise the sample by iterated application of ϵ_θ T times according to a specified sampling schedule [41], [85], which terminates with x_0 sampled from $p_\theta(x)$:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t, \mathbf{c}) \right) + \frac{1-\bar{\alpha}_{t+1}}{1-\bar{\alpha}_t} \beta_t \mathbf{z}, \tag{5}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.