XMoP: Whole-Body Control Policy for Zero-shot Cross-Embodiment Neural Motion Planning

Prabin Kumar Rath and Nakul Gopalan
Arizona State University

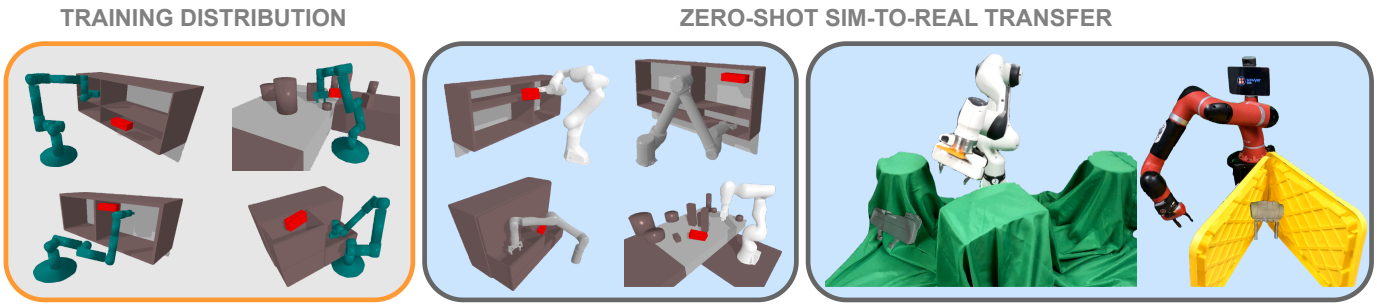


Fig. 1: XMoP learns configuration-space planning for a distribution of synthetic embodiments (*left*) and *zero-shot* transfers the planning behavior to unseen simulated (*center*) and real-world manipulators (*right*).

***Abstract*—Classical manipulator motion planners work across different robot embodiments [1]. However they plan on a pre-specified static environment representation, and are not scalable to unseen dynamic environments. Neural Motion Planners (NMPs) [2] are an appealing alternative to conventional planners as they incorporate different environmental constraints to learn motion policies directly from raw sensor observations. Contemporary state-of-the-art NMPs can successfully plan across different environments [3]. However none of the existing NMPs generalize across robot embodiments. In this paper we propose Cross-Embodiment Motion Policy (XMoP), a neural policy for learning to plan over a distribution of manipulators. XMoP implicitly learns to satisfy kinematic constraints for a distribution of robots and *zero-shot* transfers the planning behavior to unseen robotic manipulators within this distribution. We achieve this generalization by formulating a whole-body control policy that is trained on planning demonstrations from over three million procedurally sampled robotic manipulators in different simulated environments. Despite being completely trained on synthetic embodiments and environments, our policy exhibits strong sim-to-real generalization across manipulators with different kinematic variations and degrees of freedom with a single set of frozen policy parameters. We evaluate XMoP on 7 commercial manipulators and show successful cross-embodiment motion planning, achieving an average 70% success rate on baseline benchmarks. Furthermore, we demonstrate our policy sim-to-real on two unseen manipulators solving novel planning problems across three real-world domains even with dynamic obstacles.**

I. Introduction

Motion planning for robotic manipulators is the task of finding a sequence of robot configurations connecting a start joint state to the goal joint state while respecting joint limits of the robot and avoiding obstacles. Even after decades of research in this domain, real-time motion planning in complex unseen environments is still a challenging problem [4–6].

Classical motion planners either use random sampling to explore the configuration-space (C-space) [7–10] or employ gradient-based optimization methods [11–14] to search for a valid plan. While these algorithms generalize across embodiments, they often demand a non real-time computation budget for generating desired motion behaviors in geometrically complex environments [5, 10]. Furthermore, classical algorithms assume the availability of a pre-computed geometric representation of the robot's workspace for state validation, which hinders their scalability in unseen and dynamic environments. To overcome these limitations, neural planners learn to generate trajectories directly from visual observations [2, 3, 15–18]. However, these policies are individually trained on data from a single manipulator, trading-off the cross-embodiment flexibility offered by classical planners that are agnostic to the robot's morphology.

We identify two fundamental constraints for learning cross-embodiment motion planning. First, different manipulators have varying kinematic properties such as link lengths, as well as diverse morphologies characterized by their degrees of freedom. Each manipulator operates within a particular configuration-subspace bounded by its joint limits. Thus, training a single neural policy to generate actions spanning multiple bounded subspaces renders cross-embodiment C-space policies a challenging task to learn. Second, data for training cross-embodiment policies is difficult to gather as there are only a limited number of embodiments available commercially, which do not fully capture the distribution of possible kinematic variations.

To address the above challenges, we present **Cross**-Embodiment **Mo**tion **P**olicy (**XMoP**), a family of data-driven methods to learn neural policies for cross-embodiment motion planning. Our contributions are outlined as follows:

- Our novel control policy utilizes the robot's physical description (i.e., URDF [19]) and generates C-space plans for a distribution of 6- and 7-DoF manipulators. We demonstrate *zero-shot* generalization of our policy to 7 unseen commercial manipulators, using a set of frozen policy parameters.
- We propose a 3D semantic segmentation-based model

Videos and code are available at https://prabinrath.github.io/xmop.

for perceptual cross-embodiment collision detection that achieves a 98% recall and *zero-shot* transfers to different real-world unseen planning environments.
- Finally, we combine our control policy with the collision model under a model-predictive framework, achieving an average 70% success rate for motion planning using purely visual inputs with manipulators and environments which were never seen during training.

To the best of our knowledge, XMoP is the first configuration space neural planning policy that *zero-shot* transfers to unseen robotic manipulators. We demonstrate sim-to-real transfer along with success rate evaluations on Franka FR3 and Sawyer robots solving novel planning problems in unseen real-world environments.

II. Related Work

Learning to Plan: Previous studies have proposed learning deep priors for informed configuration-space sampling in RRTs [2, 16, 20]. Another line of work have investigated latent space representations and optimization for motion planning [15, 17]. Motion planning has also been formulated as sampling from a distribution of valid trajectories using diffusion models [18, 21, 22]. Recently, behavior cloning has shown success in learning planning policies from synthetic demonstrations [3, 22]. In contrast, Reinforcement Learning (RL) has also been used for training unsupervised motion planning policies [23, 24]. Many of these learning based approaches have been evaluated in toy environments and simulations where the environment was fully observable [20–24], making it unclear how they can be extended for real-world manipulator motion planning. Few methods have been deployed on real robots and show environment generalization [2, 3, 15–18]; however they require a separate policy to be trained for each embodiment. None of the previous works demonstrate motion planning across embodiments, thus lacking the versatility offered by their classical counterparts. In contrast, our method shows *zero-shot* generalization over a distribution of manipulators enabling cross-embodiment neural motion planning.

Cross Embodiment Learning: Prior works have shown embodiment generalization with RL using robot-environment mix-match strategies [25], vector representation of the hardware [26], and latent representation for robots [27]. These methods only work on seen manipulators having fixed DoF whereas our method generalizes across manipulators with different DoF. Recent supervised learning methods showcase policy transfer across robots in task-space using robot specific controllers [28–32]. As these policies do not generalize in robot configuration-space, they are not suitable for motion planning formulation. There has been a surge of interest in meta policies that are trained on procedurally sampled robot morphologies with kinematics and dynamics randomization [33, 34]. These policies have shown *zero-shot* generalization to unseen embodiments in simulation, but have never been deployed on robots in the real world. [35] show *zero-shot* generalization to real robots; however they use a fixed DoF template and do not account for morphology variations. None of the prior methods have demonstrated generalization to commercial robots with morphology and kinematic variations whereas our method shows *zero-shot* sim-to-real generalization across different robotic manipulators in the real world.

III. Methodology

A. Whole-Body Control Formulation

Prior methods for neural motion planning directly predict configuration-space actions that do not generalize across embodiments [2, 3, 15–18]. We formulate XMoP as a Markovian motion dynamics model $f(p_{t+1:t+H}|p_t, g_t)$ that provides the future states of manipulator over a horizon of H steps. The instantaneous state observation for the manipulator is represented as a sequence of rigid-body SE(3) link poses with respect to the robot's base i.e., $p_t \in \mathbb{R}^{D\times4\times4}$, where D is the number of rigid-body links on the manipulator. The goal $g_t \in \mathbb{R}^{4\times4}$ represents the end-effector SE(3) target for motion planning. Our control policy $\pi(a_t|p_t, g_t)$ as shown in Fig. 2, learns to predict link-wise relative SE(3) transformations $a_t = T_{t+1:t+H}$ for reconstructing the whole-body manipulator pose in future time steps. The formulation for the transformation target $T_{t+k} \in \mathbb{R}^{D\times4\times4}$ for $k \in \{1, ..., H\}$ is shown in eq. 1.

$$T_{t+k}p_t = p_{t+k} \implies T_{t+k} = p_{t+k}(p_t)^{-1} \tag{1}$$

where link poses p_t are obtained using the manipulator's forward kinematics function $\phi(j_t)$, with $j_t \in \mathbb{R}^{\text{DoF}}$ as the instantaneous configuration-space observation. The C-space action in future time step j_{t+k} is retrieved from the predicted whole-body pose $\hat{p}_{t+k} = \hat{T}_{t+k}p_t$ by solving for whole-body IK using the following constrained optimization procedure:

$$\min_{j_{t+k}} \quad \|\hat{p}_{t+k} - \phi(j_{t+k})\|, \quad \text{s.t.} \quad j_L < j_{t+k} < j_U \tag{2}$$

where j_L and j_U are the lower and upper joint limits of the manipulator. The above optimization objective is non-convex, and hence a close initial guess is required for convergence. We address this issue by collecting dense planning demonstrations with a maximum per-joint deviation of 0.05 rad. Thus, making the instantaneous observation j_t to be an initial guess that lies within the close neighbourhood of j_{t+k}. In practice, we also employ multiple retries with random initial joint states to handle redundancy and singularities in manipulators.

B. Pose Transformation Policy

We formulate the whole-body control policy $\pi_\theta(a_t|p_t, g_t)$ as a stochastic Transformer diffusion policy [37] parameterized by θ that predicts a batch of possible future trajectories for model predictive control. While training, the noise prediction model ϵ_θ takes the noisy sample a_t^τ, which is obtained by applying the forward diffusion process to $a_t^0 = T_{t+1:t+H}$, where τ is the diffusion step. We use the noisy sample a_t^τ as link transform query, which is passed to the diffusion model, as shown in Fig. 2. Additionally, we also pass $c = (p_t, g_t)$ for observation and goal conditioning. For step conditioning, we follow the adaptive layer normalization strategy proposed in Diffusion Transformers [38]. We train the noise prediction

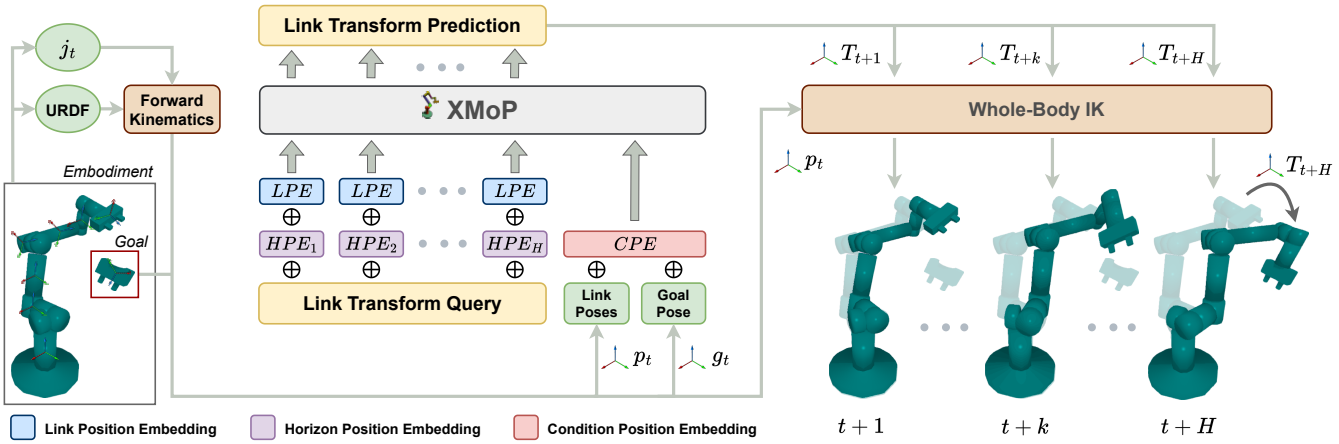


Fig. 2: XMoP perceives the embodiment state as a sequence of whole-body SE(3) link poses p_t and predicts link-wise pose transformations $T_{t+1:t+H}$ over a horizon H to move the end-effector towards the goal pose g_t. We use a Transformer [36] base policy architecture, that operates on an input pose sequence $(p_t,\ g_t)$ and uses self-attention to convert the query tokens into a sequence of link-wise relative pose transformations. Contextual information is provided to the Transformer using three types of position embeddings: (1) LPE is a fixed set of sinusoidal position embeddings that are repeated over the horizon providing kinematic chain awareness for every horizon step; (2) HPE and (3) CPE, are learned position embeddings providing awareness for horizon and input, respectively. Additionally, we use novel attention masking strategies within the Transformer for cross-embodiment adaptation. The predicted link-wise transformations are applied to the instantaneous link poses to reconstruct the future whole-body pose of the manipulator. This target pose is achieved using a whole-body IK procedure, which retrieves the configuration-space joint state within the bounds specified in the manipulator's URDF.

model using mean square error loss as shown in eq. 3 which minimizes the variational lower bound of the KL-divergence between the original data distribution and the DDPM [39] distribution.

$$\mathcal{L}_{xmop} = \|\epsilon_\theta(a_t^\tau, c, \tau) - \epsilon\|_2^2, \quad \epsilon \in \mathcal{N}(0, I) \tag{3}$$

We convert the input pose matrices in p_t to compact 9D representations $r = (\vec{o}, \vec{x}, \vec{y})$, where $\vec{o} \in \mathbb{R}^3$ is the 3D translation component and $(\vec{x}, \vec{y}) \in \mathbb{R}^6$ is the 6D rotation component in SE(3) [40]. The model predicts the relative pose transformations in the same 9D representation space which is converted back to homogeneous matrices using Gram–Schmidt orthogonal decomposition [40]. Our policy utilizes the Transformer [36] model as the underlying backbone which expects input in a sequential format. On that aspect, we emphasize on four key design decisions in our policy:

1) **SE(3) Proprioception**: The embodiment state is provided to and queried from the policy as a sequence of SE(3) *pose-tokens*, allowing the policy to learn motion synergies between rigid-body links.
2) **Kinematic Masking**: We introduce an inductive bias for kinematics by restricting attention to parent or ancestor links at the current horizon step, and to the same link at both the current and previous horizon steps.
3) **Morphology Adaptation**: To enable learning across different morphologies, we mask out the *pose-tokens* for unavailable links. For example, we use D *pose-tokens* per horizon step, with $D = 8$ for both 6- and 7-DoF robots (including the base link *pose-token* that is perpetually set to identity). For a 6-DoF robot however, we mask out one *pose-token* and pass $\vec{0}$ to account for the missing link.
4) **Link-Horizon Position Embedding**: Contextual information is provided to the Transformer using position embeddings for query and input *pose-tokens*. Fig. 2 shows the position embedding scheme used in XMoP.

Prior works have used link-wise tokens [33], temporal state inputs [21, 34], action diffusion [18, 21, 37], and SE(3) pose observations [41, 42] for learning and planning applications. We combine these ideas with our whole-body pose transformation method to learn neural policy for cross-embodiment motion planning.

C. Collision-Free Motion Synthesis

We formulate collision detection as a semantic segmentation problem for identifying the links of the robot that are in collision given a pointcloud observation of the workspace. Our semantic collision detection model **XCoD** : $\mathbb{R}^4 \rightarrow \mathbb{R}^2$ takes segmented pointcloud of the workspace as input where each point consists of 3D spatial coordinates, and a semantic label. These points are uniformly sampled from the surface of individual links (URDF mesh files) of the manipulator and scene obstacles for the future time steps as predicted by our control policy π_θ. We assign unique semantic labels to each link of the robot, while using a separate reserved label for all obstacles. For training the collision model, we utilize point-wise binary label y, where collision-free link points are assigned a training label of 0, whereas link points in collision are assigned a training label of 1. Fig. 3 shows planning scenes and corresponding colorized pointclouds highlighting point-wise training labels. We utilize a Point Transformer V3 (PTv3) [43]

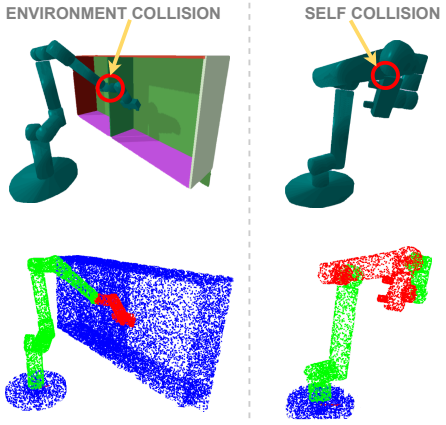


Fig. 3: Point-wise training labels for XCoD. Collision, Not Collision, Obstacles.

model for the semantic segmentation task and train it using cross-entropy loss along with an additional surrogate Lovász hinge loss, that has shown to improve semantic segmentation performance in prior work [44].

We use XCoD to assign scores for a batch of trajectories predicted by XMoP and choose the future trajectory with the fewest collisions for locally reactive planning. Eq. 4 shows the formulation of the proposed Model Predictive Control (MPC) method where N is the number of surface points sampled from the manipulator, B is the MPC batch size, and $\hat{y}$ is the per-point collision logit from XCoD.

$$\begin{aligned} a_t^* &= a_t^{(q)}, \quad q = \arg\min[s_1, s_2, \ldots, s_B], \\ s &= \frac{1}{HN} \sum_{h=1}^{H} \sum_{i=1}^{N} \arg\max \hat{y}_i \end{aligned} \tag{4}$$

Similar to Diffusion Policy [37], we set the prediction horizon $H_p = 16$, while the execution horizon H_a is determined on the fly based on the collision scores from XCoD during MPC rollouts. We empirically found that H_a between 2 to 4 with MPC batch size of $B = 16$ works best for both simulation and real-world experiments.

D. Data Generation and Training

Kinematic Templates: Synthetic manipulators are represented with open kinematic chains connecting a series of rigid-body links. We design these links using 3 axis-aligned cylinders forming a rigid-body template. Each link template is parameterized with the following information: (1) length of the cylinders, (2) radius of the cylinders, and (3) constraints for the joint that connects the link to the preceding link. We follow the design pattern of two commercially available robots: (1) 6-DoF UR [45] (2) 7-DoF Sawyer [46]. Fig. 1 (*left*) shows composed manipulators sampled from our synthetic embodiment distribution by randomizing the parameters for constituent link templates. We adopt the 3.27 million synthetic planning problems from the MπNets dataset [3] and generate demonstration data by sampling a unique embodiment for each problem and solving it using the AIT* [10] motion planner.

Data Augmentation: During training, we randomize the position and orientation of the link frames for pose computation by uniformly choosing cylinders from the constituent link templates. Fig. 4 shows two possible frame sequences for a sampled robot. With the frame augmentation technique, number of possible sequences for a single manipulator is 3^D which promotes cross-embodiment generalization during training. Additionally, we adopt the data augmentation strategies from MπNets [3] and add Gaussian noise to joint and pointcloud observations during training while also sampling unique pointclouds during each training iteration to promote sim-to-real visual robustness.

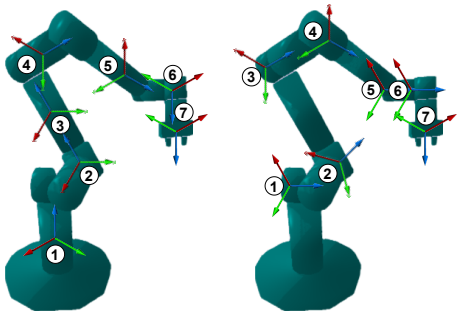


Fig. 4: Two possible sets of frame sequences for a sampled 7-DoF manipulator. Link frame considered for *pose-token*.

IV. Experiments and Results

It should be noted that none of the robots used in our benchmark or real-world experiments were part of XMoP's training dataset. All of our results (simulation and real-world) are *zero-shot* evaluations using a single set of frozen XMoP (38.7M) and XCoD (2.5M) checkpoints, which were completely trained on synthetic planning demonstration data.

A. Simulation Benchmark Evaluations

We evaluate XMoP on 7 different robotic manipulators from 5 commercial manufacturers: Franka Panda, Rethink Sawyer, Kuka IIWA, Kinova Gen3 6-DoF, Kinova Gen3 7-DoF, Universal Robots UR5, and Universal Robots UR10. For each manipulator we use a set of 500 novel problems from the MπNets [3] test distribution, ensuring that valid collision-free IK solutions exist for both start and goal end-effector targets. Policy rollouts are terminated if the manipulator's end-effector reaches the goal or a maximum of 200 rollout steps are exhausted. We consider a goal to be reached when the $L2$ norm of the 9D pose (III-B) difference between end-effector and goal is less than a pre-specified threshold of 0.01.

Baseline Planners: We compare our policy against the upper performance threshold of AIT* [10] planner that has access to an oracle collision checker from PyBullet. We also evaluate generalization capabilities of our learned collision model XCoD by combining it with the AIT* baseline. This hybrid planner utilizes the XCoD model for collision queries.

We utilize the following quantitative metrics to evaluate the planning performance: (1) *Success Rate (SR)* - A trajectory is successful if the final end-effector position is within 1 cm

Embodiment	XMoP+XCoD			AIT*+XCoD			AIT*+PyBullet		
	SR[%] ↑	PL ↓	ST[s] ↓	SR[%] ↑	PL ↓	ST[s] ↓	SR[%] ↑	PL ↓	ST[s] ↓
Panda	71.8	4.6 ± 4.7	49.8 ± 65.8	86.0	3.6 ± 2.3	39.7 ± 27.3	94.4	2.9 ± 1.5	4.0 ± 0.3
Sawyer	70.8	4.8 ± 5.3	42.9 ± 53.6	90.4	3.3 ± 2.8	34.6 ± 27.1	92.4	1.9 ± 0.9	3.9 ± 0.5
IIWA	71.0	5.1 ± 5.6	38.3 ± 52.5	87.6	2.8 ± 2.1	32.3 ± 21.2	93.4	2.1 ± 1.0	3.9 ± 0.4
Gen3 6-DoF	67.6	4.7 ± 6.0	51.8 ± 70.9	71.0	2.5 ± 2.6	24.2 ± 11.0	92.4	2.0 ± 0.9	3.9 ± 0.5
Gen3 7-DoF	78.2	5.5 ± 5.9	44.0 ± 53.0	88.4	3.3 ± 2.6	35.0 ± 22.6	94.2	2.2 ± 1.2	3.9 ± 0.4
UR5	70.8	3.1 ± 3.3	42.2 ± 71.2	80.8	2.6 ± 1.9	31.0 ± 20.7	88.8	2.1 ± 1.5	3.9 ± 0.4
UR10	67.4	3.1 ± 3.4	31.5 ± 52.0	72.6	2.9 ± 2.6	33.2 ± 24.8	92.2	2.1 ± 1.2	3.8 ± 0.6

TABLE I: Benchmark results. There are no neural planning baselines that work across multiple embodiments (see IV-D). Hence, we compare to the upper baselines of AIT*+PyBullet which we expect to perform better as it has access to the ground truth obstacle geometry information. In contrast, XMoP plans directly from visual inputs. See IV-C for discussion on the hybrid baseline. All inference are on RTX A5000 GPU.

Robot	Planning Success Rate (%)		
	Unstructured Obstacles	Wall Hopping	Bin-to-Bin
Franka FR3	70.0	80.0	70.0
Rethink Sawyer	80.0	80.0	50.0

TABLE II: Sim-to-real results on three real-world domains. The average duration of these experiments including planning and rollout was 43.77 ± 22.20 seconds. All inference are on RTX 3090 GPU.

and orientation is within 5° of the goal, with no collisions or joint limit violations. (2) *Path Length (PL)*: Sum of $L2$ norm between consecutive configuration-space way points. (3) *Solution Time (ST)*: Total time elapsed to generate a successful trajectory. Table I shows the benchmark results.

B. Real-world Evaluations

Prior works on neural planning do not have any evaluation domains for real-world, hence we created three novel domains for quantitative evaluations and statistics.

- **Unstructured Obstacles Domain**: A domain with random obstacles covered with a green screen to demonstrate planing abilities in unstructured and cluttered environments.
- **Wall Hopping Domain**: A domain with structured shapes acting as obstacles that occupy a significant amount of space in front of the robot.
- **Bin-to-Bin Domain**: A task-oriented domain where the robot plans its movement from one bin to another, similar to tasks it might perform in a warehouse environment.

Fig. 5 shows the three real-world domains. We used mono-color obstacles and segmented them from a calibrated depth camera to extract the obstacle point cloud. As discussed in III-C, a segmented pointcloud of the robot was augmented into the scene for generating the input for XCoD. We manually assigned end-effector goal poses evenly distributed around the obstacles for each of our experiments. A rollout was considered successful if the manipulator reached the goal without touching any obstacles or itself. All domains had obstacles with dimensions equal to or larger than the links of the manipulator. We conducted 10 experiments for each domain with two different robots Franka FR3 and Rethink Sawyer. XMoP achieved an overall success rate of 71.6%. The success rate of our experiments is shown in Table II.

C. Result Analysis and Discussion

Plan Optimality: Table I shows that the AIT* upper baseline generates approximately 50% more optimal plans and is 10 times faster compared to the XMoP planner. However, this comes with the assumption of privileged obstacle information being available for collision detection, thus making it difficult to deploy in unseen real-world environments. We believe our work is a preliminary step in the direction of learning neural planners independent of the embodiment, thus bringing their capabilities one step closer to classical motion planners.

Collision Detection: As shown in Table I, the hybrid AIT*+XCoD planner achieves an average success rate of 82.4%, which is within 10% of the oracle baseline, demonstrating the effectiveness of our learned collision detection model. However, this hybrid planner fails in the real world, as binary collision queries using XCoD are inaccurate with noisy point clouds. In contrast, XMoP's MPC design allows for more robust real-world deployment. XCoD is a generalized collision model that enables collision-free rollouts for MPC-based policies, such as XMoP. Although XCoD is trained on fully observable synthetic pointclouds, we found that the backbone PTv3 [43] model is effective in handling partial pointclouds captured from real-world depth cameras.

Computational Efficiency: The current SOTA neural planner MπNets [3] needs two weeks of training for a single manipulator and requires a compute heavy data generation effort. In contrast, XMoP needs *one-time* data generation and takes two days to train on a single RTX A5000 GPU.

***Zero-shot* Generalization**: Table I demonstrates XMoP's ability to plan for manipulators with novel designs that were unseen during training. XMoP exploits the fact that motion behavior is characterized by the whole-body pose of the embodiment [47, 48]. For similar whole-body poses, different manipulators might have contrasting joint configurations, but the link poses are relatively closer in SE(3). Similarly, configuration-space actions for different manipulators are dependent on their morphology, but link pose transformations are similar for individual manipulator links.

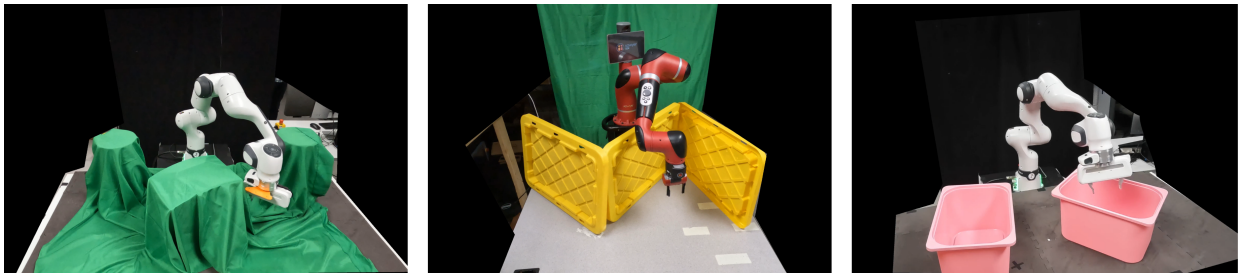


Fig. 5: XMoP successfully plans for two unseen robotic manipulators in three real-world domains. Unstructured Obstacles (*left*), Wall Hopping (*center*), and Bin-to-Bin (*right*). Videos of policy rollouts are available at https://prabinrath.github.io/xmop.

Sim-to-real Experiments: We found the Bin-to-Bin domain particularly challenging, as the policy needed to accurately plan the approach angle to avoid collisions with the bin walls (see Table II). However, this experiment demonstrates a practical use case where XMoP can be applied for *zero-shot* task execution in unseen real-world environments.

Kinematic Constraints Satisfaction: Our policy predicts link-wise transformations that obey kinematics constraints across different manipulators. Our hypothesis is that it learns a correlation between *pose-token* sequence and the distribution of kinematically feasible whole-body transformations. We are actively investigating this property of XMoP and will provide a more detailed analysis in our future work.

Limitations: XMoP's performance is limited by the quality of synthetic training data, leading to struggles with highly out-of-distribution (OOD) planning setups. Moreover, XCoD collision checking is slow, accounting for approximately 90% of the solution time (see Table I), which could be improved by using better semantic segmentation models in the future.

D. Failed Baselines and Ablations

MπNets Baseline: Table III shows the benchmark results for the MπNets model trained on demonstration data from the Franka Panda robot. We also trained a similar model on XMoP training data, which overfitted to the geometric design of synthetic robots and could not transfer *zero-shot* to any of the unseen robotic manipulators (resulting in 0% SR). On the contrary, XMoP achieves an average SR of 71.1% across 7 unseen robots as shown in Table I.

Baseline	**Planning Success Rate (%)**						
	Panda	Sawyer	IIWA	Gen3 6-DoF	Gen3 7-DoF	UR5	UR10
MπNets	89.6	0	0	0	0	0	0

TABLE III: The SOTA neural planner MπNets pre-trained on single embodiment data fails to generalize across unseen robots.

ACT Baseline: We evaluated the SOTA C-space behavior cloning policy ACT [49] for cross-embodiment planning. This policy was provided with privileged embodiment information including link lengths, link radius, and joint limits. However, it failed to control (0% SR) both unseen synthetic and commercial robots. In contrast to prior works discussed in II, XMoP control policies are not trained on any embodiment-specific information, thus showing that such information is not absolutely necessary for cross-embodiment generalization.

Ablation Studies: We ablated the XMoP policy to understand the importance of each of our design decisions.

- **w/o SE(3) Proprioception**: This is the most critical component for cross-embodiment generalization without which the SR drops to 0%.
- **w/o Kinematic Masking and Morphology Adaptation**: We completely removed the masking scheme, thus allowing every *pose-token* to attend every other *pose-token*. Compared to XMoP, the average SR dropped by 4.4%.
- **w/o Link Horizon Position Embedding**: We replaced the proposed position embedding scheme with learned position embeddings from the Transformer paper [36]. The average SR compared to XMoP dropped by 1.2%.
- **w/o Frame Augmentation**: We did not use the frame augmentation technique from III-D and instead used the pose of the first cylinder in each link template. Without frame augmentation, XMoP's average SR dropped by 68.1%.
- **Scale of Data**: We trained XMoP with reduced dataset sizes i.e., 1M and 2M demonstrations which resulted in 51.8% and 23.2% drop in average SR respectively.

In summary, we observed that the masking scheme and position embedding scheme do not significantly contribute to the success of XMoP. However, SE(3) proprioception, frame augmentation, and higher scale of data are critical for cross-embodiment generalization.

V. Conclusion

In this paper, we presented XMoP, a novel configuration-space neural motion policy that solves novel planning problems *zero-shot* for unseen robotic manipulators which has not been achieved by any prior robot learning algorithm. We formulated C-space control as a link-wise SE(3) pose transformation method, showcasing its scalability for data-driven policy learning. We used fully synthetic data to train models for motion planning and collision detection while demonstrating strong sim-to-real generalization with a 70% success rate. Our work demonstrates for the first time that C-space motion policies can be learned without embodiment bias and that these learned behaviors can be transferred to novel unseen embodiments in a *zero-shot* manner. We hope our work will enable more generalized robot foundation models that are capable of generating whole-body motion for a diversity of robots. Our work is a preliminary step towards such desirable generalization in learning robot behaviors.

References

[1] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387–424, 2022.

[2] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.

[3] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox, "Motion policy networks," in *Conference on Robot Learning*. PMLR, 2023, pp. 967–977.

[4] S. H. Tang, W. Khaksar, N. Ismail, and M. Ariffin, "A review on robot motion planning approaches," *Pertanika Journal of Science and Technology*, vol. 20, no. 1, pp. 15–29, 2012.

[5] L. Petrović, "Motion planning in high-dimensional spaces," *arXiv preprint arXiv:1806.07457*, 2018.

[6] T. Sandakalum and M. H. Ang Jr, "Motion planning for mobile manipulators—a systematic review," *Machines*, vol. 10, no. 2, p. 97, 2022.

[7] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] 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.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[10] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*) and effort informed trees (eit*): Asymmetric bidirectional sampling-based path planning," *The International Journal of Robotics Research*, vol. 41, no. 4, pp. 390–417, 2022.

[11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 489–494.

[12] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[13] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Berlin, Germany, 2013, pp. 1–10.

[14] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs." in *Robotics: Science and Systems*, vol. 12, no. 4, 2016.

[15] J. Huh, V. Isler, and D. D. Lee, "Cost-to-go function generating networks for high dimensional motion planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8480–8486.

[16] J. J. Johnson, A. H. Qureshi, and M. C. Yip, "Learning sampling dictionaries for efficient and generalizable robot motion planning with transformers," *IEEE Robotics and Automation Letters*, 2023.

[17] J. Yamada, C.-M. Hung, J. Collins, I. Havoutis, and I. Posner, "Leveraging scene embeddings for gradient-based motion planning in latent space," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5674–5680.

[18] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 1916–1923.

[19] D. Tola and P. Corke, "Understanding urdf: A survey based on user experience," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–7.

[20] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

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

[22] K. Saha, V. Mandadi, J. Reddy, A. Srikanth, A. Agarwal, B. Sen, A. Singh, and M. Krishna, "Edmp: Ensemble-of-costs-guided diffusion for motion planning," 9 2023. [Online]. Available: http://arxiv.org/abs/2309.11414

[23] T. Jurgenson and A. Tamar, "Harnessing reinforcement learning for neural motion planning," *Robotics: Science and Systems XV*, 2019.

[24] R. Strudel, R. G. Pinel, J. Carpentier, J.-P. Laumond, I. Laptev, and C. Schmid, "Learning obstacle representations for neural motion planning," in *Conference on Robot Learning*. PMLR, 2021, pp. 355–364.

[25] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2169–2176.

[26] T. Chen, A. Murali, and A. Gupta, "Hardware conditioned policies for multi-robot transfer learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[27] A. Ghadirzadeh, X. Chen, P. Poklukar, C. Finn, M. Björkman, and D. Kragic, "Bayesian meta-learning for few-shot policy adaptation across robotic platforms. in 2021 ieee," in *RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1274–1280.

[28] J. H. Yang, D. Sadigh, and C. Finn, "Polybot: Training

one policy across robots while embracing variability," in *Conference on Robot Learning*. PMLR, 2023, pp. 2955–2974.

[29] J. Yang, C. Glossop, A. Bhorkar, D. Shah, Q. Vuong, C. Finn, D. Sadigh, and S. Levine, "Pushing the limits of cross-embodiment learning for manipulation and navigation," *arXiv preprint arXiv:2402.19432*, 2024.

[30] D. Shah, A. Sridhar, N. Dashora, K. Stachowicz, K. Black, N. Hirose, and S. Levine, "Vint: A foundation model for visual navigation," in *Conference on Robot Learning*. PMLR, 2023, pp. 711–733.

[31] G. Salhotra, I. Liu, C. Arthur, and G. Sukhatme, "Bridging action space mismatch in learning from demonstrations," *arXiv preprint arXiv:2304.03833*, 2023.

[32] L. Y. Chen, K. Hari, K. Dharmarajan, C. Xu, Q. Vuong, and K. Goldberg, "Mirage: Cross-embodiment zero-shot policy transfer with cross-painting," *arXiv preprint arXiv:2402.19249*, 2024.

[33] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei, "Metamorph: Learning universal controllers with transformers," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=Opmqtk_GvYL

[34] I. Schubert, J. Zhang, J. Bruce, S. Bechtle, E. Parisotto, M. Riedmiller, J. T. Springenberg, A. Byravan, L. Hasenclever, and N. Heess, "A generalist dynamics model for control," *arXiv preprint arXiv:2305.10912*, 2023.

[35] G. Feng, H. Zhang, Z. Li, X. B. Peng, B. Basireddy, L. Yue, Z. Song, L. Yang, Y. Liu, K. Sreenath *et al.*, "Genloco: Generalized locomotion controllers for quadrupedal robots," in *Conference on Robot Learning*. PMLR, 2023, pp. 1893–1903.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[37] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[38] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4195–4205.

[39] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[40] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5745–5753.

[41] W. Liu, T. Hermans, S. Chernova, and C. Paxton, "Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects," in *Workshop on Language and Robotics at CoRL 2022*, 2022.

[42] A. Simeonov, A. Goyal, L. Manuelli, Y.-C. Lin, A. Sarmiento, A. R. Garcia, P. Agrawal, and D. Fox, "Shelving, stacking, hanging: Relational pose diffusion for multi-modal rearrangement," in *Conference on Robot Learning*. PMLR, 2023, pp. 2030–2069.

[43] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao, "Point transformer v3: Simpler faster stronger," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4840–4851.

[44] M. Berman, A. R. Triki, and M. B. Blaschko, "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4413–4421.

[45] "Universal robots," accessed: 2024-05-27. [Online]. Available: https://www.universal-robots.com/

[46] "Rethink robotics," accessed: 2024-05-27. [Online]. Available: https://www.rethinkrobotics.com/

[47] M. Arduengo, A. Arduengo, A. Colomé, J. Lobo-Prat, and C. Torras, "Human to robot whole-body motion transfer," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2021, pp. 299–305.

[48] X. Cheng, Y. Ji, J. Chen, R. Yang, G. Yang, and X. Wang, "Expressive whole-body control for humanoid robots," *arXiv preprint arXiv:2402.16796*, 2024.

[49] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *arXiv preprint arXiv:2304.13705*, 2023.

[50] A. Demir, F. J. Prael III, and B. Kiziltan, "Se (3)-invariant multiparameter persistent homology for chiral-sensitive molecular property prediction," in *NeurIPS 2023 AI for Science Workshop*, 2023.

[51] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[52] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin, "The franka emika robot: A reference platform for robotics research and education," *IEEE Robotics & Automation Magazine*, vol. 29, no. 2, pp. 46–64, 2022.

[53] D. Rakita, H. Shi, B. Mutlu, and M. Gleicher, "Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9995–10 001.

[54] B. K. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Josa a*, vol. 4, no. 4, pp. 629–642, 1987.

[55] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, T. S. Lim, and P. Wong, "Collision detection using axis aligned

bounding boxes," *Simulations, Serious Games and Their Applications*, pp. 1–14, 2014.

[56] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[57] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

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

[59] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International conference on machine learning*. PMLR, 2021, pp. 8162–8171.

[60] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[61] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

APPENDIX

A. Embodiment Sampling with Kinematic Templates

Link Template: We design 3-Cylinder and 2-Cylinder templates to represent rigid-body links for our synthetic manipulators. These cylinders sequentially extrude along the 3D axis as shown in Fig. 6(a). Each link is parameterized with six numbers leading to a vector representation for link i as $L_i = (p, l_x, l_y, l_z, r, cf)$ where $i \in \{1, 2, \ldots, D\}$ and D is the number of rigid-body links on the manipulator. The definition of the parameters are as follows:

- p: A sequence of three characters representing connection pattern of cylinders within the link. It has 3^3 possible permutations i.e., $\{\texttt{xyz}, \texttt{yxz}, \texttt{zxy}, \texttt{yzy}, \ldots\}$. Each pattern maps to a number that is referenced in L_i. For example, the pattern `zxy` means the first cylinder extrudes along z axis, the second cylinder extrudes along x axis, and the third cylinder extrudes along y axis.
- l_x, l_y, l_z: Length of the cylinders along x, y, and z axis respectively.
- r: Radius of all the three cylinders in the link template.
- cf: Represents chiral flip [50] with a binary value of 0 or 1. When the value is 1, a mirror link is created instead of a regular link. Only the last cylinder in the connection pattern is flipped for chirality. A visual representation of a chiral link pair is shown in Fig. 6(a) and Fig. 6(b).

For a 2-Cylinder template as shown in Fig. 6(d), we set the length of the middle link to 0.

End-Effector Template: The end-effector template is represented with a vector of three numbers $E = (h, r, s, -1, -1, -1)$ where padding -1 is used to make the length consistent with the link templates. The end-effector is represented using a 3-Cuboid mesh on top of a base cylinder as shown in Fig. 6(e). The definition of the parameters are h: height of the base cylinder, r: radius of the base cylinder, and s: scaling factor for the 3-Cuboid mesh.

Joint Constraints: A pair of link templates L_{i-1} and L_i can be chained together only if the third character in the pattern of link $i-1$ matches with the first character in the pattern of link i. Such a chain is connected with a revolute joint parameterized using a vector of two numbers $J_i = (ll, ul)$. The definition of the parameters are ll: lower joint limit, and ul: upper joint limit.

Kinematic Template: We hypothesize that these basic templates can be used to compose arbitrary manipulators that are kinematically valid and lie within the distribution of commercial manipulators. A synthetic manipulator is represented using matrix `KT` of shape $(8 \times D)$ as shown in eq. 5.

$$\texttt{KT} = \begin{bmatrix} L_1 & L_2 & \cdots & L_{D-1} & E \\ J_1 & J_2 & \cdots & J_{D-1} & J_D \end{bmatrix} \tag{5}$$

We fix the patterns p and chiral flips cf to match with that of the commercial robots Sawyer, and UR. Rest of the parameters in the kinematic template are sampled using one of the following strategies:

- `normal`: Parameters are sampled from a normal distribution centered at the estimated parameters from Sawyer, and UR robots. The variance is hard specified for individual parameters to ensure that the generated robots are valid.
- `uniform`: Parameters are sampled from a uniform distribution with hard conditions to ensure that the generated robots are valid.

Fig. 7 shows sampled manipulators from the distribution of Sawyer, and UR robots. We use ROS [51] `xacro` to process kinematic templates and generate URDF [19] files on the fly for data generation.

B. Synthetic Data Generation

Planning Problems: We adopt planning problems from the MπNets [3] dataset, which contains 3.27 million motion plans for the Franka Panda [52] robot across three different indoor environments. We use the terminal joint states of these motion plans to extract start and goal end-effector poses of the Panda robot using forward kinematics function ϕ. Then, we attempt collision-free IK to find valid start and goal joint configurations, thereby defining novel planning problems with our synthetic manipulators. Similar to [53], we formulate IK as a quadratic optimization problem as shown in eq. 6.

$$\min_j f(j) \quad \text{s.t. } j_L \leq j \leq j_U \tag{6}$$

where j is the configuration-space candidate for optimization, and j_L and j_U are lower and upper joint limits of the manipulator. We define the cost function $f(j)$ as sum of the following objectives:

- *Position Cost*: Position $x, x_g \in \mathbb{R}^3$ are current and goal positions of the end-effector with respect to the manipulator's base frame.

$$x = \phi(j).ee.pos$$
$$c_{pos}(x, x_g) = -\exp\left(-\frac{\|x - x_g\|^2}{0.08}\right) + 25 * (\|x - x_g\|)^4 \tag{7}$$

- *Orientation Cost*: Quaternion $q, q_g \in \mathbb{R}^4$ are current and goal orientations of the end-effector with respect to the manipulator's base frame. Where $dist(q, q_g)$ is the absolute quaternion distance [54] accounting for the sign ambiguity.

$$q = \phi(j).ee.rot$$
$$c_{rot}(q, q_g) = -\exp\left(-\frac{dist(q, q_g)^2}{0.08}\right) + 25 * (dist(q, q_g))^4 \tag{8}$$

- *Collision Cost*: We use PyBullet to obtain collision distances from the links of the manipulator within an AABB [55] collision radius of 0.1 m for self-collision and 0.3 m for environment collision. These distances are then passed into an exponential function, $g(x)$, for computing the net collision cost.

$$g(x) = \begin{cases} 1.8^{(-x+0.01)} & \text{if } (x - 0.01) < 0, \\ 0 & \text{otherwise,} \end{cases}$$
$$c_{coll}(j) = \sum_{x \in \text{AABB}(j)} g(x) \tag{9}$$

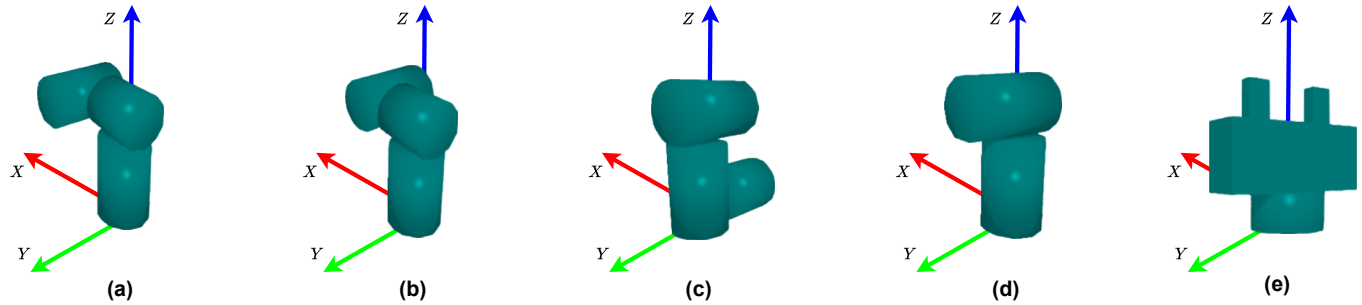


Fig. 6: Example of random templates that are composed to sample synthetic embodiments for data generation. (a) 3-Cylinder `zxy` template with no chiral flip, (b) 3-Cylinder `zxy` template with chiral flip, (c) 3-Cylinder `yzy` template with chiral flip, (d) 2-Cylinder `zxy` template with no chiral flip, (e) End-effector template.

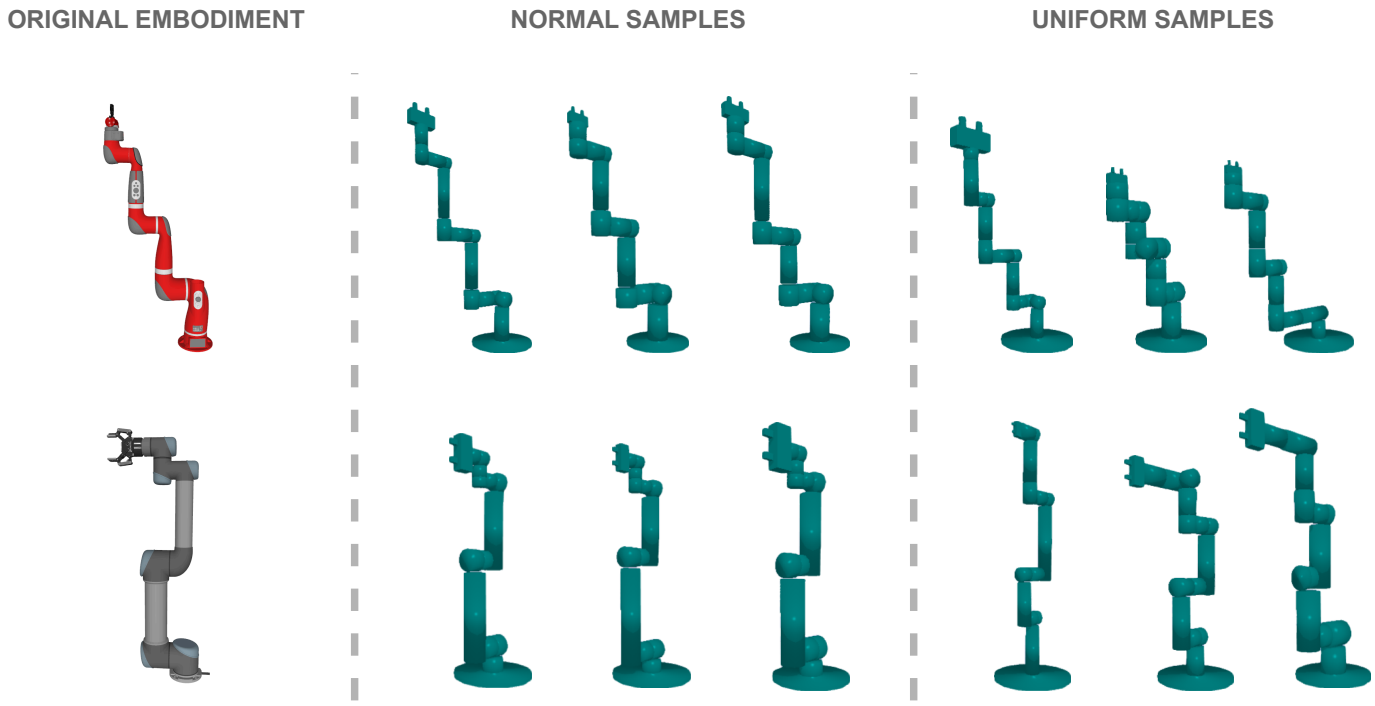


Fig. 7: Sampled robots from our training distribution. XMoP is trained on planning demonstrations data from such synthetic embodiments and *zero-shot* transfers to unseen manipulators in both simulation and real-world.

For eq. 7 and 8, we use the groove objective from [53] with default parameters from the paper; whereas for eq. 9 we empirically found that the exponential objective provides faster optimization convergence. We use the SLSQP implementation from `scipy` [56] and attempt IK five times with random resets if the optimizer gets stuck in a local minima. Each optimization attempt is capped to a maximum of 500 iterations.

Planning Data Generation: We employ the SOTA sampling-based planner AIT* [10] from OMPL [57] along with PyBullet's oracle collision checker to generate a valid solution trajectory. For demonstration optimality, we enforce a minimum search time of 15 seconds with a maximum 20 seconds timeout while optimizing over the path length objective. Additionally we set a maximum allowed path length (IV-A) threshold of 10 to avoid generating highly non-optimal trajectories. To aid the non-convex optimization as discussed in III-A, we re-time the solution path using more granular collision checks along a B-spline interpolation of the generated trajectory allowing a maximum per joint velocity of 0.05 rad/s. We resample a new embodiment until we solve a given environment, thus ensuring valid motion plans for all the 3.27 million problems in MπNets dataset. The final motion plan along with the kinematic template `KT` are saved to the disk for training purposes. Our data generation process for XMoP was executed on a cluster of 150 cloud compute cores all solving planning problems parallely for two weeks.

Collision Data Generation: For every planning problem in our synthetic dataset, we randomly sample an intermediate joint configuration from the saved motion plan for collision

Algorithm 1 WholeBodyIK

Input: Manipulator `URDF`, Whole-body predicted pose $\hat{p}_{t+k}$, Configuration-space proprioception j_t, Forward kinematics function ϕ, Max attempts M, Cost threshold κ
for *attempts* $1, 2, \ldots, M$ **do**
  $obj \leftarrow \|\hat{p}_{t+k} - \phi(\texttt{URDF}, j_{t+k})\|$ ▷ Set objective
  $bounds \leftarrow \texttt{URDF}$ ▷ Get joint limits
  $j_{t+k}, cost \leftarrow \texttt{SLSQP.minimize}(obj, bounds, j_t)$
  if $cost < \kappa$ **then**
    return j_{t+k}
  end if
  $j_t \leftarrow j_t + 0.1 * \mathcal{N}(0, I)$ ▷ Perturb and retry
end for

data generation. This intermediate configuration is guaranteed to be collision-free as it is part of a valid motion plan. To introduce collision into the scene, we add scaled Gaussian noise as shown in eq. 10 where $j \in \mathbb{R}^{\text{DoF}}$ is the collision-free joint configuration. The uniform scaling factor η ensures that the collision model is trained on various noise levels and remains reactive to subtle collisions of the end-effector typically encountered while moving close to environment obstacles. To prevent class specific bias, we ensure equal number of collision and collision-free joint configurations for training XCoD.

$$j_{\text{noisy}} = j + \eta * \mathcal{N}(0, I), \quad \eta = \text{Uniform}(0, 1) \tag{10}$$

C. Whole-Body IK for Configuration-Space Retrieval

The exact procedure for whole-body IK is shown in Algorithm 1. We attempt to retrieve the configuration-space candidate j_{t+k} using a quadratic program, starting with the initial guess j_t, which is the instantaneous joint state proprioception from the manipulator. We add Gaussian noise to j_t and re-attempt IK for ten times if the optimization gets stuck in a local minima. In practice, we find that 100 SLSQP iterations are enough for convergence with most manipulators.

D. Implementation Details

Model Architecture: A detailed architecture diagram of XMoP is shown in Fig. 8. As discussed in III-A and III-B, we use $D = 8$ (number of rigid-body links) and $H = 16$ (prediction horizon) as XMoP parameters. This makes the input sequence length as $D \times (H + 1) + 1 = 137$ *pose-tokens*.

Masking Strategy: As discussed in III-B, we restrict self-attention in XMoP using kinematic and morphology masking. Fig. 9 shows the horizon wise breakup of the square attention mask used in XMoP. The 1^{st} and 8^{th} *pose-tokens* at every horizon step is used for base link and end-effector respectively. However, the 6-DoF robots do not have a value for the 7^{th} *pose-token*; therefore, we mask out this key at all horizon steps encouraging morphology adaptation.

	XMoP	XCoD
Optimizer	AdamW	AdamW
Weight Decay	0.05	0.05
Learning Rate	1e-4	5e-4
LR Schedule	Linear to 1e-5 over 1 epoch	Cosine Annealing to 5e-5
Batch Size	64	12
Dataset Size	3.27×10^6	1×10^6
Epochs	20	1
Training Time	1 day 22 hours	5 hours

TABLE IV: Training hyperparameters for XMoP planner.

Loss Masking: While training XMoP, we backpropagate the loss only for unmasked morphology *pose-tokens*; i.e, the loss for 7^{th} *pose-token* at every horizon step is ignored for 6-DoF robots. For XCoD, the loss is backpropagated only for the augmented points that were sampled from the surface of the robot.

Diffusion Model: For the diffusion model, we use the DDIM [58] approach with 100 training iterations and 10 inference iterations. We use the Square Cosine Schedule [59] for diffusion noise parameterization which has been shown effective for policy training in prior work [37]. Furthermore, we follow the conventional practice of using an Exponential Moving Average (EMA) [60] during training and update the model weights with a decay factor of 0.9999 [37, 38].

Training and Inference: The training and inference algorithms for XMoP control policy is shown in Algorithm 2 and Algorithm 3 respectively. Table IV shows the training hyperparameters.

Classical Planner Baselines: We set a planning timeout of 20 seconds for AIT*+PyBullet, and 100 seconds for AIT*+XCoD, with a preferred minimum search time of 4 seconds and 20 seconds respectively. For fair comparison, we evaluate the smooth trajectory obtained after path shortcutting and interpolation, as these are necessary for practical deployment of classical planners.

E. XCoD Benchmark Experiments

Table V shows benchmark evaluations for the learnt collision model XCoD. We evaluate these metrics by sampling equal number of collision and collision-free states. For collision detection we use a binary condition i.e., a configuration is considered to be in collision if the ratio of detected link points in collision to the total number of manipulator points is greater than 0.001. This condition is also used for AIT*+XCoD baseline experiments as shown in Table I.

The XMoP control policy predicts whole-body poses for future time steps, which are evaluated by XCoD during MPC rollouts. In some out-of-distribution (OOD) scenarios, XMoP's predictions may lack accuracy, leading to spatial distortions in the reconstructed future poses that violate the manipulator's kinematic constraints. To examine XCoD's behavior in such cases, we pushed the collision model to its limits by analyzing

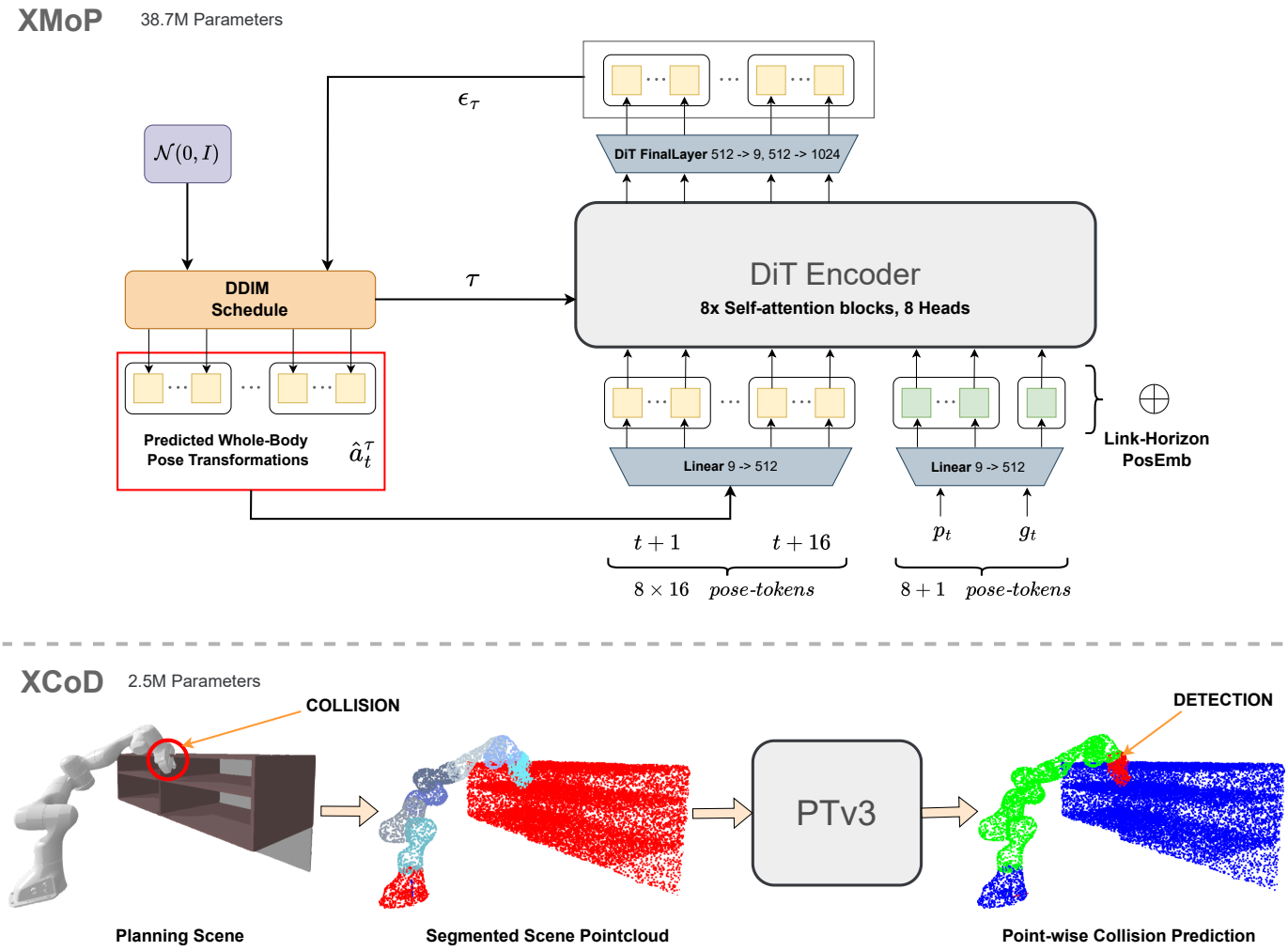


Fig. 8: Architecture diagram of different components in XMoP. Our planning policy uses self-attention to convert query *pose-tokens* into link-wise relative pose transformations $\hat{a}_t$ conditioned on whole-body poses p_t and goal end-effector pose g_t. The collision model XCoD takes a link-wise segmented pointcloud of the planning scene and predicts point-wise collision labels for the manipulator.

highly OOD problems. Collisions occur when the robot makes contact with an environmental obstacle or with itself. Fig. 11 presents results from our OOD evaluations, where the links of a Kuka IIWA robot are randomly scattered in 3D space. Our model successfully identifies both self-collisions and collisions with the environment, demonstrating strong generalization for collision detection in highly OOD scenarios.

F. Real-World Experiments

We developed a ROS [51] RViz-based interface to visualize the environment pointcloud and specify goal end-effector positions for XMoP. For open-loop rollouts, we take a single pointcloud of the obstacles at the beginning and internally rollout the MPC policy to generate the motion plan that is executed on the real robot. Whereas, for closed-loop rollouts, we deploy XMoP policy in real-time. Fig. 12 shows intermediate snapshots of two commercial robots using XMoP to plan across the three real-world domains as discussed in IV-B.

G. Architecture Ablations

Policy Architecture: We evaluated our whole-body pose transformation method using a comparatively simpler Transformer [36] policy, as depicted in Fig. 10. In contrast to the original stochastic diffusion policy used in XMoP, this policy was designed as a deterministic variant for predicting whole-body poses over a single horizon step. We refer to this policy as XMoP-S. Due to its deterministic nature, XMoP-S cannot be used with the MPC framework and therefore cannot plan in environments with obstacles. Hence, we assessed its performance on more relaxed 6-DoF reaching problems by removing obstacles from our benchmark. The XMoP-S policy achieved a 70.9% success rate on these reaching benchmark problems, showcasing that our whole-body control method is applicable to different base control models other than diffusion policies. XMoP-S was trained within 6 hours an a single A5000 GPU and the policy exhibited real-time performance while operating at an average rollout frequency of 10 hz.

Algorithm 2 Training XMoP control policy

Input: Planning demonstration dataset $\mathcal{D}$, Neural policy π_θ, Horizon length H, Configuration-space observation noise η, Forward kinematics function ϕ
for *every training step* **do**
  Sample $\texttt{KT}$, $\xi \leftarrow \mathcal{D}$ ▷ Get kinematic template and demonstration trajectory
  $\texttt{URDF} \leftarrow \text{GenerateURDF}(\texttt{KT})$ ▷ Generate URDF from kinematic template
  Sample $j_{t:t+H}, j_{goal} \leftarrow \xi$ ▷ Get a trajectory chunk and goal joint configuration
  $j_t \leftarrow j_t + \eta * \mathcal{N}(0, I)$ ▷ Add C-space observation noise
  $p_t \leftarrow \phi(\sim \texttt{URDF}, j_t)$ ▷ Get frame randomized whole-body observation pose
  $g_t \leftarrow \phi(\texttt{URDF}, j_{goal}).ee$ ▷ Get goal end-effector pose
  $p_{t+1:t+H} \leftarrow \phi(\texttt{URDF}, j_{t+1:t+H})$ ▷ Get target whole-body poses
  $T_{t+1:t+H} \leftarrow p_{t+1:t+H}(p_t)^{-1}$ ▷ Get whole-body relative pose transformations
  Sample $\tau \sim \text{Uniform}(1, 100)$, $\epsilon \sim \mathcal{N}(0, I)$ ▷ Get diffusion step and Gaussian noise
  $a_t^\tau \leftarrow \text{AddNoise}(T_{t+1:t+H}, \epsilon, \tau)$ ▷ Add noise using a scheduler
  $\epsilon_\theta \leftarrow \pi_\theta$ ▷ Get the noise prediction model from policy
  Predict $\hat{\epsilon} \leftarrow \epsilon_\theta(a_t^\tau, p_t, g_t, \tau)$ ▷ Predict diffusion noise
  $\mathcal{L}_{xmop} \leftarrow \text{MSE}(\hat{\epsilon}, \epsilon)$ ▷ Compute loss
  Update θ with AdamW and $\mathcal{L}_{xmop}$ ▷ Update policy parameters with gradient descent
end for

Algorithm 3 Inference with XMoP control policy

Input: Manipulator $\texttt{URDF}$, Trained neural policy π_θ, Configuration-space proprioception j_t, Prediction horizon H, Goal end-effector pose g_t, Forward kinematics function ϕ, Denoising steps K
$p_t \leftarrow \phi(\texttt{URDF}, j_t)$ ▷ Get whole-body observation pose
Sample $\hat{a}_t^K \sim \mathcal{N}(0, I)$ ▷ Get Gaussian noise
$\epsilon_\theta \leftarrow \pi_\theta$ ▷ Get the noise prediction model from policy
for $\tau = K, K-1, \ldots, 1$ **do**
  Predict $\hat{\epsilon} \leftarrow \epsilon_\theta(\hat{a}_t^\tau, p_t, g_t, \tau)$ ▷ Predict diffusion noise
  $\hat{a}_t^{\tau-1} \leftarrow \text{Denoise}(\hat{a}_t^\tau, \hat{\epsilon}, \tau)$ ▷ Remove noise using a scheduler
end for
$\hat{p}_{t+1:t+H} \leftarrow \hat{a}_t^0 p_t$ ▷ Get predicted whole-body poses
for $k = 1, 2, \ldots, H$ **do**
  $j_{t+k} \leftarrow \text{WholeBodyIK}(\hat{p}_{t+k}, j_t, \texttt{URDF})$ ▷ Retrieve configuration-space actions
end for

Embodiment	Segmentation IoU (%)	Collision Precision (%)	Collision Recall (%)	Collision Accuracy (%)
Panda	79.4	96.9	98.6	97.7
Sawyer	88.4	97.9	99.1	98.5
IIWA	85.0	99.5	99.0	99.2
Gen3 6-DoF	67.0	71.1	99.4	79.6
Gen3 7-DoF	67.3	81.7	98.5	88.2
UR5	73.0	97.1	66.8	82.4
UR10	76.1	98.2	70.7	84.7

TABLE V: Benchmark results for the learned collision model XCoD. We evaluate the semantic segmentation performance using the Intersection Over Union (IoU) metric. Furthermore, we show the collision detection performance with precision, recall, and accuracy metrics. *Note*: The recall for UR robot was affected by the imprecise collision mesh available for ground-truth validation, which led to false positive labels during evaluations.

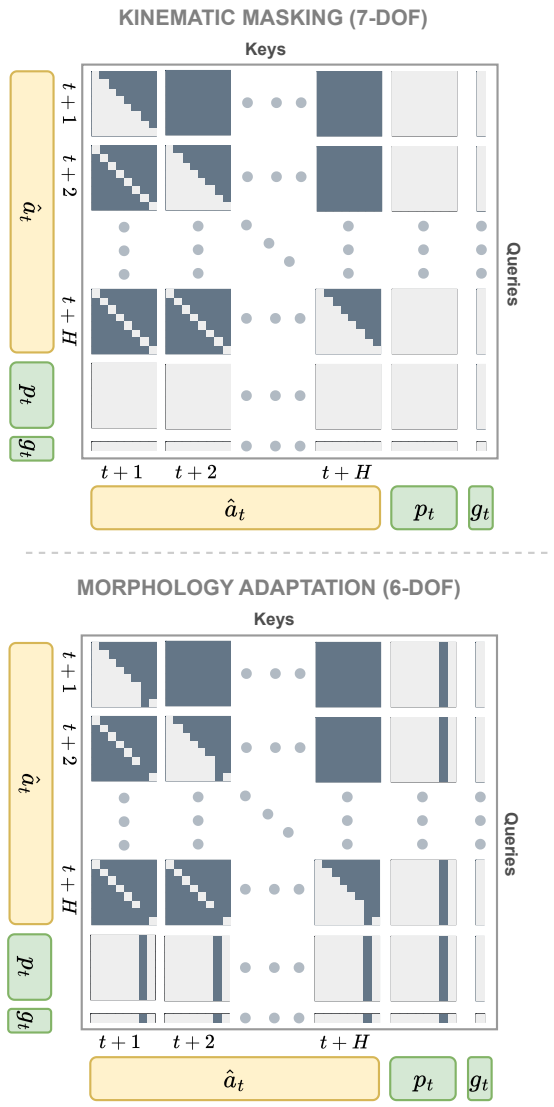


Fig. 9: Self-attention masking in XMoP. Masked, Unmasked, Query *pose-tokens*, Condition *pose-tokens*. Kinematic mask restricts the self-attention to parent or ancestor links within a horizon step, and to the same link in current and previous horizon steps. Morphology adaption for 6-DoF is enabled by masking out keys of the unused *pose-token*.

Segmentation Architecture For our collision model, we also experimented with semantic segmentation using a PointNet++ [61] backbone. We found the segmentation performance of PointNet++ to be worse than that of the Point Transformer model [43]. Furthermore, due to its high memory requirements and latency, we found PointNet++ to be practically infeasible for MPC formulation with XMoP.

H. Failed Ideas

We tried to integrate a few standard ideas from the neural planning literature, however they performed relatively poorly compared to XMoP:

- **C-space Goals for Planning**: When provided with the whole-body pose target as additional *pose-tokens* for goal conditioning, the policy overfits to the sub-optimal planning behaviors of sampling based planners and generates trajectories with higher path lengths.
- **Input Normalization**: Normalization of neural network inputs is a conventional approach that has been used for improving policy performance in prior works [3, 37, 49]. However, we did not find it useful for our pose transformation method. In contrast, the input-normalized policy takes longer time to train and fails to generalize to unseen embodiments.

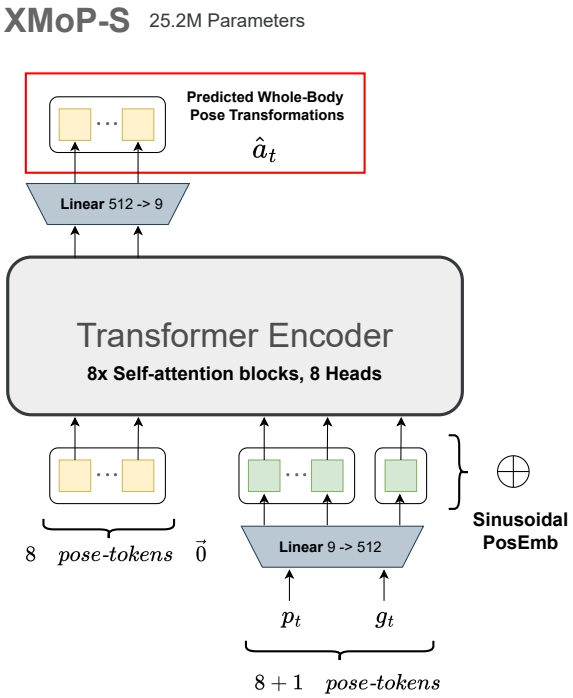


Fig. 10: XMoP-S is a deterministic 6-DoF reaching policy that *zero-shot* generalizes to unseen robotic manipulators.

I. Limitations and Future Work

Planning Failures Our policy struggles to avoid obstacles when the robot's end-effector is close to the specified goal pose. In such scenarios, the generated trajectory samples from the diffusion policy lack the required diversity to avoid obstacles and are too biased toward reaching the goal. Hence, the predicted trajectory with fewest collisions is still a trajectory in collision. A possible solution to this issue could be to add sub-optimal evasive trajectories to the demonstration dataset for near-goal collision avoidance. Moreover, due to the lack of joint bound awareness, for complex planning problems some manipulators get locked at joint limits, as shown in Fig. 13(a). Further fine-tuning XMoP policy with few demonstrations from the robot might help avoid such undesired behaviors.

Collision Failures Fig. 13(b) shows an instance of false positive detection from XCoD model when in reality the highlighted robot link is collision-free. We found such instances to be rare and attribute them to the model overfitting to biases

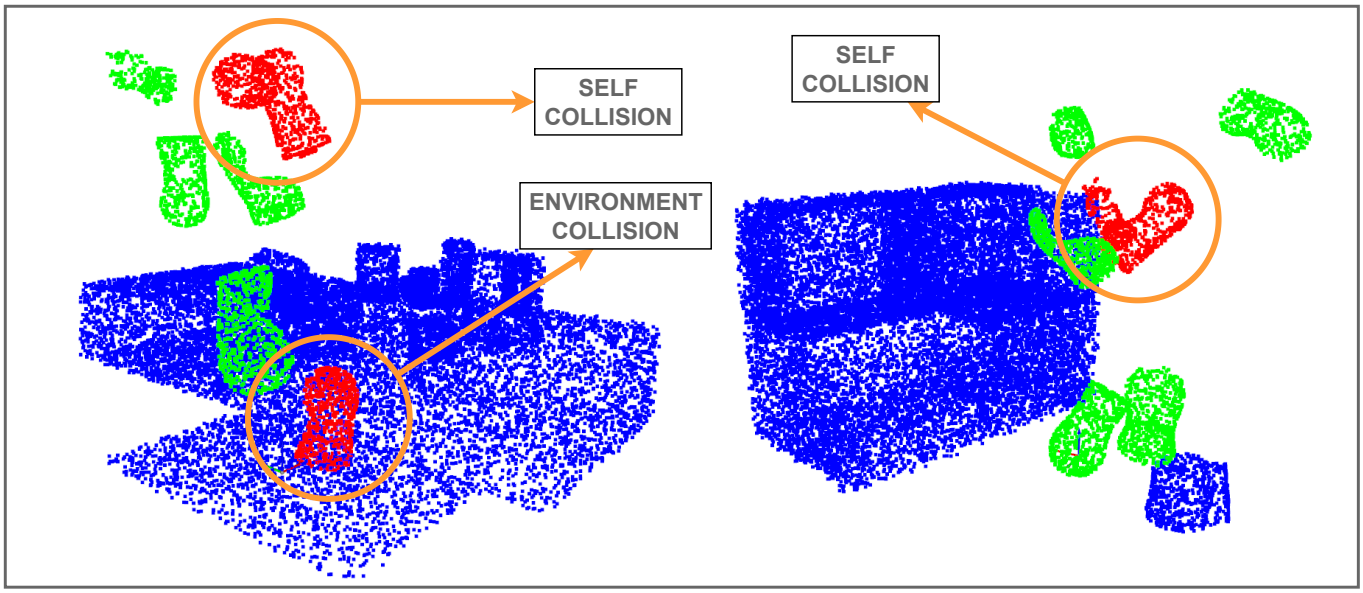


Fig. 11: Pointcloud captured from a planning scene where the links of Kuka IIWA robot are randomly scattered in 3D space. XCoD identifies self-collisions and environment collisions showcasing highly out-of-distribution generalization for collision detection.

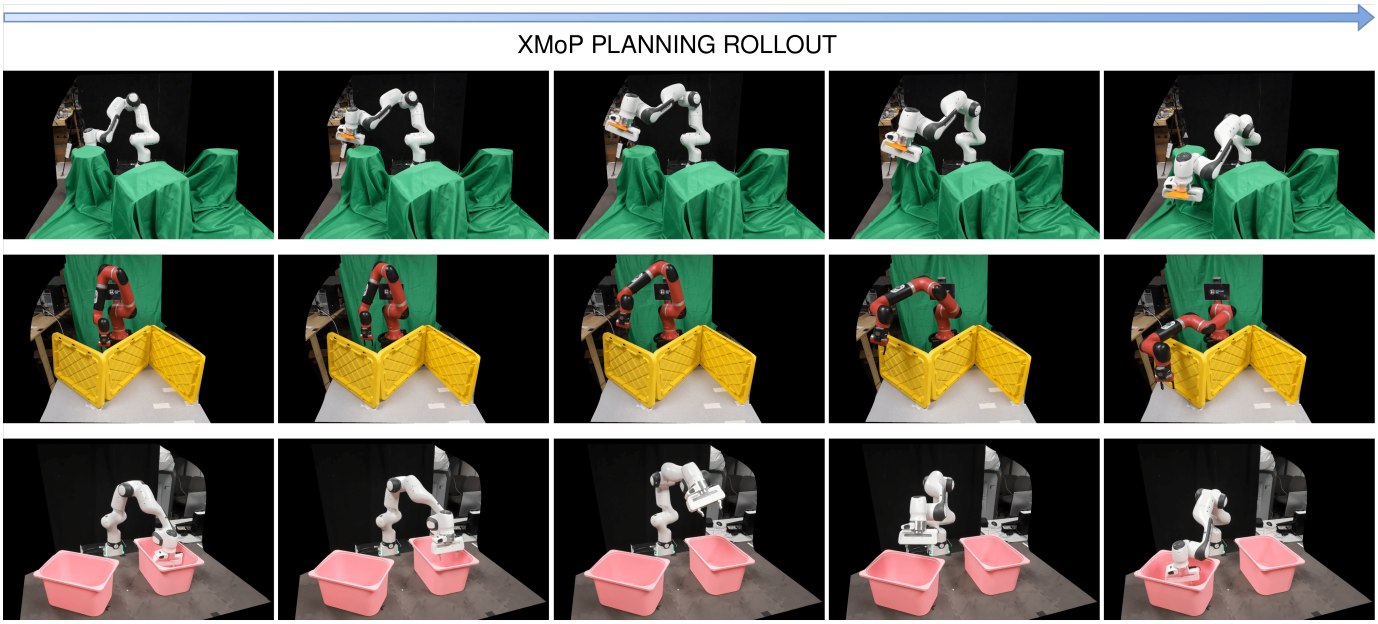
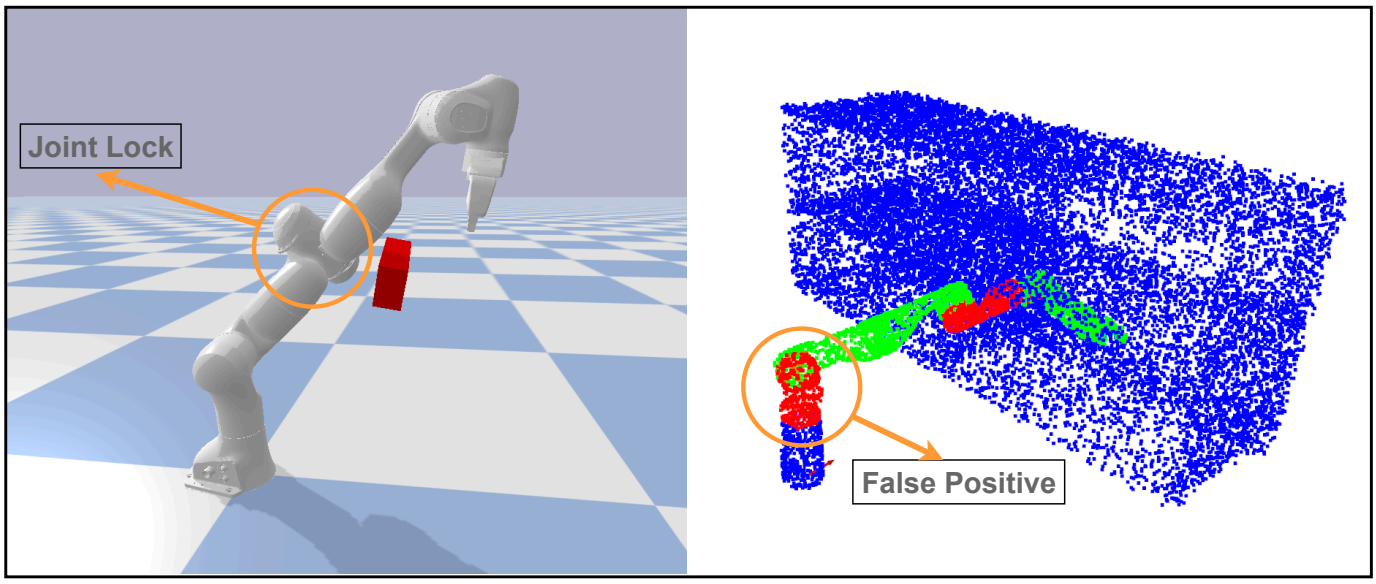


Fig. 12: XMoP planning rollouts across three unseen real-world domains for two unseen 7-DoF commercial manipulators Franka FR3 and Sawyer (better viewed when zoomed in). Videos of policy rollouts are available at https://prabinrath.github.io/xmop.

in the training data. Moreover, the solution time of our MPC policy is highly dependent on the inference speed of the collision model. Our current implementation allows 1.25 seconds for each MPC evaluation, which checks $B = 16$ different possible future trajectories of the manipulator as predicted by the control policy. We hope future improvements in 3D semantic segmentation methods will enhance the efficiency of XCoD collision detection, thereby improving the inference speed of our MPC policy.

Out-of-Distribution Planning Problems XMoP policy struggles to plan for OOD goal poses and environment setups, a common issue with behavior cloning methods. Training XMoP on a more diverse planning dataset, where the goals are evenly distributed within the reachable workspace, might help in learning better policies for OOD planning generalization.



(a) (b)

Fig. 13: Limitations of XMoP planner. (a) As our policy is not conditioned on joint bounds, it runs into joint limits for complex planning tasks. The problem is illustrated for a Panda robot, where our policy does not recover from the locked configuration and fails to plan for the goal pose specified in red. (b) False positive collision detection for a Gen3 7-DoF robot.