Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

Sketch-to-Skill: Bootstrapping Robot Learning with Human Drawn Trajectory Sketches

Peihong Yu*†, Amisha Bhaskar*†, Anukriti Singh*, Zahiruddin Mahammad* and Pratap Tokekar*
*Department of Computer Science
University of Maryland, College Park, Maryland, 20742
Emails: {peihong, amishab, anukriti, zahirmd, tokekar}@umd.edu
†Peihong Yu and Amisha Bhaskar contributed equally to this work.

Abstract—**Training robotic manipulation policies traditionally requires numerous demonstrations and/or environmental rollouts. While recent Imitation Learning (IL) and Reinforcement Learning (RL) methods have reduced the number of required demonstrations, they still rely on expert knowledge to collect high-quality data, limiting scalability and accessibility. We propose SKETCH-TO-SKILL, a novel framework that leverages human-drawn 2D sketch trajectories to bootstrap and guide RL for robotic manipulation. Our approach extends beyond previous sketch-based methods, which were primarily focused on imitation learning or policy conditioning, limited to specific trained tasks. SKETCH-TO-SKILL employs a Sketch-to-3D Trajectory Generator that translates 2D sketches into 3D trajectories, which are then used to autonomously collect initial demonstrations. We utilize these sketch-generated demonstrations in two ways: to pre-train an initial policy through behavior cloning and to refine this policy through RL with guided exploration. Experimental results demonstrate that SKETCH-TO-SKILL achieves ∼96% of the performance of the baseline model that leverages teleoperated demonstration data, while exceeding the performance of a pure reinforcement learning policy by ∼170%, only from sketch inputs. This makes robotic manipulation learning more accessible and potentially broadens its applications across various domains.**

I. INTRODUCTION

Robots are increasingly being deployed in dynamic environments, where they must perform a wide range of tasks with precision and adaptability. One of the key challenges in enabling robots to learn new skills lies in specifying complex, task-specific behaviors. Learning from Demonstration (LfD) ([4]) has become a widely used approach, allowing robots to acquire novel motions by imitating expert-provided trajectories. However, collecting demonstration data for LfD is challenging, particularly for high degree-of-freedom (DOF) robots performing manipulation.

While methods like kinesthetic teaching and teleoperation [7, 12, 5] are well-established and effective approaches for collecting demonstrations, the robotics community continues to explore complementary methods to expand the toolkit available for robot learning. Several new approaches have emerged, such as using manually-operated grippers instrumented with smartphone apps [27] and Virtual Reality (VR) based teleoperation systems [17], offering more intuitive hardware interfaces for collecting demonstrations. There has also been growing interest in leveraging an innate human ability to communicate spatial ideas and motions through simple sketches. For example, a quick sketch of a path can easily communicate the intended movement for navigating toward a goal location.

Researchers have begun to explore this promising direction. RT-Trajectory [14] introduced the notion of sketches and showed how to use coarse trajectory sketches for policy conditioning in Imitation Learning (IL). RT-Sketch [30] extended this concept to leverage hand-drawn sketches of the entire environment for goal-conditioned IL. These methods demonstrated the potential of utilizing sketches in robotics, but they were primarily focused on IL and biased towards tasks they were specifically trained on. Zhi et al. [34] expanded this idea with *diagrammatic teaching*, where users instruct robots by sketching motion trajectories directly on 2D images of the scene. Their approach uses density estimation and ray tracing to reconstruct 3D trajectories from the sketches, thus limiting its ability to replicate only the provided sketches and restricting its generalization to new or unseen task setups.

Unlike prior work that used sketches only as conditioning in IL, we present a more generalizable approach that learns to predict 3D trajectories from sketches in Reinforcement Learning (RL). Specifically, we propose SKETCH-TO-SKILL (Figure 1), a framework that bootstraps and guides RL using sketches. Our approach first learns to map 2D sketches to 3D trajectories, which are then used to collect demonstrations. We utilize these sketch-generated demonstrations in two ways: first, by pre-training an initial policy through Behavior Cloning (BC), and second, by refining this policy through reinforcement learning with guided exploration. Although sketch-generated demonstrations are not as precise or high-quality as teleoperated ones, they still contain enough useful information to aid RL and reduce learning time.

This approach capitalizes on the accessibility of sketching, allowing contributions from non-experts without requiring specialized hardware. By treating these sketch-based trajectories as approximate guiding signals, we demonstrate that even simple 2D sketches can enable policies that achieve performance comparable to those trained with teleoperated demonstrations. We summarize our contributions as follows:

(1) We identify and address a crucial gap by integrating sketches into RL, extending their application beyond imitation learning and policy conditioning.
(2) We propose SKETCH-TO-SKILL, a framework that leverages sketches to bootstrap and guide RL, reducing

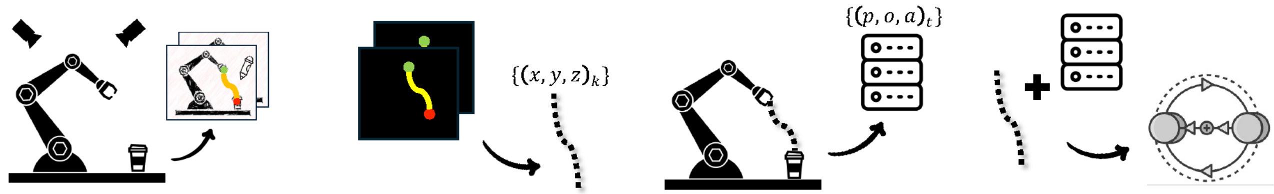


Fig. 1: Learning a new skill in the SKETCH-TO-SKILL framework. Step 1: Capture the task scenario from two views and collect human-drawn sketches. Step 2: Convert 2D sketches to 3D trajectories using a pretrained generator. Step 3: Execute generated trajectories to collect experience data. Step 4: Learn manipulation policy using reinforcement learning bootstrapping from behavior cloning and using guidance for experience data.

reliance on high-quality, real-world demonstrations.

(3) Through extensive experiments, we demonstrate that sketches, despite their low fidelity, significantly accelerate learning by improving exploration and task comprehension in RL. SKETCH-TO-SKILL achieves $\sim$96% of the performance of the baseline model utilizing high-quality teleoperation demonstrations, while exceeding the performance of a pure reinforcement learning policy by $\sim$170% during evaluation.

II. RELATED WORK

Learning from Demonstration (LfD). LfD [4] is a key method in robot learning, allowing robots to acquire skills through expert demonstrations, bypassing the complexities of action programming and cost function design [6]. Kinesthetic teaching, where an expert physically guides the robot while its movements are recorded, is widely used in methods like DMPs [21, 16], Probabilistic Movement Primitives [24], and stable dynamical systems [19, 23, 1]. However, it is labor-intensive and challenging to scale. Teleoperation [28], where users control robots remotely, offers more flexibility but can be complex and requires expertise to operate. VR interfaces [33, 17] provide a more immersive alternative but depend on specialized hardware. To overcome these limitations, recent research has introduced more accessible approaches, like sketch-based demonstrations [11].

Sketches in Robotics. Sketches have become a powerful tool in computer vision, aiding tasks like scene understanding [9] and object detection [8, 3]. RT-Sketch [30] first explored hand-drawn sketches for goal-conditioned imitation learning (IL), using them to define tasks intuitively. RT-Trajectory [14] extended this by using trajectory sketches as IL policy conditioning, either drawn by users or generated by a Large Language Model from task descriptions. Similarly, the Diagrammatic Teaching framework [34] uses density estimation and ray tracing to reconstruct 3D trajectories from the sketches. These methods, however, only use sketches as conditioning for task completion, and thus do not generalize beyond the tasks where the sketches are provided.

Demonstration-Enhanced Strategies for Efficient RL. Incorporating demonstration data in RL can improve sample efficiency, especially in environments where rewards are sparse. Methods such as Reinforcement Learning from Prior Data (RLPD) [29], Imitation Bootstrapped RL (IBRL) [15] and PLANRL [2] take advantage of prior demonstrations by embedding them into the agent's replay buffer. During training, these examples are oversampled, offering the agent more frequent exposure to expert-guided trajectories. Such approaches significantly improve learning speed and performance, particularly in continuous control tasks where learning from scratch can be prohibitively slow and inefficient [31]. Our research expands upon these techniques by exploring how sketch-based trajectories can be used as an additional source of prior data in RL.

III. SKETCH-TO-SKILL

Our approach bootstraps robot learning from trajectory sketches, significantly lowering the barrier to entry for robotic task specification. This section details our three-stage method: (1) training a Sketch-to-3D Trajectory Generator, (2) obtaining 3D trajectories and execution experiences through the Generator and open-loop servoing, (3) pre-training an initial robotic manipulation policy through behavior cloning, and refining the policy through reinforcement learning with guided exploration. By integrating intuitive human input with powerful learning algorithms, our approach aims to create more accessible and adaptable robotic learning systems.

A. Sketch-to-3D Trajectory Generator

Our method begins with a Sketch-to-3D Trajectory Generator, $\boldsymbol{T}$, that translates a pair of 2D sketch images (I^1, I^2) obtained from different viewpoints into corresponding 3D robot trajectory ξ_g. To train this generator, we use a dataset consisting of 3D robot end-effector trajectories along with their 2D sketches from two viewpoints. These trajectories can be obtained from various sources, such as robot arm play data in simulation or real world where the robot executes sequences of actions, or existing recorded trajectory datasets transformed to match the workspace coordinates. Sketches during inference can be provided by a human on RGB images of the scene, as shown in Figure 4. However, the sketches fed as input to the generator are 2D projections on blank

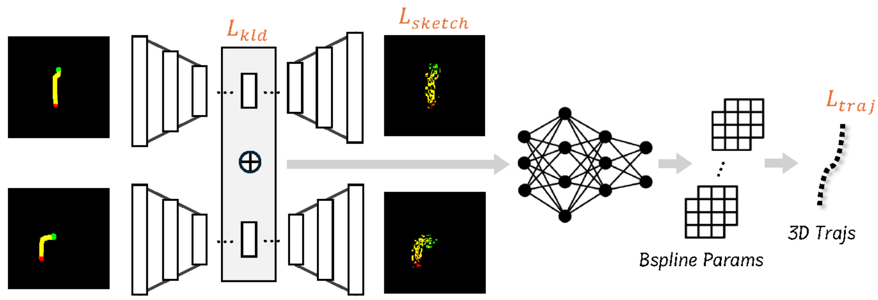


Fig. 2: The Sketch-to-3D Trajectory Generator takes dual-view 2D sketches as inputs and predicts B-spline parameters to generate the final 3D trajectory output.

backgrounds, with green and red dots representing the start and end points respectively, and yellow lines for the trajectory (see Figure 2 for an example). By focusing solely on the trajectory information without additional scene complexity, our model can efficiently learn to encode the dual-view sketches and decode them into the corresponding 3D trajectory.

The generator uses a neural network to map dual-view 2D sketches to 3D trajectories, where we adopt a hybrid architecture combining a Variational Autoencoder (VAE) [20] and a Multilayer Perceptron (MLP), as illustrated in Figure 2. The VAE encodes sketches from two viewpoints, ideally orthogonal, to resolve depth ambiguity and capture essential trajectory features. The MLP decoder generates B-spline [25] control points $\boldsymbol{C} \in \mathbb{R}^{n_{cp}\times 3}$ from the latent representation, which we then use to interpolate smooth 3D trajectories. We adopted uniform knots and pre-compute the B-spline parametrization matrix $\boldsymbol{W} \in \mathbb{R}^{n_{tp}\times n_{cp}}$ to reduce computational complexity and facilitate efficient backpropagation. The calculation of $\boldsymbol{W}$ only depends on the uniform knot vector $\boldsymbol{u}$ and the desired number of points n_{tp} in the generated trajectory, and can be pre-calculated using the Cox-de Boor recursion formula (also known as de Boor's algorithm [10], see Appendix VII-A for details). Then the final trajectory generation is simply a matrix multiplication: $\xi_g \in \mathbb{R}^{n_{tp}\times 3} = \boldsymbol{W}\cdot\boldsymbol{C}$. With the generated control point parameter $\boldsymbol{C}$, we can also easily generate trajectories of varying density from the same parameters, making our method adaptable to different task requirements.

Our training process uses a multi-component loss function $L = L_{traj} + L_{sketch} + L_{kld}$, where L_{traj} handles trajectory reconstruction, L_{sketch} manages sketch reconstruction (Mean Square Error), and L_{kld} is KL-divergence for latent space regularization (Figure 2). This ensures accurate trajectory generation while preserving sketch fidelity and latent space structure. We also applied data augmentation to both the sketch images and the trajectories to enhance the model's robustness and generalization (more details can be found in Appendix VII-B).

We can use the trained Sketch-to-3D Trajectory Generator, $\boldsymbol{T}$, to generate demonstrations $\{\xi_D\}$ for learning new tasks using sketches drawn by a human. Specifically, the human draws trajectory sketches on two views of RGB images captured from the initial task state. This is similar to how human-drawn sketches are generated in prior works [14, 34]. These paired sketch images, $\{(I^1, I^2)\}$, are input into our trained generator, which produces corresponding 3D trajectories, $\{\xi_g\}$, serving as the basis for guiding the robot's actions. We can also generate more than one trajectory from the same pair of sketches by adding controlled noise to the latent representation. Then we proceed to collect demonstrations for manipulation policy learning. We execute these trajectories on the robotic arm using open-loop servoing, which enables precise trajectory following based on pre-computed motor commands. During execution, we record a demonstration dataset $\{\xi_D = \{(p_t, o_t, a_t)\}_{t=1}^{T}\}$ at a fixed frequency, where $p_t = (x, y, z)_t$ denotes the robot's end-effector 3D position, o_t represents the robot's observation, a_t is the corresponding action, and T is the total number of timesteps per demonstration. The collected demonstrations, which do not need to be optimal, follow the intended path while capturing the robot's actual behavior in the target environment. They serve as an effective starting point for bootstrapping the policy learning process, offering initial guidance grounded in the robot's real-world performance.

B. Policy Learning

We now describe the policy learning of the SKETCH-TO-SKILL algorithm (given in Algorithm 1). Taking as input the demonstration data $\{\xi_D\}$ collected from our Sketch-to-3D Trajectory Generator $\boldsymbol{T}$ and through open-loop serving (lines 4–5), our approach combines IL and RL to effectively bootstrap and refine the policy. Specifically, we build upon the Imitation Bootstrapped Reinforcement Learning (IBRL) framework [15], integrating our sketch-based trajectories to guide and constraint policy search space.

In IBRL we replace traditional real-world demonstrations with sketch-generated demonstrations. Initially, these sketch-based demonstrations are used to train an IL policy (line 6), which serves as a coarse approximation of the task. Although these sketches do not capture every fine detail of manipulation (e.g., gripper closing/opening actions or exact force control), our hypothesis posits that they still carry significant, actionable information that can effectively guide the learning process in reinforcement learning (RL). We leverage this information in RL in two ways (as shown in Fig. 3):

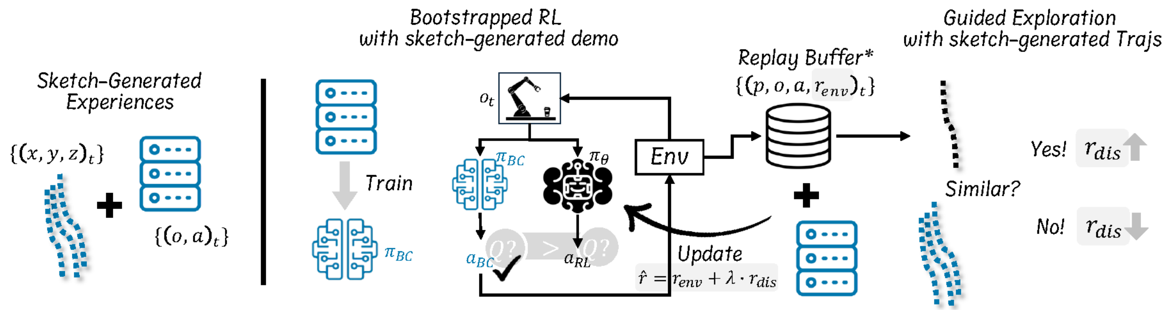


Fig. 3: Overview of SKETCH-TO-SKILL integrating sketch-generated demonstrations with reinforcement learning. Sketch-generated experiences train an IL policy, which bootstraps the RL process. A discriminator guides exploration by rewarding similarity to sketch-generated trajectories. The final action, combining IL and RL policy outputs, further enhances the exploration guidance. The asterisk after "Replay Buffer" indicates that the buffer is initialized with the open-loop servoing demonstrations.

Algorithm 1 SKETCH-TO-SKILL. Major modifications of IBRL highlighted in blue.

1: **Hyperparameters:** Number of critics E, number of critic updates G, update frequency U, exploration std σ, noise clip c, number of generated trajectories m per input sketch pair, reward weighting term λ
2: **Inputs:** Pre-trained Sketch-to-3D Trajectory Generator $\boldsymbol{T}$, sketch dataset $\mathcal{S} = \{(I_i^1, I_i^2)\}_{i=1}^n$,
3: **Outputs:** Policy π_θ, discriminator D_ψ

——— *Stage 1: Demonstration Generation* ———

4: $\{\xi_g\}_{1:mn} \leftarrow$ generate m trajectories per sketch from $\mathcal{S}$ using $\boldsymbol{T}$
5: $\{\xi_D\}_{1:mn} \leftarrow$ generated demonstrations through open-loop servoing

——— *Stage 2: Policy Learning* ———

6: Train imitation policy π_{IL} on demonstrations $\{\xi_D\}_{1:mn}$ using the selected IL algorithm.
7: Initialize policy π_θ, target policy $\pi_{\theta'}$, and critics Q_ϕ, target critics $Q_{\phi'}$, discriminator D_ψ for $i = 1, \ldots, E$
8: Initialize replay buffer B with demonstrations $\{\xi_D\}_{1:mn}$
9: **for** $t = 1$ **to** N **do**
10: Observe current observation o_t from the environment
11: Compute IL action $a_t^{\text{IL}} \sim \pi_{IL}(o_t)$ and RL action $a_t^{\text{RL}} = \pi_\theta(o_t) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$
12: Sample a set K of 2 indices from $\{1, 2, \ldots, E\}$
13: Select action a_t with higher Q-value from $\{a^{\text{IL}}, a^{\text{RL}}\}$
14: Execute action a_t
15: Store transition $(p_t, o_t, a_t, r_t, p_{t+1}, o_{t+1})$ in replay buffer B
16: **if** $t\%U = 0$ **then**
17: Perform discriminator D_ψ update by optimizing Equation 1
18: Perform TD3 update using minibatches from replay buffer B with augmented reward by Equation 2 [13]
19: **end if**
20: **end for**

(1) Bootstrap RL with Sketch-Generated Demos: Even though sketch-generated trajectories are not as detailed as teleoperated demonstrations, they provide a foundational blueprint of the task. We leverage these initial trajectories to bootstrap our RL algorithm, giving it a preliminary direction and reducing the cold start problem common in RL scenarios. This use of imperfect demonstrations is intended to establish an initial policy that avoids random exploration at the outset, making subsequent training more focused and efficient.
(2) Guide Exploration During RL: As the agent progresses in its learning, the sketch-generated trajectories continue to serve as a guide, shaping the exploration strategy. Instead of relying on these trajectories as definitive guides, we treat them as rough outlines that suggest areas of the task space worth exploring. This guided exploration helps concentrate the agent's learning efforts on potentially fruitful regions of the action space, thus optimizing the learning speed and improving the relevance of the experiences gathered.

In both steps, the use of sketch-generated trajectories acknowledges their limitations—they are not treated as ground truth but as valuable signals to help bootstrap RL and guide exploration throughout the learning process. For the RL algorithm, we employ TD3 [13], an off-policy algorithm known for its sample efficiency. In our approach, the replay buffer is initialized with the sketch-generated demonstration trajectories (line 8), which provide an initial foundation for learning and is later updated with online experiences as the agent interacts with the environment. This combination allows the agent to refine its policy through both sketch-generated demonstration data ξ_D and real-world interaction (line 13).

To further enhance the learning process and maintain consistency with the sketch-generated trajectories, we introduce a discriminator-based guided exploration mechanism [18]. This discriminator, D_ψ, is trained to distinguish between trajectories produced by our Sketch-to-3D Trajectory Generator and

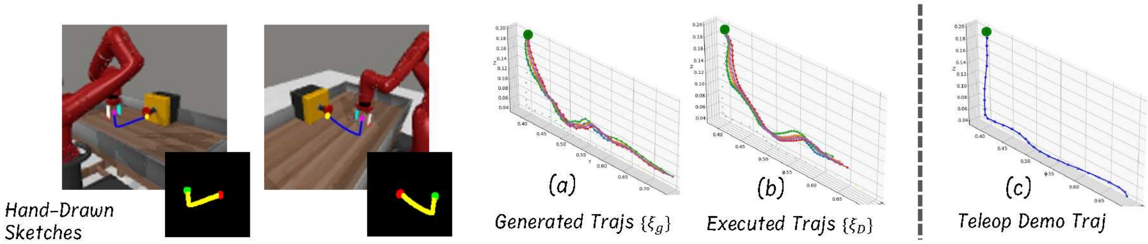


Fig. 4: Multi-stage trajectory generation and execution. On the left, we show hand-drawn sketches on scenario RGB images and the extracted sketches on a blank background, (a) generated trajectory from the Sketch-to-3D Trajectory Generator, and (b) executed trajectory via open-loop serving. In (c), we visualize a teleoperated demo for the same task for reference.

those generated by the current policy:

$$\mathcal{L}_{D(\psi)} = \mathbb{E}_{p,g\sim\{\xi_D\}}[\log D_\psi(p, \Delta p, g)] + \mathbb{E}_{p,g\sim\pi_\theta}[\log(1 - D_\psi(p, \Delta p, g))], \quad (1)$$

where p represents the end-effector location, Δp is the normalized difference between the current and next end-effector positions, capturing local trajectory characteristics, and g is the task-specific information (e.g., target location). This formulation allows the discriminator to assess trajectory similarity while accounting for task variability. We then augment the TD3 reward function with an additional term based on the discriminator's output (line 18):

$$\hat{r}(o_t, a_t) = r(o_t, a_t) + \lambda \log D_\psi(p_t, \Delta p_t, g), \quad (2)$$

where λ is a hyperparameter controlling the influence of the discriminator. This augmented reward encourages the policy to explore state-action spaces more likely to produce trajectories similar to those generated from human sketches, potentially leading to faster learning and better performance.

Our overall learning process iterates between TD3 optimization and discriminator training. In each iteration: (1) We update the discriminator using the latest policy-generated trajectories and the original sketch-generated trajectories (line 17). (2) We then update the policy and Q-functions using TD3, with the augmented reward (line 18) and guidance from the frozen IL policy (line 13). This iterative process allows the policy to refine its behavior while maintaining similarity to the initial demonstrations derived from human sketches. By combining IL, TD3, and discriminator-based guided exploration, we create a cohesive learning framework that effectively leverages sketch-based demonstrations to accelerate and improve the learning of complex manipulation tasks. Please see the Appendix for more implementation details and a complete list of hyper-parameters.

IV. Experiments

We report our evaluation of Sketch-to-Skill, focusing on its main components: the Sketch-to-3D Trajectory Generator, the Imitation-Bootstrapped RL Policy learning, and the use of the discriminator. Our experiments address the following key questions:

Q1 How effectively does the Sketch-to-3D Trajectory Generator convert 2D sketches into usable 3D robot trajectories?

Q2 Can Sketch-to-Skill utilize sketch-generated demonstrations to achieve comparable performance to traditional methods using high-quality demonstration data?

Q3 How do various design choices in Sketch-to-Skill, such as the number of generated demonstrations per sketch and the discriminator reward weighting, affect the learning and refinement of robotic policy?

Q4 How well does our method translate to the real world?

A. Evaluation of the Sketch-to-3D Trajectory Generator

The Sketch-to-3D Trajectory Generator is a key component of Sketch-to-Skill, translating 2D sketch inputs into 3D robot trajectories. To train this generator, we collect data of the robot arm executing *play* trajectories. We record the 3D trajectories as well as their 2D projections from two viewpoints. We create such a dataset in the Metaworld [32] simulation environment as well as a separate one using actual hardware (Figure 13). Once the Sketch-to-3D Trajectory Generator is trained, we can use hand-drawn sketches as input to predict 3D trajectories.

Performance on Hand-drawn Sketches. We provide an example using the `ButtonPress` task to qualitatively assess the generator's effectiveness with hand-drawn inputs (Figure 4). We asked users to provide sketches for the task and also separately collected actual demonstrations as a reference. We see that the Sketch-to-3D Trajectory Generator was able to predict trajectories (Figure 4a) similar to the actual demonstrations (Figure 4c). We also generate more than one trajectory from the same pair of sketches by adding controlled noise to the latent representation. This approach allows us to produce a range of plausible trajectories for a given sketch input, enriching the demonstration set and potentially leading to more robust and adaptable robot policies. We then execute the generated trajectories to produce demonstrations for training the policy (Figure 4b). Despite the inherent variability in sketch inputs, the executed trajectory further validates the practical applicability of our approach. This demonstrates our model's robustness to sketch imperfections and its ability to reliably interpret user intent, bridging the gap between simple

2D sketches and actionable 3D robot trajectories.

Latent Space Representation and Interpolation. To further understand the generator's latent space, we performed linear interpolation in the latent space between different input samples. Specifically, we selected two distinct sketch pairs with different trajectories, extracted their feature vectors, linearly interpolated between them, reconstructed the sketches, and generated new trajectories. Figure 12 shows smooth transitions in both 2D sketches and 3D trajectories across the interpolated latent space. This smoothness demonstrates that our model has learned a continuous and semantically meaningful representation, suggesting good generalization capability to unseen inputs that lie between known examples [20]. The coherence between interpolated sketches and their corresponding 3D trajectories further validates the model's robust sketch-to-trajectory mapping.

Effect of VAE in Sketch-to-3D Trajectory Generator. We also conducted an ablation study of the Sketch-to-3D Trajectory Generator to evaluate the effect of the VAE on the architecture. Using a dataset of 1000 trajectories split 80:20 for training and validation, we report the training and validation losses in Table I. We compare the performance of the model with the VAE (shown in Figure 2) to a variant without the VAE, where only a CNN is used. This CNN has the same architecture as the VAE's encoder, but without the decoder and loss components. Our results show that incorporating the VAE consistently improves performance across all metrics, including reconstruction loss and trajectory loss.

TABLE I: Performance Metrics for generator model

Metric	with VAE	without VAE
Training Loss	**0.6019**	0.6286
Validation Loss	**0.9128**	1.2804
Reconstruction Loss	**0.0004**	0.3258
KLD Loss	**69.8592**	107.167
Parameter MSE Loss	**0.0820**	0.0928
Trajectory Loss	**0.770**	0.1180

B. Comparisons with Baselines

In this section, we conduct extensive experiments in Robomimic [22] and MetaWorld [32] to answer Q2: *can* SKETCH-TO-SKILL *utilize sketch-generated demonstrations to achieve comparable performance to traditional methods using high-quality demonstration data?* Specifically, we compare SKETCH-TO-SKILL with: (1) IBRL [15], a strong baseline that utilizes traditional high-quality demonstration data (rather than sketches as what our method uses), and (2) TD3 [13], a state-of-the-art pure RL approach without using any demonstrations. We hypothesize that although the sketches have only partial information (namely, 2D projections of 3D trajectories and no gripper information), we can still generate good enough demonstration data to perform comparably with the baseline that uses full demonstrations. We show that to be the case in these experiments.

We perform evaluations on two-stage `PickPlaceCan` task from Robomimic and six tasks from the MetaWorld benchmark, namely `Coffeepush`, `Boxclose`, `Buttonpress`, `Reach`, `Reachwall`, and `ButtonpressTopdownwall`, each using sparse 0/1 task completion rewards at the end of each episode. For each task, we collected 10 high-quality demonstrations using an expert policy in Robomimic and 3 high-quality demonstrations in MetaWorld. These demonstrations served as our baseline for traditional demonstration-based methods. For our approach, we collected a total of three hand-drawn sketches, one on each demonstration's initial frames (Figure 4). These sketches were used to generate and execute trajectories, creating a parallel set of sketch-based demonstrations for comparison.

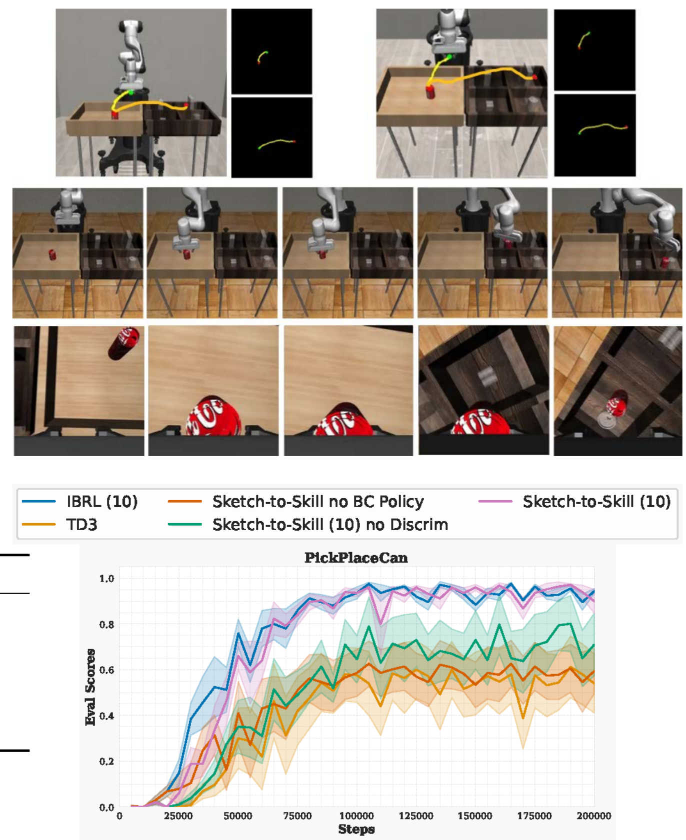


Fig. 5: Evaluation Scores (success rate) for the robomimic PickPlaceCan environment during evaluation.

Figure 5 shows the setup where the sketch-based demonstrations provide initial positional cues for the robot in the Robomimic environment. We ask users to draw separate sketches for each stage, and then combine the trajectories during open-loop servoing. Our framework utilizes RL to refine these initial cues, dynamically adjusting the robot's approach to manage both the orientation and timing necessary for successful task execution. This method proves particularly effective as it does not rely on fully detailed trajectory information from the start; instead, it uses sketches to guide the initial exploration phase of RL, simplifying the data collection process. Our sketch-to-skill framework, even with limited initial data, performs admirably in this demanding

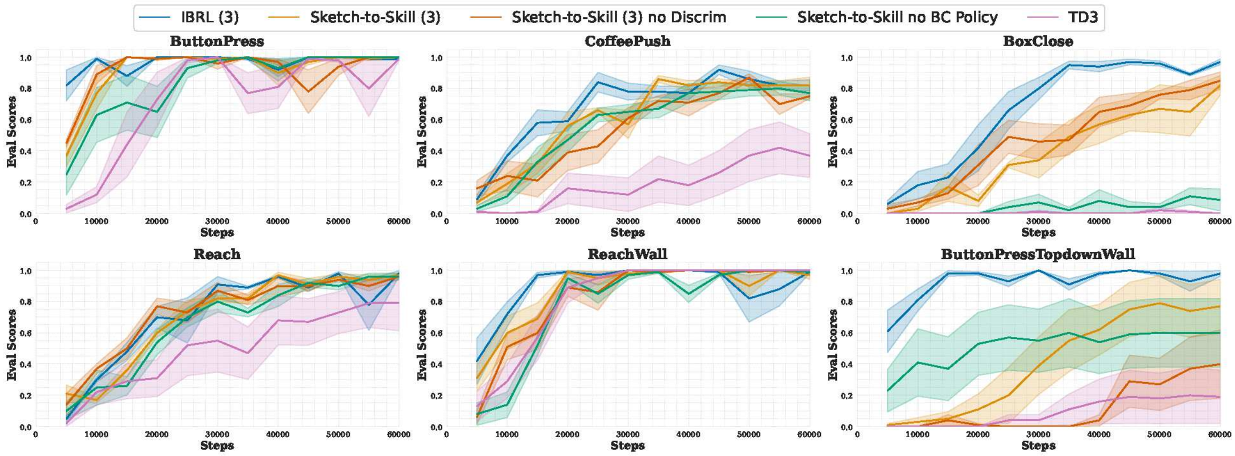


Fig. 6: **Evaluation Performance of SKETCH-TO-SKILL in MetaWorld** This figure shows the success rate across six MetaWorld tasks under randomized initial gripper and object positions. SKETCH-TO-SKILL achieves comparable performance to IBRL while surpassing pure RL.

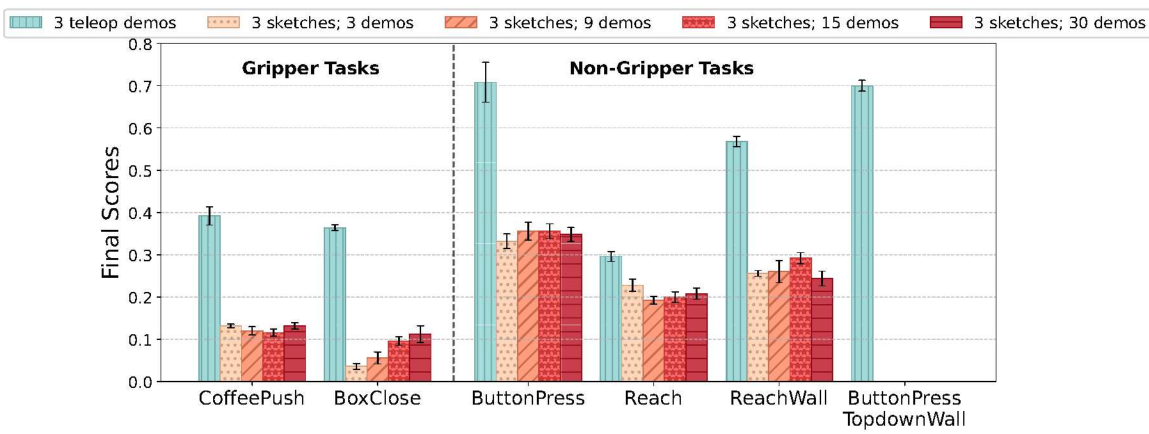


Fig. 7: **Behavioral Cloning (BC) scores using actual teleoperated data and sketch generated demonstrations**. The blue bars represent the baseline BC policy trained with 3 high-quality demonstrations, while the red bars show BC policies trained with sketch-generated demonstrations, varying in the number of demonstrations m per input sketch pair (1, 3, 5, and 10). Darker shades of red indicate an increase in the number of sketch-based demonstrations used for training. Despite poor success rate, the actual trajectories and policy learned with sketches are useful for bootstrapping as evidenced by the training performance (Figures 6).

scenario. The results are on par with those from IBRL methods $\sim$98%, which benefit from complete and detailed human demonstrations, including explicit orientation details (as shown in Figure 5).

Figures 6 show the evaluation performance across all the tasks in the MetaWorld environment. To assess the effectiveness of individual components, we also compare SKETCH-TO-SKILL's results with variants where the BC policy guidance or the discriminator reward is removed. We observe that SKETCH-TO-SKILL consistently outperforms pure RL across all tasks, often by a notable margin. Both the BC policy guidance and the discriminator reward contribute to policy learning, resulting in the most stable performance across tasks. Remarkably, SKETCH-TO-SKILL achieves performance comparable to IBRL, which relies on high-quality demonstrations, even when using only sketches as input. This is particularly evident in the `CoffeePush` and `BoxClose` tasks. These tasks require actuating the gripper — information that is not provided in the sketches. Nevertheless, SKETCH-TO-SKILL is able to bootstrap and use guidance from the sketch generated suboptimal demonstrations to learn a policy efficiently. This provides evidence to the claim that the sketch-generated demonstrations do not lead to much degradation in performance while being much easier to obtain.

Behavioral Cloning Performance. SKETCH-TO-SKILL employs behavior cloning (BC) to bootstrap policy learning, similar to IBRL. However, the key difference is that IBRL relies on high-quality teleoperated demonstrations, whereas SKETCH-TO-SKILL uses sketch-generated demonstrations. We compare the performance between them (Figure 7) and

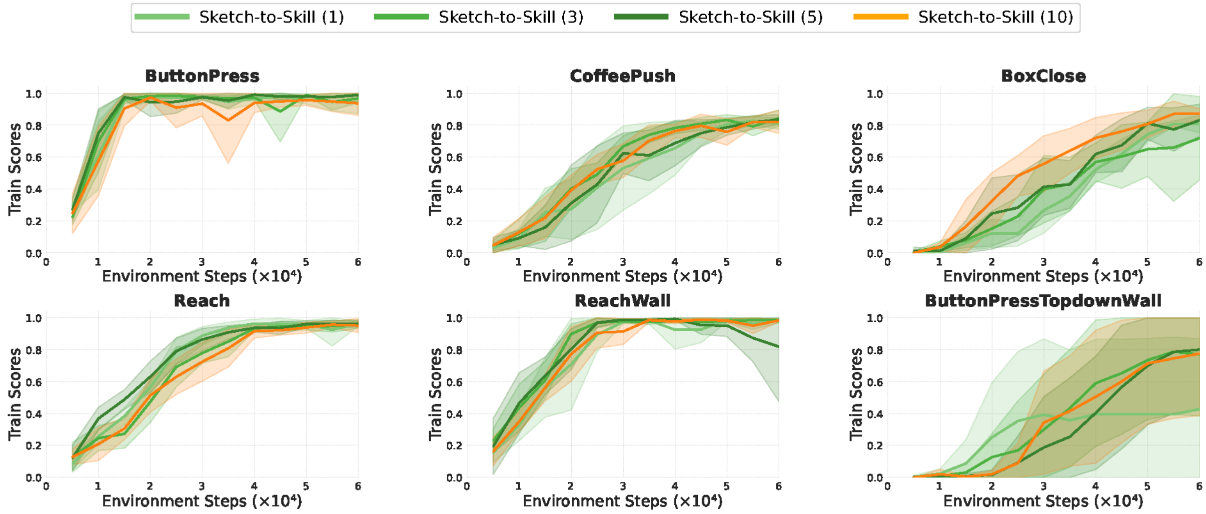


Fig. 8: This figure illustrates ablation training scores for SKETCH-TO-SKILL with varying m, the number of demonstrations generated per sketch pair (1, 3, 5, and 10).

ablate the number of generated demonstrations m per input sketch pair. Not surprisingly the BC policy with teleoperated data performs better than the sketch generated ones. However, despite the lower performance of the BC policy, SKETCH-TO-SKILL is still able to achieve comparable performance in RL training (as seen in Figures 5 and 6), showing that it is not as sensitive to the quality of the bootstrapping policy. Increasing the number of generated demonstrations m per input sketch pair (from 1 to 10) does not significantly improve the BC performance.

C. Ablation Studies

To understand the impact of the key components in SKETCH-TO-SKILL and answer Q3, we conducted ablation studies focusing on two critical aspects: the number of generated trajectories m per input sketch pair and the reward weighting scheme λ.

Impact of Generated Trajectories per Sketch. We investigated how the number of trajectories generated from each input sketch pair affects the learning performance. Figure 8, shows the learning curves for policies trained with varying numbers of generated trajectories per sketch. We observe improved performance when generating $m = 3$ trajectories per sketch instead of just one. The additional demonstrations help compensate for the lack of actual teleoperated demonstrations. However, the benefit of increasing the number of trajectories per sketch becomes less pronounced beyond a certain point. This approach is particularly useful for challenging tasks, such as `BoxClose` and `CoffeePush`, which involve precise gripper actions, but has limited impact on simpler tasks.

Effect of Reward Weighting: We examined the impact of different reward weighting schemes on policy learning. Our reward function combines the environmental reward with a discriminator-based reward by Equation 2, where λ is the weighting parameter. Figure 9 shows the learning performance across various λ values: 0.005, 0.05, 0.1, and 0.5. The model performs comparably with λ values of 0.005, 0.05, and 0.1, indicating relative insensitivity to this hyperparameter. However, it significantly underperforms when λ is set to 0.5.

D. Hardware Experiments

We validate SKETCH-TO-SKILL on physical robot hardware to demonstrate its effective transfer from simulation to real-world applications.

Experimental Setup. We set up 3 real-world tasks namely *Buttonpress, ToastPress* and *ToastPickPlace* as shown in Figure 10. We use a UR3e robot equipped with a Robotiq hand-e gripper and a realsense camera mounted on the wrist. We also use two additional environmental cameras to capture frames for humans to draw sketches on (Figure 13). The details of the task, success detection, and reset mechanism are in the Appendix.

Performance. The evaluation of our approach across multiple tasks highlights the effectiveness of sketch-generated demonstrations in training policies. For the **ButtonPress** task, the BC policy achieved a notable success rate of 80% in a randomized environment, demonstrating the robustness of the

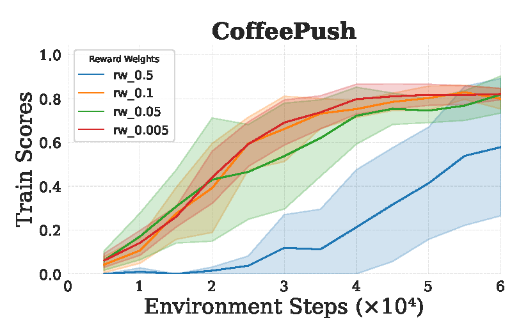
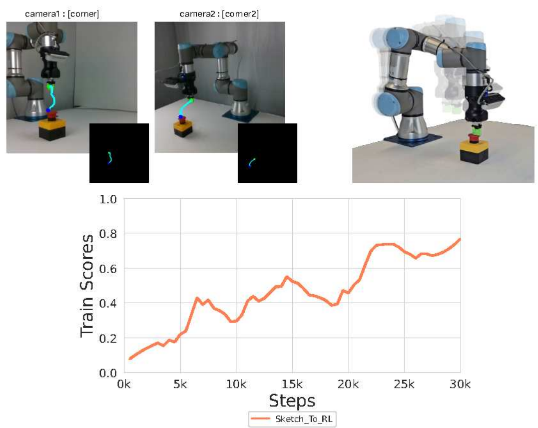
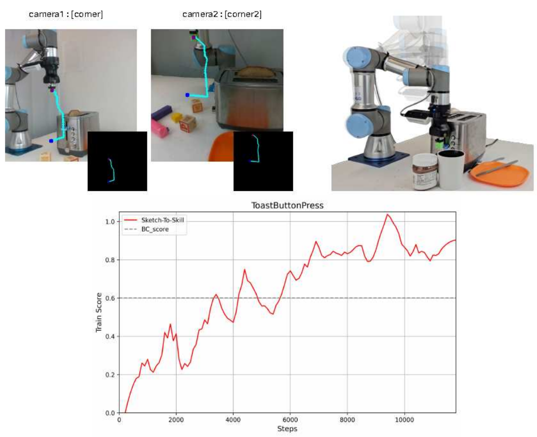


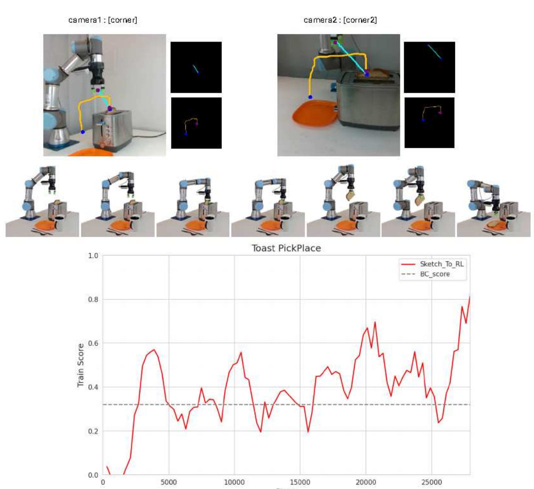
Fig. 9: Reward weighting term ablation



(a) ButtonPress.



(b) ToastButtonPress.



(c) ToastPickPlace.

Fig. 10: Training curves and hardware setups for real-world robotic tasks: ButtonPress, ToastButtonPress, and ToastPick-Place.

approach. Similarly, the Sketch-to-Skill policy, without a discriminator, reached a high training success rate of 80% within just 30,000 samples. In the **ToastPress** task, the BC policy achieved a preliminary success rate of 60% using sketch-generated demonstrations, while the Sketch-to-Skill policy significantly improved performance, achieving approximately 90% success within 10,000 interactions. Lastly, in the **Toast-PickPlace** task, the BC policy showed an initial success rate of 36% under dynamic conditions. However, the Sketch-to-Skill policy demonstrated considerable improvement, achieving approximately 80% success within 30k interactions.

V. Conclusions and Future Work

We present Sketch-To-Skill that uses 2D sketches to improve the efficiency of learning a manipulation skill. While prior work has demonstrated the utility of sketches in imitation learning (IL), we are the first to integrate them effectively within a reinforcement learning (RL) framework. The key ideas involve training a 2D sketch-to-3D trajectory generator, whose output bootstraps RL policy learning and serves as an exploration guidance signal, leading to improved efficiency.

An exciting direction for future work is to explore automated sketch generation from sources beyond human input. For example, Gu et al. [14] demonstrated how Vision-Language Models (VLMs) can generate sketches from natural language task descriptions. Integrating such models within our framework could further enhance accessibility and scalability.

VI. Limitations

While Sketch-to-Skill provides a scalable and accessible approach to bootstrapping robot learning, it is more suitable for shorter horizon tasks where the trajectories to be followed by the robot are intuitive. Our current framework demonstrates success in structured tasks, but extending it to more intricate manipulations warrants further exploration.

Towards addressing this, we have incorporated multi-step task decomposition and RL-guided refinement, improving adaptability in complex scenarios. Enhancing sketches with visual markers and temporal encoding could further refine trajectory representation, particularly for handling overlapping or ambiguous paths.

Furthermore, while sketches provide an accessible alternative to teleoperation-based demonstrations, they do not fully replace high-precision methods. Instead, they serve as an effective complement, especially in resource-constrained settings. Expanding the scope of sketch-based learning to incorporate hybrid approaches—such as combining sketch guidance with sparse teleoperation data—could enhance scalability to more sophisticated robotic applications.

References

[1] Petar Bevanda, Johannes Kirmayr, Stefan Sosnowski, and Sandra Hirche. Learning the koopman eigendecomposition: A diffeomorphic approach. In *2022 American Control Conference (ACC)*, volume 22, page 2736–2741. IEEE, June 2022. doi: 10.23919/acc53348.2022.

9867829. URL http://dx.doi.org/10.23919/acc53348.2022.9867829.

[2] Amisha Bhaskar, Zahiruddin Mahammad, Sachin R Jadhav, and Pratap Tokekar. Planrl: A motion planning and imitation learning framework to bootstrap reinforcement learning. *arXiv preprint arXiv:2408.04054*, 2024.

[3] Ayan Kumar Bhunia, Subhadeep Koley, Amandeep Kumar, Aneeshan Sain, Pinaki Nath Chowdhury, Tao Xiang, and Yi-Zhe Song. Sketch2saliency: learning to detect salient objects from human drawings. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2733–2743, 2023.

[4] A. Billard and D. Grollman. Robot learning by demonstration. *Scholarpedia*, 8(12):3824, 2013. doi: 10.4249/scholarpedia.3824. revision #138061.

[5] Joao Bimbo, Claudio Pacchierotti, Marco Aggravi, Nikos Tsagarakis, and Domenico Prattichizzo. Teleoperation in cluttered environments using wearable haptic feedback. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3401–3408. IEEE, 2017.

[6] Harish chaandar Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annu. Rev. Control. Robotics Auton. Syst.*, 3:297–330, 2020. URL https://api.semanticscholar.org/CorpusID:208958394.

[7] Linping Chan, Fazel Naghdy, and David Stirling. Application of adaptive controllers in teleoperation systems: A survey. *IEEE Transactions on Human-Machine Systems*, 44(3):337–352, 2014.

[8] Pinaki Nath Chowdhury, Ayan Kumar Bhunia, Aneeshan Sain, Subhadeep Koley, Tao Xiang, and Yi-Zhe Song. What can human sketches do for object detection? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15083–15094, 2023.

[9] Pinaki Nath Chowdhury, Ayan Kumar Bhunia, Aneeshan Sain, Subhadeep Koley, Tao Xiang, and Yi-Zhe Song. Scenetrilogy: On human scene-sketch and its complementarity with photo and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10972–10983, 2023.

[10] Carl de Boor. Package for calculating with b-splines. *SIAM Journal on Numerical Analysis*, 14(3):441–472, 1977. doi: 10.1137/0714026. URL https://doi.org/10.1137/0714026.

[11] Michael Drolet, Simon Stepputtis, Siva Kailas, Ajinkya Jain, Jan Peters, Stefan Schaal, and Heni Ben Amor. A comparison of imitation learning algorithms for bimanual manipulation. *IEEE Robotics and Automation Letters*, 2024.

[12] Federica Ferraguti, Nicola Preda, Marcello Bonfe, and Cristian Secchi. Bilateral teleoperation of a dual arms surgical robot with passive virtual fixtures generation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4223–4228. IEEE, 2015.

[13] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[14] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

[15] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

[16] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, February 2013. ISSN 1530-888X. doi: 10.1162/neco_a_00393. URL http://dx.doi.org/10.1162/neco_a_00393.

[17] Tatsuya Kamijo, Cristian C Beltran-Hernandez, and Masashi Hamaya. Learning variable compliance control from a few demonstrations for bimanual robot with haptic feedback teleoperation system. *arXiv preprint arXiv:2406.14990*, 2024.

[18] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International conference on machine learning*, pages 2469–2478. PMLR, 2018.

[19] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, October 2011. ISSN 1941-0468. doi: 10.1109/tro.2011.2159412. URL http://dx.doi.org/10.1109/tro.2011.2159412.

[20] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*. IEEE, May 2009. doi: 10.1109/robot.2009.5152577. URL http://dx.doi.org/10.1109/robot.2009.5152577.

[22] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[23] S. Mohammad Khansari-Zadeh and Aude Billard. Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765, June 2014. ISSN 0921-8890. doi: 10.1016/j.robot.2014.03.001. URL http://dx.doi.org/10.1016/j.robot.2014.03.001.

[24] Alexandros Paraschos, Elmar Rueckert, Jan Peters, and Gerhard Neumann. Model-free probabilistic movement

primitives for physical interaction. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, September 2015. doi: 10.1109/iros.2015.7353771. URL http://dx.doi.org/10.1109/iros.2015.7353771.

[25] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*, volume 6. Springer, 2002.

[26] Younggyo Seo, Danijar Hafner, Hao Liu, Fangchen Liu, Stephen James, Kimin Lee, and Pieter Abbeel. Masked world models for visual control. In *Conference on Robot Learning*, pages 1332–1344. PMLR, 2023.

[27] Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[28] Weiyong Si, Ning Wang, and Chenguang Yang. A review on manipulation skill acquisition through teleoperation-based learning from demonstration. *Cognitive Computation and Systems*, 3(1):1–16, 2021.

[29] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

[30] Priya Sundaresan, Quan Vuong, Jiayuan Gu, Peng Xu, Ted Xiao, Sean Kirmani, Tianhe Yu, Michael Stark, Ajinkya Jain, Karol Hausman, et al. Rt-sketch: Goal-conditioned imitation learning from hand-drawn sketches. 2024.

[31] Peihong Yu, Manav Mishra, Alec Koppel, Carl Busart, Priya Narayan, Dinesh Manocha, Amrit Bedi, and Pratap Tokekar. Beyond joint demonstrations: Personalized expert guidance for efficient multi-agent reinforcement learning. *arXiv preprint arXiv:2403.08936*, 2024.

[32] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

[33] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[34] Weiming Zhi, Tianyi Zhang, and Matthew Johnson-Roberson. Learning from demonstration via probabilistic diagrammatic teaching. *arXiv preprint arXiv:2309.03835*, 2023.

VII. Appendix

A. De Boor's Algorithm Details

The uniform knot vector for a B-spline of degree p with $n+1$ control points is defined as:

$$\boldsymbol{u} = [\underbrace{0, ..., 0}_{p+1}, \frac{1}{n-p+1}, \frac{2}{n-p+1}, ..., \frac{n-p}{n-p+1}, \underbrace{1, ..., 1}_{p+1}] \tag{3}$$

The B-spline basis functions $\boldsymbol{W}_{i,p}(t)$ are defined recursively using the Cox-de Boor recursion formula:

$$\boldsymbol{W}_{i,0}(t) = \begin{cases} 1 & \text{if } u_i \leq t < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\boldsymbol{W}_{i,p}(t) = \frac{t - u_i}{u_{i+p} - u_i} \boldsymbol{W}_{i,p-1}(t) + \frac{u_{i+p+1} - t}{u_{i+p+1} - u_{i+1}} \boldsymbol{W}_{i+1,p-1}(t) \tag{5}$$

where u_i are the knot values from the knot vector $\boldsymbol{u}$.

The B-spline parametrization matrix N for m evaluation points is an $m \times (n+1)$ matrix:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_{0,p}(t_1) & \boldsymbol{W}_{1,p}(t_1) & \cdots & \boldsymbol{W}_{n,p}(t_1) \\ \boldsymbol{W}_{0,p}(t_2) & \boldsymbol{W}_{1,p}(t_2) & \cdots & \boldsymbol{W}_{n,p}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{W}_{0,p}(t_m) & \boldsymbol{W}_{1,p}(t_m) & \cdots & \boldsymbol{W}_{n,p}(t_m) \end{bmatrix} \tag{6}$$

where t_j $(j = 1, ..., m)$ are evenly spaced parameters in the interval $[0, 1]$.

B. Sketch-to-3D Trajectory Generator Architecture

Overview of the Model: The proposed model converts 2D image sketches into 3D motion trajectories using a *Variational Autoencoder (VAE)* combined with a *Multi-Layer Perceptron (MLP)*. The VAE encoder processes 64×64 pixel 2D sketches (3 channels) into a latent vector ($d_{\boldsymbol{v}} = 32$), while the decoder reconstructs the sketches to retain essential features for trajectory generation. The latent space outputs the mean (μ) and variance (σ^2), sampled using the reparameterization trick.

The MLP takes the latent vectors from two sketches, concatenates them, and generates 3D control points for B-spline trajectory interpolation. The MLP takes an input of size ($d_{\boldsymbol{v}} \times 2$), processes it through hidden layers [1024, 512, 256], and outputs $n_{cp} \times 3$. The generated 3D control points are then used for B-spline interpolation to produce smooth trajectories.

Initialization, Regularization, and Hyperparameters: We initialize all network parameters using *Xavier initialization*. Regularization is done with *Kullback-Leibler Divergence (KLD)*, using a loss function that combines *Sketch Reconstruction Loss*, *KLD Loss*, and *Trajectory Loss*. The *Sketch Reconstruction Loss* is the MSE loss on sketch images, the *Trajectory Loss* computes the MSE between predicted and ground truth trajectories, where the ground truth trajectory is first densely fitted and resampled to ensure uniform point spacing, and the *KLD Loss* applies Kullback-Leibler Divergence for regularization. Key hyperparameters include:

- **Image size:** 64×64 pixels
- **Latent dimension:** 32
- **Number of control points:** 20
- **B-spline degree:** 3
- **MLP hidden layers:** [1024, 512, 256]
- **Learning rate:** 1×10^{-3} (Adam optimizer)
- **Batch size:** 128
- **KLD weight:** 0.0001 (with optional annealing)
- **Training epochs:** 200

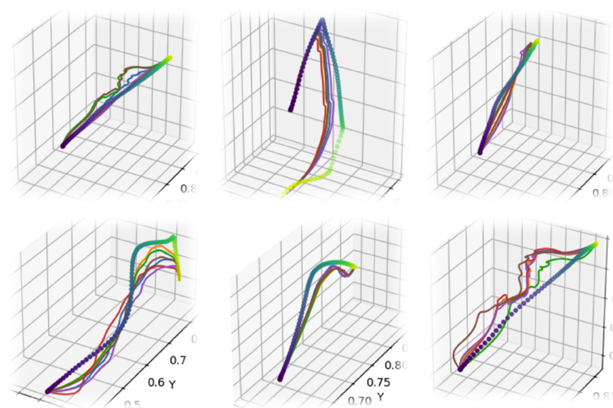


Fig. 11: Diversity in generated 3D trajectories. Each subplot shows multiple generated trajectories (colored lines) for a single input, demonstrating variability. Scattered points represent the ground truth trajectory.

To enhance robustness and generalization, our training process employs two concurrent data augmentation strategies. The first applies diverse image augmentations (rotations, scaling, affine transformations, noise) to input sketches, used exclusively for updating the VAE to learn robust sketch representations. The second strategy targets potential mismatches in hand-drawn sketches by subtly modifying both original sketches and their 3D trajectories. This involves adding noise and minor elastic deformations to sketches, and noise with refitting to trajectories. These augmented pairs update the entire model, preparing it for hand-drawn input variability while maintaining sketch-trajectory consistency. This augmentation approach enhances the model's ability to handle diverse, imperfect sketches while ensuring accurate 3D trajectory generation in real-world scenarios. To train the sketch generation network we collect 22000 samples for simulation tasks and 85 for the real world. During the training for the real-world task, we use the data collected from simulation to train the VAE part as well. We provide additional visualizations to demonstrate the learned model's generalizability in Figure 11 and 12.

C. Implementation Details and Hyperparameters of Sketch-To-Skill Policy and baselines

This section outlines the implementation details of Sketch-To-Skill and the baselines. The behavior cloning (BC) policies utilize a ResNet-18 encoder, where the output is flattened and processed by MLPs to produce final 4D

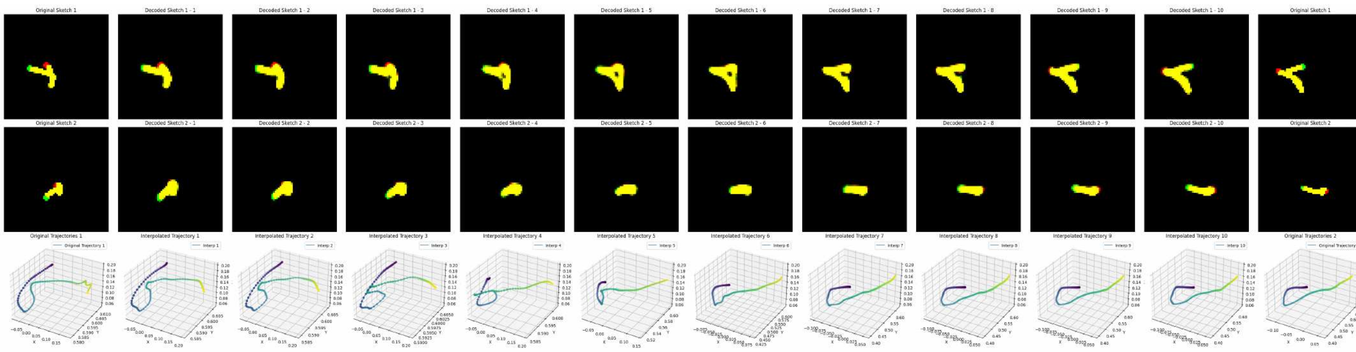


Fig. 12: Latent space interpolation results showing smooth transitions between original samples (leftmost and rightmost) and reconstructed samples (middle) from two viewpoints, demonstrating the model's ability to generate coherent 3D trajectories from 2D sketch pairs.

actions. We replace the BatchNorm layers in ResNet with GroupNorm, matching the number of groups to the input channels. To prevent overfitting, we employ random-shift data augmentation.

In the Meta-World environment, we utilize a corner2-image camera setup. We use wrist cameras to enhance generalization and sample efficiency in real-world experiments, specifically using them for the ButtonPress task.

TABLE II: Hyperparameters for RL in SKETCH-TO-SKILL.

Parameter	Meta-World	Real-World
Optimizer	Adam	Adam
Learning Rate	1e-4	1e-4
Batch Size	256	256
Discount (γ)	0.99	0.99
Exploration Std. (σ)	0.1	0.1
Noise Clip (c)	0.3	0.3
EMA Update Factor (ρ)	0.99	0.99
Update Frequency (U)	2	2
Actor Dropout	0.5	0.5
Q-Ensemble Size (E)	2	N/A
Num Critic Update (G)	1	N/A
Image Size	96×96	96×96
Use Proprio	No	N/A
Proprio Stack	N/A	N/A
State Stack	N/A	N/A
Action Repeat	N/A	N/A

D. Additional Details of Real-world Experiments

In this section, we present insights into the real-world experiments conducted using SKETCH-TO-SKILL. We utilized a UR3e robot equipped with a Robot Hand gripper, operating in an action space with 4 dimensions: 3 for end-effector position deltas under a Cartesian impedance controller and 1 for the absolute gripper position, with policies functioning at 7.5 Hz.

To train the Sketch-To-3D Trajectory generator, we collected approximately 85 teleoperated trajectories. RGB images were captured using two orthogonally positioned RealSense cameras to enhance trajectory insight. A green marker was placed on the gripper tip to facilitate sketch generation on the frames, enabling the model to learn 2D sketch projections onto 3D trajectories. All methods maintained the same hyperparameters and network architectures as those used in the Meta-World tasks.

1) Common Implementation Details:

- **Reset Protocol:** For all tasks, we manually reset the environment between episodes by returning objects to their initial states and randomizing their positions. The robot is initially set to a specific joint configuration known as the home position. Whenever the agent receives a reward or an episode ends, it resets back to the home configuration and the training continues. The object randomization typically begins with placement at the center, remaining unchanged until the agent receives its first reward. After that, the object is gradually moved towards the boundary, circled around the workspace, returned to the center, and the process is repeated until the training ends.
- **Safety Boundaries:** We have restricted the movement of the robot to the x, y, and z directions of the end-effector. Each step that the robot takes is limited to a specific value. If the action taken by the agent exceeds this limit, the robot will not move and will remain in its current position. To avoid collision with the workspace surface, we have set a minimum limit for the z direction of the robot. Even if the agent attempts to move downward beyond this threshold, the robot will remain at the specified z position.
- **Reward Structure:** We used a manual method to reward the agent if it successfully completes the task. The agent receives a reward of 1 for successfully completing the task and a reward of 0 in all other cases. Each episode has a set number of timesteps, and if the agent doesn't succeed within that limit, it resets and starts over. The length of each episode may be shorter than the given limit depending on how quickly the agent completes the task.

2) Task-Specific Details:

- **Button Press Task:** After training the Sketch-To-3D Trajectory generator, we created 30 sketches based on RGB frames from the two cameras. We then collected

30 sketch-generated demonstrations, ξ_D using openloop servoing on 3D trajectory ξ_g produced by the generator $\boldsymbol{T}$. We then train Behavior Cloning (BC) policy using the sketch-generated demonstrations, ξ_D, achieving a score of 0.8 and thus leading to good performance in the SKETCH-TO-SKILL policy. The button position was randomized within a 20cm by 20cm to 25cm trapezoidal area visible from the wrist camera.

- **Toaster Press Task:** The Toaster Press experiment was designed to test the robustness of our approach in cluttered environments. We created 10 sketches based on the camera feeds and collected 10 sketch-generated demonstrations using open-loop servoing. Each episode featured a cluttered environment around the toaster, with objects commonly found in household settings to simulate realistic conditions. The initial position of the gripper was randomized in every episode to test the adaptability of the learned policy. Training a Behavior Cloning (BC) policy using the sketch-generated demonstrations achieved a preliminary success rate of 60%, and SKETCH-TO-SKILL training over 12k interactions achieved ∼90% success rate within 10k interactions.
- **Bread Pick and Place Task** This task involved picking a piece of bread from a toaster and placing it on a nearby plate, requiring precise manipulation and handling. Sketches were specifically collected in randomly cluttered environments to reflect typical variability in real-world scenarios. Both the environment clutter and the initial gripper positions were randomized in each episode, presenting a different challenge each time to test the robustness of the policy. The BC policy achieved a success rate of 36% and SKETCH-TO-SKILL trained over 30K interactions achieved ∼80% success within 30K interactions.

These experiments underscore our method's capability to handle real-world variability and complex task execution, supporting its potential utility beyond controlled experimental setups. The detailed results from these tasks, illustrated in Figures 16 and 17, highlight the practical applications and adaptability of our SKETCH-TO-SKILL framework in dynamic, cluttered settings.

E. Additional Details of Robomimic Experiments

1) Task Complexity: The PickPlaceCan task in RoboMimic is an demanding two-stage challenge. It requires the robot to accurately locate and reach a can, then pick it up and place it into a designated bin. This task not only tests the robot's ability to handle objects with precision but also demands correct orientation of the gripper throughout the process. RoboMimic, a well-established benchmark, provides high-quality demonstrations collected via human teleoperation, which are instrumental for training successful policies.

2) Implementation and Strategy: In this setup, the sketch-based demonstrations provide initial positional cues for the robot. Our framework utilizes reinforcement learning (RL) to refine these initial cues, dynamically adjusting the robot's approach to manage both the orientation and timing necessary for successful task execution. This method proves particularly effective as it does not rely on fully detailed trajectory information from the start; instead, it uses the sketches to guide the initial exploration phase of RL, significantly simplifying the data collection process.

3) Results and Performance Metrics:

- **Performance:** Our SKETCH-TO-SKILL framework, even with limited initial data, performs admirably in this demanding scenario. The results are on par with those from IBRL methods 98%, which benefit from complete and detailed human demonstrations, including explicit orientation details (as shown in Figure 15).
- **Comparison to IBRL:** The comparative success illustrates the robustness and effectiveness of our method in managing the task's orientation and other complexities without fully specified trajectory inputs. This is particularly notable given that RoboMimic's Pick and Place Can task offers a significantly higher level of difficulty than similar tasks in MetaWorld.

F. Additional Experiments in Metaworld Assembly

1) Task Complexity: The Assembly task in MetaWorld is a "hard" task [26] that requires the robot to execute two-stage manipulations. The task involves precise movements to pick up a peg, navigate it to a specific location, and insert it into a hole within a larger assembly fixture. This task tests the robot's precision, spatial awareness, and ability to handle complex sequences.

2) Results and Performance Metrics:

- SKETCH-TO-SKILL: Demonstrated a high success rate of 93%, effectively using sketches for coarse guidance to navigate and complete complex task sequences, even without actual teleoperated demonstrations.
- SKETCH-TO-SKILL without Discriminator: Initially struggled with a success rate of 20%, but extended interaction up to 100K steps improved performance dramatically, reaching a near-perfect success rate of 98%. This underscores the potential for learning even without discriminator guidance, given sufficient training time.
- IBRL: Achieved the highest success rates of approximately 100%, benefiting from high-quality teleoperated demonstrations that include precise details on orientation and positioning.
- Standard BC: Achieves a success rate of around 60%, showing limitations in environments where adaptive behaviors and fine-tuning through reinforcement learning are necessary.

These results highlight the effectiveness of the SKETCH-TO-SKILL approach in handling complex, multi-stage tasks through initial coarse guidance, with the potential for significant improvement over time. They also illustrate the advantage of incorporating discriminator feedback to accelerate learning and enhance performance.

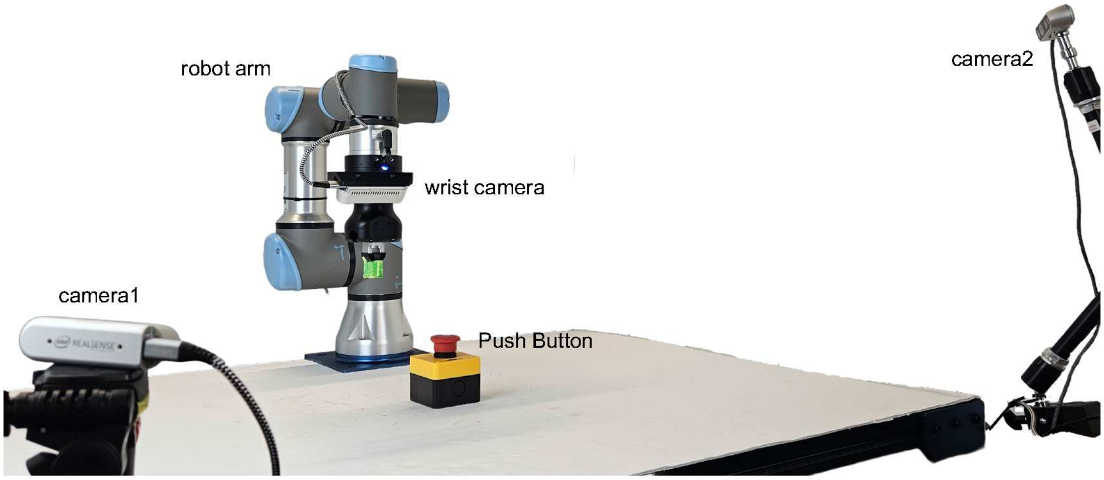


Fig. 13: Complete setup for the ButtonPress task in a real-world experiment. The configuration includes a UR3e robot arm equipped with a Robot Hand gripper, and a RealSense D435i camera mounted on the wrist. Two additional RealSense cameras are positioned orthogonally to capture the trajectory from two different viewpoints.

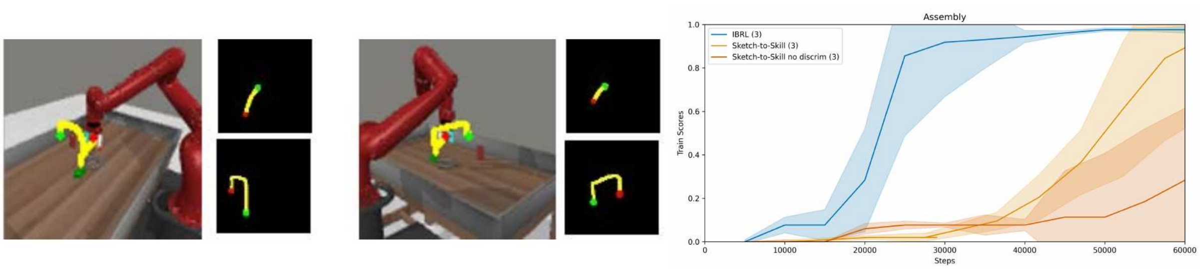


Fig. 14: Left: sketches for the MetaWorld Assembly task. Right: training scores (success rate).

G. Using Color Gradients for Overlapping Trajectories

We can also incorporate time-parameterization of the trajectory in the sketches using a color gradient, instead of a binary sketch image. This notion of color gradient in sketches was introduced by RT-Trajectory [14]. An example is shown in Figure 15. Here, the color of the sketch changes from green (start) to red (end). This is particularly helpful when the sketch crosses each other or overlaps. We conduct additional experiments with the generator to evaluate the effect of incorporating gradients in the Sketch-To-3D trajectory generator.

Specifically, we use the MetaWorld Assembly task (Figure 14) where the sketch overlaps in the middle as the end-effector picks up the tool and carries it to the goal position. We trained two generators without and with color gradients. The performance of these two generators are reported in Table III. We observe that incorporating the gradient in such a case results in lower reconstruction and trajectory losses. With lower trajectory losses we can handle more complex trajectories that overlap with color gradients. Note that incorporating gradients also does not require any change to the downstream architecture, and only requires minimal changes to the generator architecture.

TABLE III: Performance Metrics for Overlapping and Non-Overlapping Trajectories

Metric	with color gradient	without color gradient
Training Loss	**0.2438**	0.3107
Validation Loss	**0.3120**	0.3585
Reconstruction Loss	**0.0001**	0.0003
KLD Loss	**77.167**	76.4683
Parameter MSE Loss	**0.1028**	0.1135
Trajectory loss	**0.1530**	0.2379

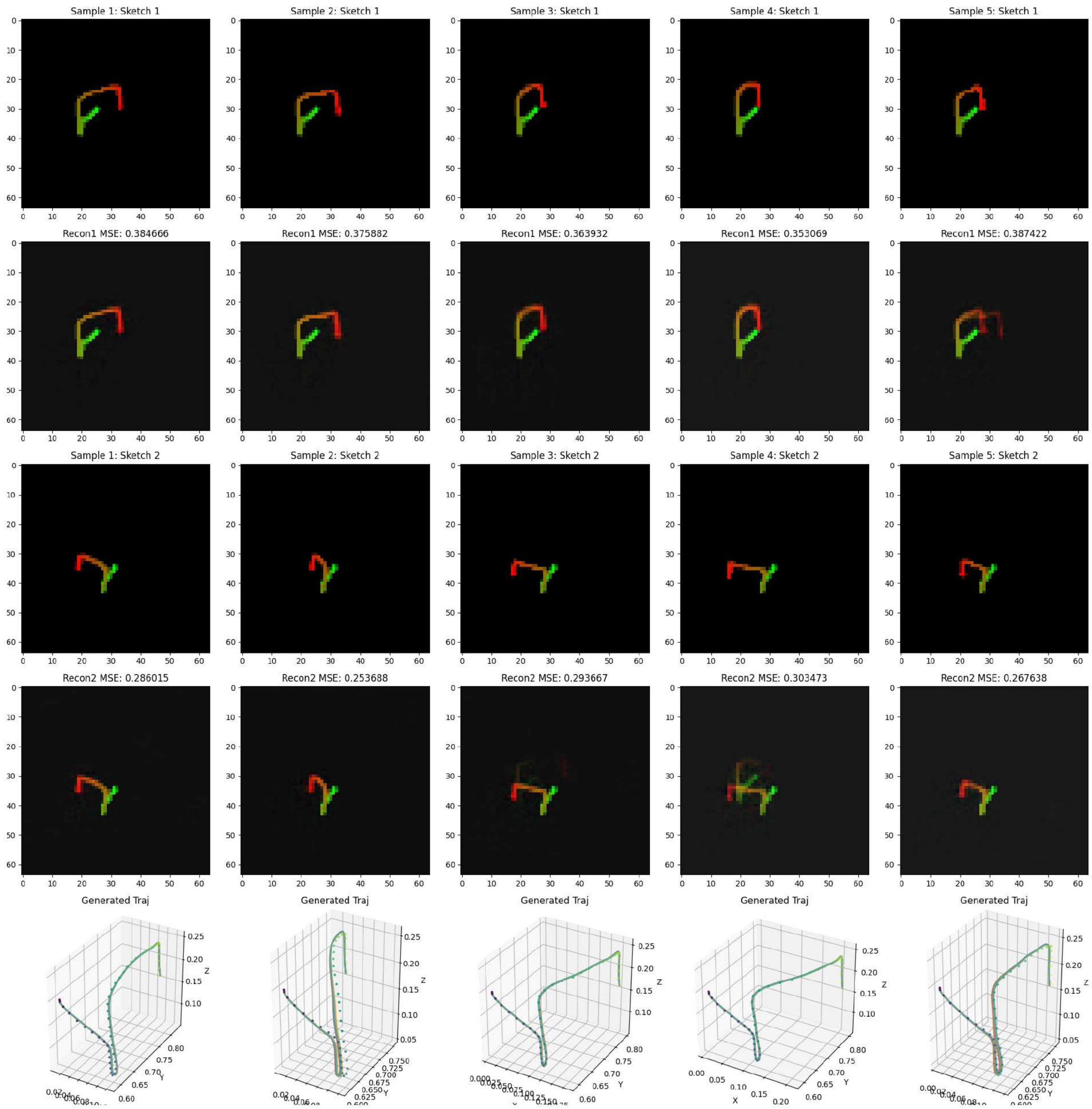


Fig. 15: Trajectories with time-based color gradient for Assembly Metaworld simulation task.