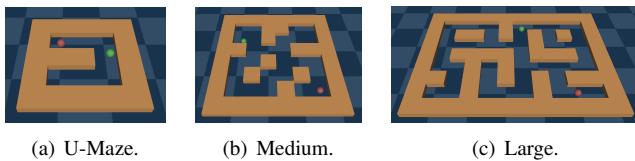


# Implementation Details and Additional Results of LTLDoG

Zeyu Feng<sup>†1</sup>, Hao Luan<sup>†1</sup>, Pranav Goyal<sup>1</sup>, and Harold Soh<sup>1,2</sup>

## A. Data

1) *Maze2d*: The Maze2d environment is part of a well-known open-source benchmark for offline reinforcement learning — D4RL [1]. Datasets of trajectories are publicly available in the D4RL library. Each of the trajectories within is one continuous trajectory of an agent navigating to random goals without interruption. See Fig. 1 for an overview of the environments. We utilize the original dataset in D4RL as is to train the baseline DIFFUSER, which also serves as the diffusion model of LTLDoG.



(a) U-Maze. (b) Medium. (c) Large.

Fig. 1. Maze2d environments. The agent is tasked to move force-actuated ball to goals (red circle) from random start locations (green ball). There are three kinds of maze configurations with increasing difficulties and all of them are long-horizon path planning tasks with continuous states and actions.

2) *PushT*: PushT is a robot manipulation benchmark adopted in DIFFUSION POLICY [2] to showcase diffusion models' planning capabilities. A dataset of expert policies for pushing a T block to the same goal position starting from random poses are publicly available at [2]. Though DIFFUSION POLICY achieves good performance on the original manipulation task, trained with around only 200 expert demonstrations (*i.e.*, 206 episodes), we find that these trajectories lack the modalities or abundant behaviors that potentially or partially satisfy different  $LTL_f$  constraints. As such, expanding the original dataset is necessary for our method to work.

**Data Augmentation** Given that obtaining extra expert demonstrations is inefficient, we performed common data augmentation techniques to expand the original dataset. Concretely, we first employed simple geometric augmentation, only rotation and flipping, for each trajectory in the original dataset, assuming basic physical symmetry in this simulation environment. Then, we used a pretrained DIFFUSION POLICY to push the T block to the goal pose, aligning with the original dataset in which the goal of the manipulation task is implicitly encoded in trajectories. In this way, we augmented

the original set from 206 trajectories to 6686 trajectories, and elongated the average episode length from 125 to 289 steps. The augmented dataset is made public<sup>1</sup> together with this appendix.

In this task, the baseline (DIFFUSION POLICY) and our methods share the same diffusion model backbone, which is trained over the augmented dataset.

3) *Real Robot Scenarios*: Data for both of the real-world scenarios are collected in simulated environments. For the office we collected 4 long continuous trajectories in the simulation of step size of 11,000, 107,000, 60,000 and 130,000. Similarly for the lab map we collected 4 trajectories of steps 11,000, 19,000, 110,000 and 120,000. First, we built by SLAM the maps of the two environments, a studio-like lab and an office room, and the maps are used to create two simulation scenarios. For creating the maps, we teleoperated a quadruped robot (Unitree Go2) mounted with LiDAR in each of these environments. We used the `slam_toolbox` along with Nav2 for generating the maps in both environments. We created simple 3D world for each of the maps for simulation in gazebo. After that, we enabled a Go2 robot simulated in Gazebo and driven by ROS Navigation Stack to navigate to random goal positions sampled in the maps for data collection, creating long trajectories in a similar fashion to Maze2d.

4) *LTL Satisfaction Values*: The soft labels (values indicating how well the  $LTL_f$ s are satisfied) for training the regressor guide (LTLDoG-R) are calculated with an adapted version of the DTL toolbox [3] and saved before each training session, and can be reused if possible, since each training session may sample and cut different segments of the trajectories in datasets based on different training configurations. Values calculation for benchmarking test results are performed online simultaneously with all tests.

## B. $LTL_f$ Generation

To generate practical  $LTL_f$ s meaningful for real robotic tasks, we query GPT-4 via prompting with some constraints on  $LTL_f$ s. The prompt and the response from ChatGPT is listed in TABLE I. In the real experiments, we tested the first four returned  $LTL_f$ s as these are also commonly used in literature [4]–[6]. Some other generated  $LTL_f$ s shown in this table are not logically correct or meaningful for robotic tasks. For instance, the  $LTL_f$ s with imply operator do not enforce any constraints if the condition is not met (*e.g.*, the condition in the 6-th  $LTL_f$  can never be satisfied for disjoint regions).

<sup>†</sup>Equal contribution.

<sup>1</sup>All authors are with the Department of Computer Science, School of Computing, National University of Singapore, Singapore. {zeyu, haoluan, pgoyal, harold}@comp.nus.edu.sg

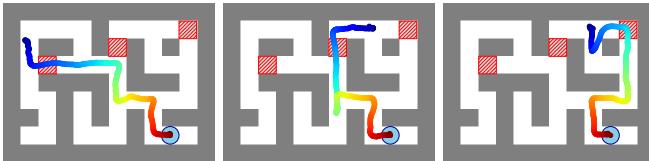
<sup>2</sup>H. S. is also with the Smart Systems Institute, National University of Singapore.

<sup>1</sup><https://github.com/clear-nus/ltdog>

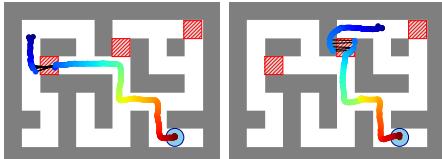
TABLE I  
PROMPTS AND RESPONSES FROM CHATGPT (GPT-4)<sup>1</sup> FOR LTL<sub>f</sub> GENERATION.

<b>Prompt</b>	<p>In a robot navigation task, the robot is required to visit four key regions (referred to as p_0, p_1, p_2 and p_3) in specific temporal orders. The constraints on the robot's trajectory concerning the visitation (or non-visitation) of these regions can be specified using finite Linear Temporal Logic (LTLf). Please generate several LTLf formulas (e.g., 10) that are realistic in real-world environments and tasks involving robot navigation. The details of the task and additional requirements are as follows:</p> <ol style="list-style-type: none"> <li>1. The LTLf operators that can be used are: <math>\neg</math> (Negate), <math>\wedge</math> (And), <math>\vee</math> (Or), F (Eventually), G (Always), U (Until). Avoid using other LTL operators. Also, avoid using X (Next) since the number of timesteps needed to move from one region to another is unknown.</li> <li>2. The atomic propositions are determined by whether the robot is inside p_0, p_1, p_2 and p_3. Therefore, the labelling function returns assignments with a 4-dimensional 0-1 vector at each timestep of the robot's trajectory.</li> <li>3. Only one proposition can be true at each timestep because these regions are disjoint. An assignment at a timestep can be a zero vector (0, 0, 0, 0), indicating that the robot is not currently in any of these regions. This occurs when the robot is initially not in these regions and all atomic propositions are False.</li> <li>4. The robot cannot move directly from one region to another as the regions might not be connected. Therefore, LTL formulas such as p_0 U p_1 are not valid since both propositions will be False when the robot is not inside these two regions.</li> <li>5. The robot's trajectory is finite, so the robot cannot repeatedly visit different regions.</li> <li>6. The LTLf formulas are supposed to apply from the first timestep of the assignment.</li> <li>7. When generating an LTLf formula, please analyze the generated formula first to ensure that it is valid and that there exist assignments that can satisfy it.</li> </ol>
<b>ChatGPT</b>	<p>To create realistic LTLf formulas for a robot navigation task considering the specified constraints and operators, we need to structure these formulas in a way that reflects possible navigation scenarios. Here are ten LTLf formulas that align with the given requirements:</p> <ol style="list-style-type: none"> <li>1) Visit all regions at least once: <ul style="list-style-type: none"> <li>• <math>F(p_0) \wedge F(p_1) \wedge F(p_2) \wedge F(p_3)</math> This formula states that eventually, the robot will visit each of the four regions at least once during its trajectory.</li> </ul> </li> <li>2) Visit regions in a specific sequence: <ul style="list-style-type: none"> <li>• <math>F(p_0 \wedge F(p_1 \wedge F(p_2 \wedge F(p_3))))</math> This formula requires the robot to visit p_0, then p_1, followed by p_2, and finally p_3, in that order.</li> </ul> </li> <li>3) Avoid a specific region until another has been visited: <ul style="list-style-type: none"> <li>• <math>\neg p_1 \cup p_0</math> Here, the robot must avoid p_1 until p_0 has been visited.</li> </ul> </li> <li>4) Never visit a specific region: <ul style="list-style-type: none"> <li>• <math>G(\neg p_3)</math> This formula specifies that the robot should never enter region p_3.</li> </ul> </li> <li>5) Visit one region only after visiting two others: <ul style="list-style-type: none"> <li>• <math>F(p_0 \wedge F(p_1)) \Rightarrow F(p_2)</math> The robot can visit p_2 only after it has visited p_0 and then p_1.</li> </ul> </li> <li>6) Ensure two regions are visited before a third: <ul style="list-style-type: none"> <li>• <math>F(p_0 \wedge p_1) \Rightarrow F(p_2)</math> This formula ensures that p_2 and p_1 are visited (in no particular order) before visiting p_2.</li> </ul> </li> <li>7) Visit at least three regions: <ul style="list-style-type: none"> <li>• <math>F(p_0) \vee F(p_1) \vee F(p_2) \vee F(p_3)</math> This ensures the robot visits at least three out of the four regions. It is a less strict requirement, allowing for flexibility in navigation.</li> </ul> </li> <li>8) Visit a specific region last: <ul style="list-style-type: none"> <li>• <math>F(p_0 \wedge F(p_1 \wedge F(p_2))) \wedge G(\neg p_3 \cup (p_0 \wedge F(p_1 \wedge F(p_2))))</math> The robot must visit p_3 last, after visiting p_0, p_1, and p_2 in any order.</li> </ul> </li> <li>9) Avoid a region until all other regions have been visited: <ul style="list-style-type: none"> <li>• <math>\neg p_3 \cup (F(p_0) \wedge F(p_1) \wedge F(p_2))</math> The robot should avoid p_3 until it has visited p_0, p_1, and p_2 in any order.</li> </ul> </li> <li>10) Visit a pair of regions, avoiding others until then: <ul style="list-style-type: none"> <li>• <math>\neg(p_2 \vee p_3) \cup (p_0 \wedge F(p_1))</math> The robot must avoid p_2 and p_3 until it has visited p_0 and then p_1.</li> </ul> </li> </ol> <p>Each of these formulas considers the finite nature of the robot's trajectory, the impossibility of being in multiple regions simultaneously, and the requirement that the robot may need to travel through unspecified areas (not within p_0, p_1, p_2, or p_3) to move from one region to another. The formulas are structured to reflect realistic navigation tasks that could be encountered in real-world robot navigation scenarios, such as search and rescue, surveillance, or delivery tasks within a specified area.</p>

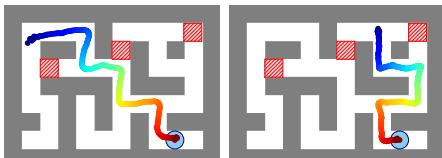
<sup>1</sup> <https://chat.openai.com/share/43670685-3cb9-4496-ad3e-1ee70cd358b8>.



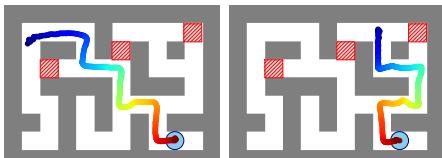
(a) Trajectories from DIFFUSER.



(b) Trajectories from SAFEDIFFUSER.



(c) Trajectories from LTLDog-S.



(d) Trajectories from LTLDog-R.

Fig. 2. Results of safe planning in Maze2d-Large with unsafe regions. There are three unsafe blocks (red squares, labeled  $p_L$ ,  $p_M$ ,  $p_R$  from left to right) that need to be avoided during navigation to the goal (shaded circle). The LTL<sub>f</sub> constraint for this task is  $\varphi = \square\neg(p_L \wedge p_M \wedge p_R)$ . (a) Trajectories from DIFFUSER ignore safety and can violate the specified constraints. (b) SAFEDIFFUSER produces discontinuous trajectories (the constraints here are slightly reshaped to accommodate the CBF function). (c, d) Our LTLDog is able to plan trajectories that detours around the obstacles to successfully arrive at the goal.

### C. Additional Illustrations

More illustrations and comparisons against the baseline for the obstacle avoidance task in Maze2d environment are in Fig. 2.

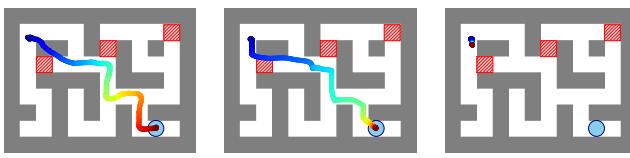


Fig. 3. Example of failure cases from LTLDog-S.

Fig. 4–Fig. 6 delineate two more examples of the comparisons between our method LTLDog against the baseline DIFFUSION POLICY on the obstacle avoidance and temporal behavior tasks. These results demonstrate our method’s capabilities in guiding diffusion models to follow not only *purely spatial* but also *spatial-temporal* constraints.

TABLE II  
RESULTS OF GENERATED LTL<sub>f</sub>S IN SIMULATION ROOMS.

Task	Method\Performance	Success rate (%) ↑	
		Lab	Office
Obstacle avoidance	DIFFUSER	11.7±3.7	0.0±0.0
	LTLDoG-S	100.0±0.0	80.0±8.1
Temporal behaviors	DIFFUSER	0.0±0.0	0.0±0.0
	LTLDoG-S	96.5±1.5	66.7±1.8

TABLE III  
COMPARISON OF CONTROLLERS<sup>1</sup> IN AVOIDANCE TASKS IN MAZE2D

Method\Perf.	Success rate (%) ↑		Unconstrained score <sup>2</sup> ↑		
	Planning	Rollout		C1	C2
		C1	C2		
DIFFUSER	9.5±3.1	11.0±1.7	15.0±2.6	142.2±5.1	92.1±3.0
SAFEDIFFUSER	99.4±0.9	12.3±3.5	14.2±2.8	135.8±5.2	131.9±5.4
LTLDoG-S	99.0±0.8	<b>73.0±3.0</b>	<b>90.6±0.6</b>	97.3±2.8	36.2±11.7
LTLDoG-R	98.8±0.8	<b>92.0±1.4</b>	<b>99.6±0.6</b>	127.1±5.1	76.4±13.4

<sup>1</sup> Planned trajectories are rolled out using two different low level control strategies, denoted as C1 and C2.

<sup>2</sup> Unconstrained scores do not take unsafe penalties into account.

We also show in Fig. 7 more snapshots of a quadruped robot running in the simulated Lab environment with different policies (baseline against ours). The overall satisfaction rate of all raw generated trajectories in different tasks and environments are summarized in Table II. The overall performance is  $85.8 \pm 14.0\%$  (*c.f.* baseline  $2.9 \pm 5.4\%$ ) as reported in the main paper.

### D. Failure case study

Fig. 3(a) shows an infeasible trajectory generated from LTLDog-S. Our hypothesis is that posterior sampling in LTLDog-S calculate gradients based on Euclidean distance on the trajectory space due to the LTL evaluation, which may push the trajectory away from the data manifold. On the other hand, the neural network in regressor guidance is trained on noised intermediate samples, which are linked to the data manifold. Gradients ascent on noise space will likely push the noised example to a place where both the satisfaction value and likelihood are high. Visualizations of the difference between two controllers as shown in Fig. 3(b) Fig. 3(c) suggest that selecting next locations sequentially without distance checking is a less safe option.

**Low-Level Control Policy** The success rate during rollout also depends on the specific controller used to visit each location of the generated trajectory. TABLE III shows the comparative results between the two low-level control policies. A more conservative controller (denoted as C2) that visits every location exactly with distance checking can enjoy higher success rate than greedily move (C1, originally used in [7]).

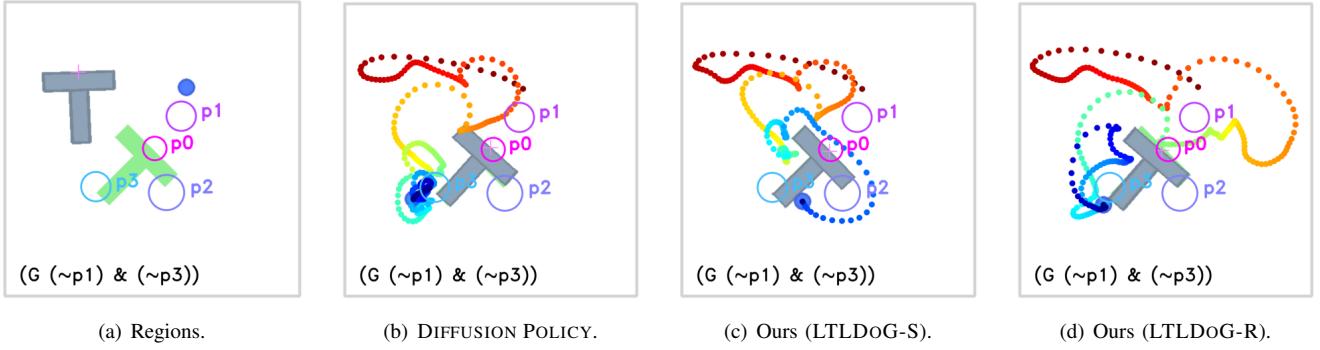


Fig. 4. PushT task. Diffusion policy generates next locations of the mover to push the T block by following these locations. Atomic propositions for LTLs are defined by constraints over actions.

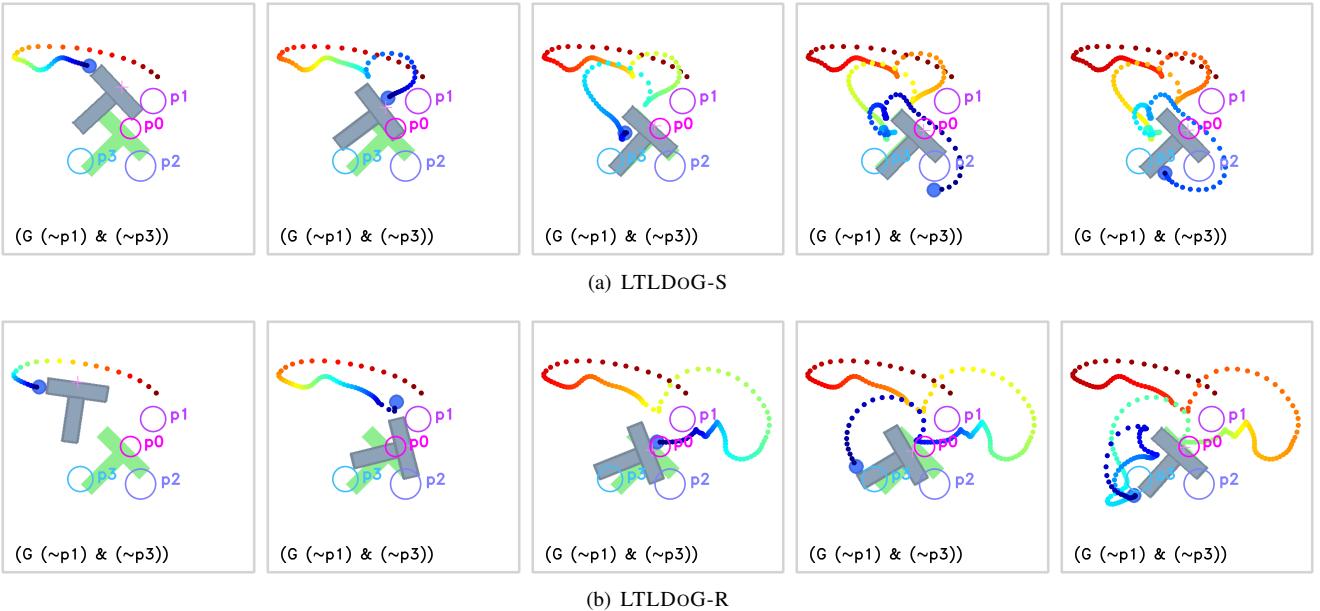
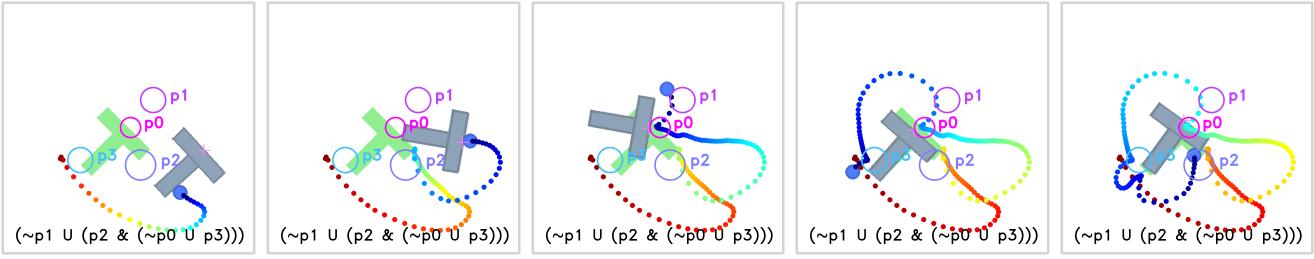


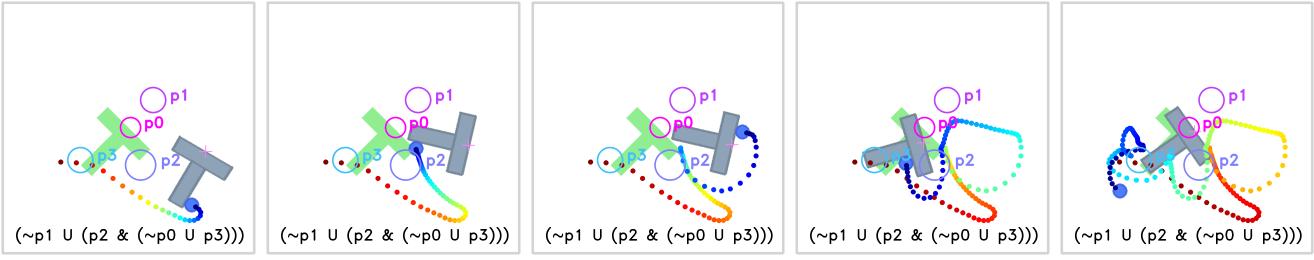
Fig. 5. Snapshots of two successful rollouts of the two variants of our method accomplishing the same task in Fig. 4. The mover (circle filled in blue) should avoid region  $p_1$  and  $p_3$  at all time, as designated in the LTL<sub>f</sub> “ $\square(\neg p_1 \wedge \neg p_3)$ .” Fig. 5(a) and Fig. 5(b) showcase that LTLDG-S and LTLDG-R exploit different modalities to complete the original manipulation task while respecting the constraint specified by the LTL<sub>f</sub>.

## REFERENCES

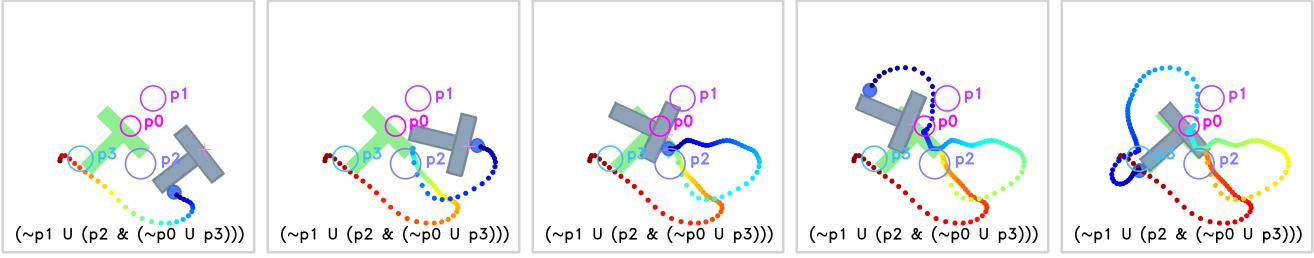
- [1] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, “D4RL: Datasets for deep data-driven reinforcement learning,” *arXiv preprint arXiv:2004.07219*, 2020.
- [2] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, “Diffusion policy: Visuomotor policy learning via action diffusion,” in *Proc. Robot.: Sci. and Syst. (RSS)*, 2023.
- [3] Z. Xu, Y. S. Rawat, Y. Wong, M. Kankanhalli, and M. Shah, “Don’t pour cereal into coffee: Differentiable temporal logic for temporal action segmentation,” in *Advances in Neural Inf. Process. Syst.*, 2022.
- [4] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, “Teaching multiple tasks to an RL agent using LTL,” in *Proc. Int. Conf. Autonomous Agents Multiagent Syst.*, 2018, pp. 452–461.
- [5] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, “LTL and beyond: Formal languages for reward function specification in reinforcement learning,” in *Int. Joint Conf. Artif. Intell.*, 2019, pp. 6065–6073.
- [6] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. McIlraith, “LTL2Action: Generalizing LTL instructions for multi-task RL,” in *Int. Conf. Mach. Learn.*, 2021, pp. 10497–10508.
- [7] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine, “Planning with diffusion for flexible behavior synthesis,” in *Int. Conf. Mach. Learn.*, vol. 162, 2022, pp. 9902–9915.



(a) DIFFUSION POLICY (baseline)

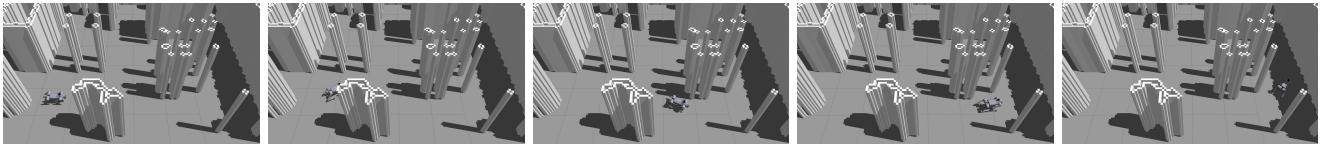


(b) LTLDG-S (ours)

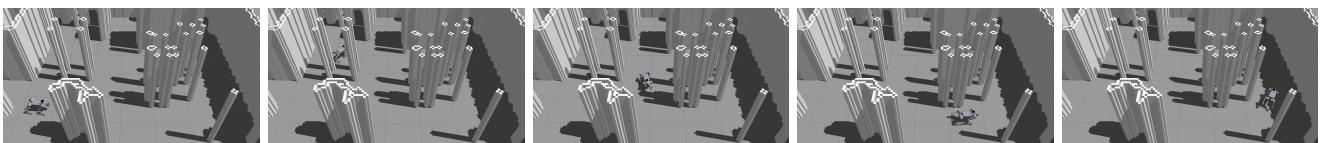


(c) LTLDG-R (ours)

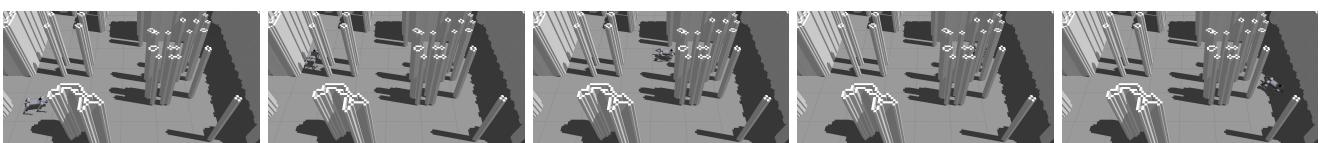
Fig. 6. Snapshots of comparisons between the baseline and LTLDG given an  $LTL_f$  with intricate *temporal* constraints. The  $LTL_f$  “ $\neg p_1 \cup (p_2 \wedge \neg p_0 \cup p_3)$ ” demands the mover never enter  $p_1$  until (or, unless by the time of entering) the two conditions are both met: 1) it has already been in  $p_2$ , and 2)  $p_0$  is never reached until  $p_3$  is visited. Clearly, the baseline breaches this complex constraint when proceeding with the original manipulation task, shown from the third to the fifth snapshot (the mover enters  $p_0$  *without* reaching  $p_3$  beforehand and then gets into  $p_0$ ). In contrast, our method, both LTLDG-S and LTLDG-R, manage to satisfy the constraint and complete the manipulation task by two different ways (the common ground is to first visit  $p_3$  and  $p_2$  during or on the way to maneuvers).



(a) Frames of trajectories from DIFFUSER in simulation.



(b) Frames of trajectories from LTLDG-S in simulation.



(c) Frames of trajectories from LTLDG-R in simulation.

Fig. 7. Results of safe planning and execution in simulation and indoor room on a quadruped robot.