Generative Trajectory Stitching through Diffusion Composition

Yunhao Luo[1], Utkarsh A. Mishra[1], Yilun Du[2,†], Danfei Xu[1,†]

***Abstract*— Effective trajectory stitching for long-horizon planning is a significant challenge in robotic decision-making. While diffusion models have shown promise in planning, they are limited to solving tasks similar to those seen in their training data. We propose CompDiffuser, a novel generative approach that can solve new tasks by learning to compositionally stitch together shorter trajectory chunks from previously seen tasks. Our key insight is modeling the trajectory distribution by subdividing it into overlapping chunks and learning their conditional relationships through a single bidirectional diffusion model. This allows information to propagate between segments during generation, ensuring physically consistent connections. We conduct experiments on benchmark tasks of various difficulties, covering different environment sizes, agent state dimension, trajectory types, training data quality, and show that CompDiffuser significantly outperforms existing methods. Project website at https://comp-diffuser.github.io/.**

I. Introduction

Generative models have demonstrated remarkable capabilities in modeling complex distributions across domains like images, videos, and 3D shapes. In robot planning, these models offer a promising approach by modeling distributions over plan sequences, which allows amortizing the computational cost of traditional search and optimization methods. This effectively transforms planning into sampling likely solutions given start and goal conditions. Recent works like Diffuser [1] and Decision Diffuser [2] have shown how diffusion models can learn to generate entire plans for long-horizon robotics tasks. However, exhaustively modeling joint distributions over entire plan sequences for all possible start and goal states remains extremely sample-inefficient, as it requires collecting long-horizon plan data covering all possible combinations of initial states and goals.

The concept of trajectory stitching [3] from Reinforcement Learning literature [4] presents a potential solution by combining chunks of different trajectories to create new, potentially better policies. The methods work by identifying high-reward trajectory chunks and stitching them together at states where they overlap or are similar enough, creating composite trajectories that can inform better policy learning. This effectively enables compositional generalization since collecting long consecutive trajectories is costly, and these short chunks can be flexibly assembled to complete new tasks. The key challenge lies in finding appropriate stitching points where trajectories can be combined while maintaining dynamic consistency and feasibility. Our goal is to enable generative planners to solve long-horizon tasks without requiring long-horizon training data, while retaining their ability to generate physically feasible, goal-directed plans.

[1]Georgia Tech [2]Harvard University [†]Equal advising

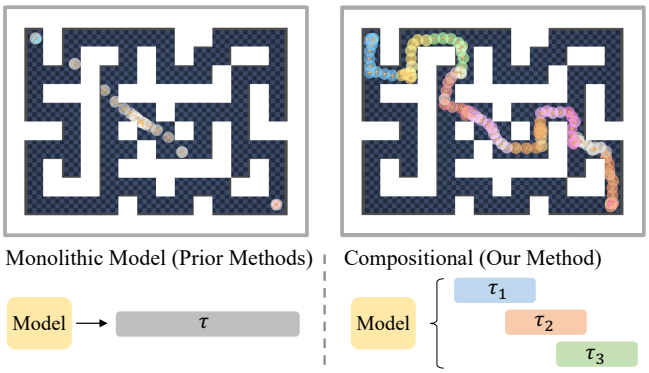


Fig. 1: **Compositional Trajectory Generation**. CompDiffuser enables generative trajectory stitching through diffusion composition. Left: Monolithic generative planner fails to generalize to tasks of longer horizon and collapses to the maze center. Right: Our method successfully navigates the ant agent from start to goal by compositionally stitching together shorter trajectories.

We propose a novel diffusion-based approach, Compositional Diffuser (CompDiffuser), that enables effective trajectory stitching through goal-conditioned causal trajectory generation. Our key insight is that we can model the trajectory distribution compositionally by subdividing it into distributions of overlapping chunks and learning their conditional relationships. Rather than learning separate models for each chunk, we train a single diffusion model that can generate trajectory chunks conditioned on neighboring chunks' states (Figure 1). This allows information to propagate bidirectionally during the reverse diffusion process as illustrated in Figure 2: each chunk's generation is influenced by both past and future chunks. This architecture naturally enables both parallel generation of chunks and causal autoregressive generation, each with different trade-offs in computational cost and planning quality.

We conduct extensive experiments across benchmark tasks of varying difficulty levels, including different environment sizes (from simple U-mazes to complex giant mazes), agent state dimensions (from 2D point agents to 50D humanoid robots), trajectory types (from maze navigation trajectories to ball dribbling trajectories), and training data quality (from clean demonstrations to noisy exploration data). Our results demonstrate that CompDiffuser significantly outperforms multiple imitation learning and offline reinforcement learning baselines across all settings. We show that our approach can effectively solve long-horizon tasks while maintaining plan feasibility and goal-reaching behavior. We validate the importance of our key technical components including the bidirectional conditioning mechanism, the autoregressive sampling process, and the flexible replanning capability.

In summary, the key contributions of this work are:

- A noisy-sample conditioned diffusion planning framework that enables learning compositional trajectory distributions by decomposing the trajectory generation procedure into a sequence of segments each generated by a separate diffusion denoising process.
- A compositional goal-conditioned trajectory planning method that uses bidirectional information propagation during denoising to maintain physical consistency between trajectory chunks.
- A set of empirical results showing significant improvements over existing methods across multiple trajectory stitching benchmarks, with detailed analysis of model capabilities and limitations.

II. Planning through Compositional Trajectory Generation

We aim to develop a generative planning framework that can generate long-horizon trajectories by composing multiple modular generative models. Our method, Compositional Diffuser (CompDiffuser), trains a single diffusion model on short-horizon trajectories. At inference time, given a start and goal, CompDiffuser runs parallel instances of this model to generate a sequence of overlapping trajectory segments, coordinating their denoising processes to ensure they smoothly connect into a coherent long-horizon plan. This approach enables us to stitch together short-horizon training trajectories to form novel long-horizon trajectory plans.

A. Compositional Trajectory Modeling

Given a planning problem, consisting of a start state q_s and a goal state q_g, we formulate planning as sampling a trajectory τ from the probability distribution

$$[s^{1:T}, a^{1:T}] \sim p_\theta(\tau | q_s, q_g), \tag{1}$$

where $s^{1:T}$ corresponds to future states to reach the goal state q_g and $a^{1:T}$ corresponds to a set of future actions. To implement this sampling procedure, prior work [2], [1] learns a generative model $p(\tau)$ directly over previous trajectories τ in the environment. However, since the generative model is trained to model the density of previously seen trajectories, it is restricted to generating plans with start and goal that are similar to those seen in the past.

In this paper, we propose to model the generative model over trajectories $p_\theta(\tau|q_s, q_g)$ compositionally [5], where we subdivide trajectory τ into a set of K overlapping sub-chunks τ_k (Figure 1). We then represent the trajectory distribution as

$$\begin{aligned} p_\theta(\tau|q_s, q_g) \propto & \, p_1(\tau_1|q_s, \tau_2) \; p_K(\tau_K|\tau_{K-1}, q_g) \\ & \prod_{k=2}^{K-1} p_k(\tau_k|\tau_{k-1}, \tau_{k+1}). \end{aligned} \tag{2}$$

In the above expression, each trajectory chunk τ_k is only dependent on nearby trajectory chunks τ_{k-1} and τ_{k+1}. This allows $p_\theta(\tau|q_s, q_g)$ to generate trajectory plans that significantly depart from previously seen trajectories, as long as intermediate trajectory chunks τ_k have been seen. Overall, the goal of our method is to enable long-horizon planning without long-horizon training data.

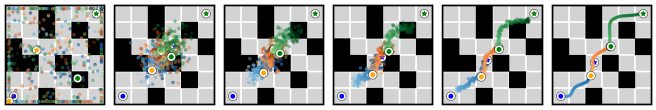


Fig. 2: **Illustrating the Trajectory Stitching Process.** Given an unseen start (blue circle) and goal (green star), CompDiffuser generates a long-horizon plan by progressively denoising three trajectory chunks in parallel, with each chunk conditioning on its neighbors to ensure smooth transitions.

B. Training Compositional Trajectory Models

One approach to represent the composed probability distribution in Equation 2 is to directly learn separate generative models to represent each conditional probability distribution. However, sampling from the composed distribution is challenging, as each individual trajectory chunk depends on the values of neighboring chunks. As a result, to sample from the composed distribution, one would need a blocked Gibbs sampling procedure, where the value of each individual trajectory chunk is iteratively sampled given decoded values of neighboring trajectory chunks. This sampling procedure is slow, and passing information across chunks to form a consistent plan is challenging.

Information Propagation Through Noisy-Sample Conditioning. We propose a more efficient approach that addresses these challenges by leveraging the progressive denoising process of diffusion models. The key challenge in composing trajectories is ensuring feasible transitions between each pair of neighboring chunks, i.e., maintaining physical constraints and dynamic consistency at connection points where segments overlap. Our key insight is that we can achieve this by having trajectory segments guide each other's generation: as one segment takes shape through denoising, it helps shape its neighbors into compatible configurations.

We implement this insight using a diffusion model that generates trajectory chunks conditioning on their neighbors' *noisy samples*. Given a dataset D of trajectories τ, we train a denoising network ϵ_θ to learn the trajectory distribution $p_\theta(\tau_k|\tau_{k-1}, \tau_{k+1})$ with the training objective

$$\mathcal{L}_{\text{nbr}} = \mathbb{E}_{\tau \in D, t, k} \left[\| \epsilon - \epsilon_\theta(\tau_k^t, t \mid \tau_{k-1}^t, \tau_{k+1}^t) \|^2 \right], \tag{3}$$

where k identifies a trajectory segment, t is the noise level, and τ_k^t represents segment k corrupted with noise level t. Crucially, when denoising each segment, the network conditions on noisy versions of neighboring segments $\tau_{k-1}^t, \tau_{k+1}^t$ at the same noise level. This allows each segment to influence its neighbors' denoising process, ensuring their final configurations are dynamically compatible. In addition, further training the network to condition on τ_{k-1}^{t-1} can enable autoregressive compositional sampling, which we will discuss in Section II-C. In practice, we only need to condition on the small overlapping regions between consecutive trajectories, making the generation process efficient while maintaining consistency across connection points.

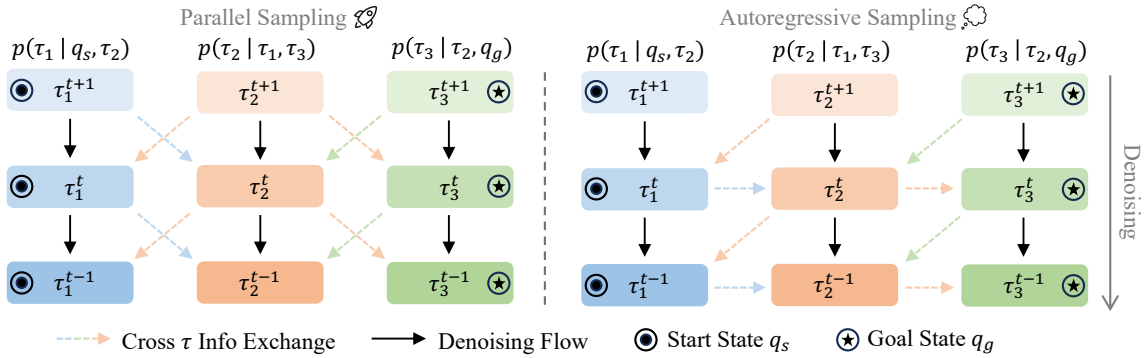


Fig. 3: **Compositional Trajectory Planning: Parallel Sampling and Autoregressive Sampling**. We present an illustrative example of sampling three trajectories $\tau_{1:3}$ with the proposed compositional sampling methods. Dashed lines represent cross trajectory information exchange between adjacent trajectories and black lines represent the denoising flow of each trajectory. In parallel sampling, $\tau_{1:3}$ can be denoised concurrently; while in autoregressive sampling, denoising τ_k depends on the previous trajectory τ_{k-1}, e.g., the denoising of τ_2 depends on τ_1 (as shown in the blue horizontal dashed arrows). Additionally, start state q_s and goal state q_g conditioning are applied to the trajectories in the two ends, τ_1 and τ_3, which enables goal-conditioned planning. Trajectories $\tau_{1:3}$ will be merged to form a longer plan τ_{comp} after the full diffusion denoising process.

Representing Initial States and Goals. In addition, we further train the same denoising network to represent the distributions $p_\theta(\tau_1|q_s, \tau_2)$ and $p_\theta(\tau_K|\tau_{K-1}, q_g)$. This corresponds to the training objective

$$\mathcal{L}_{\text{start}} = \mathbb{E}_{\tau\in D,t,k}\left[\|\epsilon - \epsilon_\theta(\tau_1^t, t \mid q_s, \tau_2^t)\|^2\right], \qquad (4)$$

with an analogous objective for the conditioned goal state q_g. We train the same denoising network ϵ_θ with both conditioning. Please see Appendix XII for implementation details. We provide an overview of our proposed training strategy in Algorithm 1.

C. Compositional Trajectory Planning

Our compositional framework enables flexible sampling strategies for generating long-horizon plans with Equation 2. The basic sampling process starts by initializing each trajectory chunk τ_k with Gaussian noise. Then, through iterative denoising, each chunk is denoised while being conditioned on its neighbors. This structure allows for different ways of coordinating the denoising process across trajectory chunks, each offering different tradeoffs between information propagation and computational efficiency. We present two sampling schemes (illustrated in Figure 3):

Parallel Sampling. Our first sampling approach conditions denoising on the noisy adjacent trajectory chunks from the previous denoising timestep, where the update rule is

$$\tau_k^{t-1} = \alpha^t(\tau_k^t - \epsilon_\theta(\tau_k^t|\tau_{k-1}^t, \tau_{k+1}^t) + \beta^t\xi), \quad \xi \sim \mathcal{N}(0,1),$$

where α^t and β^t are diffusion specific hyperparameters. This approach allows us to run denoising on each trajectory chunk in parallel, as each denoising update only requires the values of the adjacent trajectory chunks at a previous noise level. However, information propagation between the values of adjacent trajectory chunks is limited at each denoising timestep, as each trajectory chunk is denoised independently of the denoising updates of other trajectory chunks.

Autoregressive Sampling. To better couple the values of adjacent trajectory chunks, we propose to denoise each trajectory chunk autoregressively dependent on the values of neighboring chunks at each denoising timestep. In particular, we iteratively denoise each trajectory $\tau_{1:K}^t$ starting from the τ_1^t, and condition the denoising of τ_k^t on the previously decoded chunk τ_{k-1}^{t-1} at the current noise level $t-1$ and the future chunk τ_{k+1}^t at the previous noise level t, giving us the equation

$$\tau_k^{t-1} = \alpha^t(\tau_k^t - \epsilon_\theta(\tau_k^t|\tau_{k-1}^{t-1}, \tau_{k+1}^t) + \beta^t\xi), \quad \xi \sim \mathcal{N}(0,1).$$

This sequential generation process enables stronger coordination among the chunks since each chunk is conditioned on the less noisy version of its previous chunk. However, it requires generating chunks one at a time rather than simultaneously, making it computationally less efficient than parallel sampling. We compare the two sampling schemes in Table VII, where we empirically find autoregressive sampling leads to improved performance. Additionally, we provide sampling time comparison in Table X. We use this autoregressive sampling procedure throughout the experiments in the paper and illustrate pseudocode for sampling in Algorithm 2. Given such final set of generated chunks $\tau_{1:K}$, we then merge the chunks together to construct a final trajectory τ_{comp} by applying exponential trajectory blending to areas where subchunks τ_k overlap (See Appendix XII-B for details).

III. Experiments

Our objective is to (1) validate that our method enforces coherent trajectory stitching on multiple benchmarks, with varying state space dimensions, task design, and training data collection policies (2) understand how planning with higher state dimensions, varying numbers of composed trajectories, different sampling schemes, and replanning affect the performance of the proposed method. Please see Appendix VII for dataset details and Appendix VIII for descriptions of baselines. Additional results are provided in Appendix IX and X with failure analysis in Appendix XI.

Evaluation Setup. For each environment, we report the success rate over all evaluation episodes, where the success criterion is that the agent or target object is close to the goal

Env	Type	Size	GCBC	GCIVL	GCIQL	QRL	CRL	HIQL	GSC	Ours
antmaze	stitch	Medium	45 ±11	44 ±6	29 ±6	59 ±7	53 ±6	**94** ±1	**97**±2	**96**±2
		Large	3 ±3	18 ±2	7 ±2	18 ±2	11 ±2	67 ±5	66±2	**86**±2
		Giant	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	21 ±2	20±1	**65**±3
	explore	Medium	2 ±1	19 ±3	13 ±2	1 ±1	3 ±2	37 ±10	**90**±2	81±2
		Large	0 ±0	10 ±3	0 ±0	0 ±0	0 ±0	4 ±5	21±3	**27**±1
humanoid maze	stitch	Medium	29 ±5	12 ±2	12 ±3	18 ±2	71 ±3	**96** ±4	**92**±1	**91**±1
		Large	6 ±3	1 ±1	0 ±0	3 ±1	6 ±1	31 ±3	**70**±3	**72**±3
		Giant	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	12 ±2	5±1	**67**±4

TABLE I: **Quantitative Results on AntMaze and HumanoidMaze in OGBench.** We benchmark our method on the 5 test-time tasks defined in OGBench with 20 episodes per task. We average the results over 5 seeds and report mean and standard deviation.

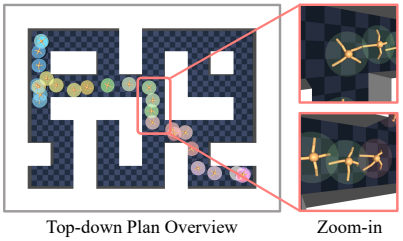


Fig. 4: **Qualitative Results of Planning in High Dimension on OGBench AntMaze Large.** Original plan is sub-sampled for clearer view.

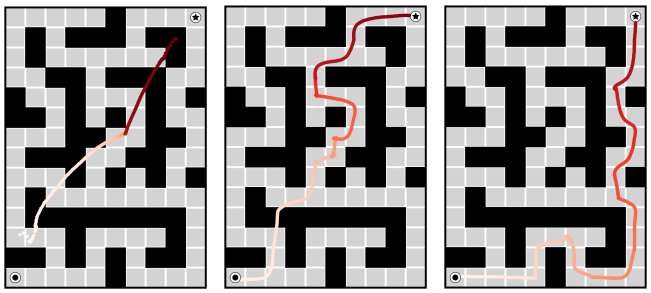


Fig. 5: **Qualitative Comparison of DD, GSC and CompDiffuser on OGBench PointMaze Giant.** Effective bidirection information propagation enables CompDiffuser to successfully synthesize trajectories from start (bottom left) to goal (upper right), while other methods generate *o.o.d* trajectories that are disconnected with the start/goal or passing through walls. See Figure 9 for per-segment visualization.

Size	HIQL	Ours (2D)	Ours (15D)	Ours (29D)
Medium	94±1	96±2	95±0	**97**±2
Large	67±5	**86**±2	66±5	66±5
Giant	21±2	**65**±3	41±3	28±4

TABLE II: **Quantitative Results of Different Planning Dimensions on OGBench AntMaze Stitch.** Our method constructs feasible plans that reach long-distance goals while modeling complex dynamics, such as agent's joint positions and velocities.

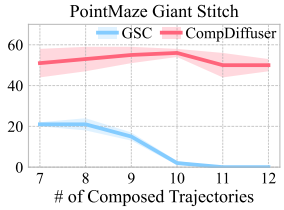


Fig. 6: Success Rate versus Different Numbers of Composed Trajectories K in OGBench `PointMaze Giant` (w/o replan).

within a small threshold. We evaluate all methods with 5 random seeds for each experiment and report the mean and standard deviation. Specifically, in [6] datasets, we evaluate on 2 tasks in `U-Maze`, 6 tasks in `Medium`, and 7 tasks in `Large` with 10 episodes per task; in OGBench [7], we evaluate on 5 tasks in each environment with 20 episodes per task. Each task is introduced in the respective papers and is defined by a base start and goal state that require trajectory stitching to complete. A random noise is added to the base start and goal state for each evaluation episode.

A. PointMaze

We present experiment results on two types of trajectory-stitching datasets in point maze environments, featuring different dataset collection strategies. Our method is trained on short trajectories of *x-y* positions of the point agent while can directly generate much longer trajectories from start and goal by composing multiple trajectories (numbers of composed trajectories in each experiment are provided in Table XIII). See Figure 5 and Figure 9 for qualitative results. Task details and quantitative results are shown in Appendix IX-A.

B. High Dimension Tasks

AntMaze and HumanoidMaze. We present the evaluation results of our method on various trajectory stitching tasks involving higher-dimensional state spaces within OGBench: `AntMaze`, `HumanoidMaze`. The data collection strategy is identical to OGBench `PointMaze`, where each episode is constrained to travel at most 4 blocks, while at inference, a successful plan requires the agent to travel up to 30 blocks. Quantitative results are shown in Table I. See Appendix IX-B for environment and implementation details. Additionally, we present results on `AntSoccer` in Appendix IX-C.

C. Ablation Studies

Planning in High Dimension Space. We report experiment results where CompDiffuser synthesizes trajectories in state space of higher dimension. We compare the success rates of planning in dimension of 2D, 15D, and 29D in Table II, and present qualitative plans in Figure 4 and Figure 12.

Different Numbers of Composed Trajectories. We study the effect of varying the numbers of trajectories to be composed K. To better study the planning performance with respect to K, we use the challenging `PointMaze-Giant-Stitch` in OGBench as the testbed. As shown in Figure 6, our method obtains consistent performance when composing 7 to 12 trajectories. Qualitatively, decreasing K will result in a sparser trajectory while increasing K will cause the final trajectory traveling back and forth to consume the redundant states (See Figure 9).

IV. Conclusion

We introduce CompDiffuser, a generative trajectory stitching method leveraging the compositionality of diffusion models. We propose a noise-conditioned score function formulation that helps perform autoregressive sampling of multiple short-horizon trajectories and can eventually stitch them to form a longer-horizon goal-conditioned trajectory. Our method demonstrates effective trajectory stitching capabilities as evident from the extensive experiments on tasks of various difficulties, including different environment sizes, planning state dimensions, trajectory types, and training data quality.

REFERENCES

[1] M. Janner, Y. Du, J. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9902–9915.

[2] A. Ajay, Y. Du, A. Gupta, J. B. Tenenbaum, T. S. Jaakkola, and P. Agrawal, "Is conditional generative modeling all you need for decision making?" in *The Eleventh International Conference on Learning Representations*, 2022.

[3] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Y. Du and L. Kaelbling, "Compositional generative modeling: A single model is not all you need," *arXiv preprint arXiv:2402.01103*, 2024.

[6] R. Ghugare, M. Geist, G. Berseth, and B. Eysenbach, "Closing the gap between td learning and supervised learning-a generalisation point of view." in *The Twelfth International Conference on Learning Representations*, 2024.

[7] S. Park, K. Frans, B. Eysenbach, and S. Levine, "Ogbench: Benchmarking offline goal-conditioned rl," in *International Conference on Learning Representations (ICLR)*, 2025.

[8] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. PMLR, 2015, pp. 2256–2265.

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

[10] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Z. Tan, I. Momennejad, K. Hofmann *et al.*, "Imitating human behaviour with diffusion models," *The Eleventh International Conference on Learning Representations*, 2023.

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

[12] H. He, C. Bai, K. Xu, Z. Yang, W. Zhang, D. Wang, B. Zhao, and X. Li, "Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning," *Advances in neural information processing systems*, vol. 36, pp. 64 896–64 917, 2023.

[13] T. Ubukata, J. Li, and K. Tei, "Diffusion model for planning: A systematic literature review," *arXiv preprint arXiv:2408.10266*, 2024.

[14] H. Lu, D. Han, Y. Shen, and D. Li, "What makes a good diffusion planner for decision making?" in *The Thirteenth International Conference on Learning Representations*, 2025.

[15] Z. Zhu, M. Liu, L. Mao, B. Kang, M. Xu, Y. Yu, S. Ermon, and W. Zhang, "Madiff: Offline multi-agent learning with diffusion models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 4177–4206, 2024.

[16] B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann, "Diffusion forcing: Next-token prediction meets full-sequence diffusion," *Advances in Neural Information Processing Systems*, vol. 37, pp. 24 081–24 125, 2024.

[17] 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.

[18] Y. Luo, C. Sun, J. B. Tenenbaum, and Y. Du, "Potential based diffusion motion planning," in *International Conference on Machine Learning*. PMLR, 2024, pp. 33 486–33 510.

[19] H. Wang, Y. Wu, S. Guo, and L. Wang, "Pdpp: Projected diffusion for procedure planning in instructional videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 836–14 845.

[20] C.-F. Yang, H. Xu, T.-L. Wu, X. Gao, K.-W. Chang, and F. Gao, "Planning as in-painting: A diffusion-based embodied task planning framework for environments under uncertainty," *arXiv preprint arXiv:2312.01097*, 2023.

[21] X. Fang, C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Dimsam: Diffusion models as samplers for task and motion planning under partial observability," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 1412–1419.

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

[23] B. Liao, S. Chen, H. Yin, B. Jiang, C. Wang, S. Yan, X. Zhang, X. Li, Y. Zhang, Q. Zhang *et al.*, "Diffusiondrive: Truncated diffusion model for end-to-end autonomous driving," *arXiv preprint arXiv:2411.15139*, 2024.

[24] J. Wang, X. Zhang, Z. Xing, S. Gu, X. Guo, Y. Hu, Z. Song, Q. Zhang, X. Long, and W. Yin, "He-drive: Human-like end-to-end driving with vision language models," *arXiv preprint arXiv:2410.05051*, 2024.

[25] J. Ye, J. Gao, S. Gong, L. Zheng, X. Jiang, Z. Li, and L. Kong, "Beyond autoregression: Discrete diffusion for complex reasoning and planning," *arXiv preprint arXiv:2410.14157*, 2024.

[26] F. Nuti, T. Franzmeyer, and J. F. Henriques, "Extracting reward functions from diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[27] W. Li, X. Wang, B. Jin, and H. Zha, "Hierarchical diffusion for offline decision making," in *International Conference on Machine Learning*. PMLR, 2023, pp. 20 035–20 064.

[28] C. Chen, F. Deng, K. Kawaguchi, C. Gulcehre, and S. Ahn, "Simple hierarchical planning with diffusion," *arXiv preprint arXiv:2401.02644*, 2024.

[29] X. Ma, S. Patidar, I. Haughton, and S. James, "Hierarchical diffusion policy for kinematics-aware multi-task robotic manipulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 18 081–18 090.

[30] Z. Liang, Y. Mu, M. Ding, F. Ni, M. Tomizuka, and P. Luo, "Adaptdiffuser: Diffusion models as adaptive self-evolving planners," in *International Conference on Machine Learning*. PMLR, 2023, pp. 20 725–20 745.

[31] Z. Dong, Y. Yuan, J. HAO, F. Ni, Y. Mu, Y. ZHENG, Y. Hu, T. Lv, C. Fan, and Z. Hu, "Aligndiff: Aligning diverse human preferences via behavior-customisable diffusion model," in *The Twelfth International Conference on Learning Representations*, 2024.

[32] L. Feng, P. Gu, B. An, and G. Pan, "Resisting stochastic risks in diffusion planners with the trajectory aggregation tree," in *Forty-first International Conference on Machine Learning*, 2024.

[33] K. Lee, S. Kim, and J. Choi, "Refining diffusion planner for reliable behavior synthesis by automatic detection of infeasible plans," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[34] S. Zhou, Y. Du, S. Zhang, M. Xu, Y. Shen, W. Xiao, D.-Y. Yeung, and C. Gan, "Adaptive online replanning with diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[35] J. Sun, Y. Jiang, J. Qiu, P. Nobel, M. J. Kochenderfer, and M. Schwager, "Conformal prediction for uncertainty-aware planning with diffusion dynamics model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[36] J. Brehmer, J. Bose, P. De Haan, and T. S. Cohen, "Edgi: Equivariant diffusion for planning with embodied agents," *Advances in Neural Information Processing Systems*, vol. 36, pp. 63 818–63 834, 2023.

[37] G. Li, Y. Shan, Z. Zhu, T. Long, and W. Zhang, "Diffstitch: Boosting offline reinforcement learning with diffusion-based trajectory stitching," in *Forty-first International Conference on Machine Learning*, 2024.

[38] C. Lu, P. Ball, Y. W. Teh, and J. Parker-Holder, "Synthetic experience replay," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[39] S. Kim, Y. Choi, D. E. Matsunaga, and K.-E. Kim, "Stitching sub-trajectories with conditional diffusion model for goal-conditioned offline rl," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 12, 2024, pp. 13 160–13 167.

[40] J. Lee, S. Yun, T. Yun, and J. Park, "GTA: Generative trajectory augmentation with guidance for offline reinforcement learning," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[41] S. Li and X. Zhang, "Augmenting offline reinforcement learning with state-only interactions," *arXiv preprint arXiv:2402.00807*, 2024.

[42] Z. Liu, L. Qian, Z. Liu, L. Wan, X. Chen, and X. Lan, "Enhancing decision transformer with diffusion-based trajectory branch generation," *arXiv preprint arXiv:2411.11327*, 2024.

[43] I. Char, V. Mehta, A. Villaflor, J. M. Dolan, and J. Schneider, "Bats: Best action trajectory stitching," *arXiv preprint arXiv:2204.12026*, 2022.

[44] X. Lei, X. Zhang, and D. Wang, "Mgda: Model-based goal data

augmentation for offline goal-conditioned weighted supervised learning," *arXiv preprint arXiv:2412.11410*, 2024.

[45] Z. Zhou, C. Zhu, R. Zhou, Q. Cui, A. Gupta, and S. S. Du, "Free from bellman completeness: Trajectory stitching via model-based return-conditioned supervised learning," in *The Twelfth International Conference on Learning Representations*, 2024.

[46] C. A. Hepburn and G. Montana, "Model-based trajectory stitching for improved offline reinforcement learning," in *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022.

[47] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[48] X. Huang, D. Wu, and B. Boulet, "Drdt3: Diffusion-refined decision test-time training model," *arXiv preprint arXiv:2501.06718*, 2025.

[49] Y.-H. Wu, X. Wang, and M. Hamaya, "Elastic decision transformer," *Advances in neural information processing systems*, vol. 36, pp. 18 532–18 550, 2023.

[50] Z. Zhuang, D. Peng, J. Liu, Z. Zhang, and D. Wang, "Reinformer: Max-return sequence modeling for offline rl," in *International Conference on Machine Learning*. PMLR, 2024, pp. 62 707–62 722.

[51] Y. Wang, C. Yang, Y. Wen, Y. Liu, and Y. Qiao, "Critic-guided decision transformer for offline reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 14, 2024, pp. 15 706–15 714.

[52] Z. Zeng, C. Zhang, S. Wang, and C. Sun, "Goal-conditioned predictive coding for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 25 528–25 548, 2023.

[53] S. Venkatraman, S. Khaitan, R. T. Akella, J. Dolan, J. Schneider, and G. Berseth, "Reasoning with latent diffusion in offline reinforcement learning," *arXiv preprint arXiv:2309.06599*, 2023.

[54] Z. Zhang, J. Xu, J. Liu, Z. Zhuang, D. Wang, M. Liu, and S. Zhang, "Context-former: Stitching via latent conditioned sequence modeling," *arXiv preprint arXiv:2401.16452*, 2024.

[55] C.-X. Gao, C. Wu, M. Cao, R. Kong, Z. Zhang, and Y. Yu, "Act: Empowering decision transformer with dynamic programming via advantage conditioning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 11, 2024, pp. 12 127–12 135.

[56] J. Kim, S. Lee, W. Kim, and Y. Sung, "Adaptive q-aid for conditional supervised learning in offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 87 104–87 135, 2024.

[57] T. Yamagata, A. Khalil, and R. Santos-Rodriguez, "Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 989–39 007.

[58] Y. Du, S. Li, and I. Mordatch, "Compositional visual generation with energy based models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6637–6647, 2020.

[59] Y. Du, C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. S. Grathwohl, "Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc," in *International conference on machine learning*. PMLR, 2023, pp. 8489–8510.

[60] T. Garipov, S. De Peuter, G. Yang, V. Garg, S. Kaski, and T. Jaakkola, "Compositional sculpting of iterative generative processes," *arXiv preprint arXiv:2309.16115*, 2023.

[61] D. Mahajan, M. Pezeshki, I. Mitliagkas, K. Ahuja, and P. Vincent, "Compositional risk minimization," *arXiv preprint arXiv:2410.06303*, 2024.

[62] A. Bradley, P. Nakkiran, D. Berthelot, J. Thornton, and J. M. Susskind, "Mechanisms of projective composition of diffusion models," *arXiv preprint arXiv:2502.04549*, 2025.

[63] M. Okawa, E. S. Lubana, R. Dick, and H. Tanaka, "Compositional abilities emerge multiplicatively: Exploring diffusion models on a synthetic task," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[64] J. Thornton, L. Bethune, R. Zhang, A. Bradley, P. Nakkiran, and S. Zhai, "Composition and control with distilled energy diffusion models and sequential monte carlo," *arXiv preprint arXiv:2502.12786*, 2025.

[65] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum, "Compositional visual generation with composable diffusion models," in *European Conference on Computer Vision*. Springer, 2022, pp. 423–439.

[66] M. Yang, Y. Du, B. Dai, D. Schuurmans, J. B. Tenenbaum, and P. Abbeel, "Probabilistic adaptation of text-to-video models," *arXiv preprint arXiv:2306.01872*, 2023.

[67] Q. Zhang, J. Song, X. Huang, Y. Chen, and M.-Y. Liu, "Diffcollage: Parallel generation of large content with diffusion models," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023, pp. 10 188–10 198.

[68] J. Su, N. Liu, Y. Wang, J. B. Tenenbaum, and Y. Du, "Compositional image decomposition with diffusion models," *arXiv preprint arXiv:2406.19298*, 2024.

[69] M. Skreta, L. Atanackovic, A. J. Bose, A. Tong, and K. Neklyudov, "The superposition of diffusion models using the itô density estimator," *arXiv preprint arXiv:2412.17762*, 2024.

[70] Y. Shafir, G. Tevet, R. Kapon, and A. H. Bermano, "Human motion diffusion as a generative prior," *arXiv preprint arXiv:2303.01418*, 2023.

[71] S. Sun, G. De Araujo, J. Xu, S. Zhou, H. Zhang, Z. Huang, C. You, and X. Xie, "Coma: Compositional human motion generation with multi-modal agents," *arXiv preprint arXiv:2412.07320*, 2024.

[72] J. Zhang, H. Fan, and Y. Yang, "Energymogen: Compositional human motion generation with energy-based diffusion model in latent space," *arXiv preprint arXiv:2412.14706*, 2024.

[73] H. Lin, X. Huang, T. Phan-Minh, D. S. Hayden, H. Zhang, D. Zhao, S. Srinivasa, E. M. Wolff, and H. Chen, "Causal composition diffusion model for closed-loop traffic generation," *arXiv preprint arXiv:2412.17920*, 2024.

[74] Z. Yang, J. Mao, Y. Du, J. Wu, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Compositional diffusion-based continuous constraint solvers," *arXiv preprint arXiv:2309.00966*, 2023.

[75] A. Ajay, S. Han, Y. Du, S. Li, A. Gupta, T. S. Jaakkola, J. B. Tenenbaum, L. P. Kaelbling, A. Srivastava, and P. Agrawal, "Compositional foundation models for hierarchical planning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[76] L. Wang, J. Zhao, Y. Du, E. H. Adelson, and R. Tedrake, "Poco: Policy composition from and for heterogeneous robot learning," *arXiv preprint arXiv:2402.02511*, 2024.

[77] O. Patil, A. Sah, and N. Gopalan, "Composing diffusion policies for few-shot learning of movement trajectories," *arXiv preprint arXiv:2410.17479*, 2024.

[78] U. A. Mishra, S. Xue, Y. Chen, and D. Xu, "Generative skill chaining: Long-horizon skill planning with diffusion models," in *7th Annual Conference on Robot Learning*, 2023.

[79] U. A. Mishra, Y. Chen, and D. Xu, "Generative factor chaining: Coordinated manipulation with diffusion-based factor graph," in *8th Annual Conference on Robot Learning*, 2024.

[80] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine, "Rvs: What is essential for offline rl via supervised learning?" *arXiv preprint arXiv:2112.10751*, 2021.

[81] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," in *Conference on robot learning*. PMLR, 2020, pp. 1113–1132.

[82] D. Ghosh, A. Gupta, A. Reddy, J. Fu, C. Devin, B. Eysenbach, and S. Levine, "Learning to reach goals via iterated supervised learning," *arXiv preprint arXiv:1912.06088*, 2019.

[83] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[84] T. Wang, A. Torralba, P. Isola, and A. Zhang, "Optimal goal-reaching reinforcement learning via quasimetric learning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 36 411–36 430.

[85] B. Eysenbach, T. Zhang, S. Levine, and R. R. Salakhutdinov, "Contrastive learning as goal-conditioned reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 35 603–35 620, 2022.

[86] S. Park, D. Ghosh, B. Eysenbach, and S. Levine, "Hiql: Offline goal-conditioned rl with latent states as actions," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[87] 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.

[88] 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.

APPENDIX

In this appendix, we first introduce the evaluation environments and stitching datasets in Section VII. Next, we provide additional quantitative results in Appendix IX and additional qualitative results in Appendix X. We provide failure mode analysis in Section XI. Following this, we provide implementation details in Section XII, including model architecture, training and evaluation setups, trajectory merging, and replanning. Lastly, we introduce notable baselines in Section VIII.

V. Related Work

Diffusion Models For Planning. Many works have studied the applications of diffusion models [8], [9] for generative planning [1], [2], [10], [11], [12], [13], [14], [15], [16]. Diffusion planning has been widely applied in various fields, such as motion planning [17], [18], procedure planning [19], task planning [20], [21], autonomous driving [22], [23], [24], reasoning [25], and reward learning [26]. Many techniques have also been combined with diffusion planning, including hierarchical planning [27], [28], [29], self-evolving planner [30], preference alignment [31], tree branch-pruning [32], refining [33], replanning [34], uncertainty-aware planning [35], equivariance [36]. However, these works are usually constrained to plan within similar horizons as training data. Our work instead proposes a compositional diffusion planning approach that generalizes to much longer horizon via generative trajectory stitching.

Trajectory Stitching. A flurry of works have explored the trajectory stitching problem given offline data. One typical category of solution is based on data augmentation or goal relabeling along with various techniques, such as generative models [37], [38], [39], [40], [41], [42], model-based approaches [43], [44], [45], [46], and clustering [6]. Other supervised learning based methods, such as sequence modeling [47], [48], [49], [50], [51], latent space learning [52], [53], [54], and learning with dynamic programming [55], [56], [57], have also demonstrated some extent of stitching capability. In this work, we propose a different trajectory stitching approach based on generative modeling, where the model only learns from plain short trajectory segments while is able to directly perform goal-conditioned trajectory stitching through test-time compositional generation.

Compositional Generative Models. Compositional Generative Models [58], [59], [60], [5], [61], [62], [63], [64] are widely studied in various domains, including visual content generation [65], [59], [66], [67], [68], [69], human motion generation [70], [71], [72], traffic generation [73], robotic planning [74], [75], [18], and policy learning [76], [77]. Most existing work on compositionality focuses on sampling under the conjunction of several given conditions. While several studies [78], [79] have explored sequential compositional models, they are restricted to planning within a given skeleton and hence unable to generalize to longer sequences or new tasks. In contrast, we propose a compositional planning framework that scales to generating much longer sequences and completing new tasks via directly stitching short trajectory segments, without relying on pre-defined task-dependent skeletons.

VI. Pseudocode for Training and Inference

Algorithm 1 Training CompDiffuser

Require: train dataset D, diffusion denoiser model $\epsilon_\theta(\tau^t, t \mid \text{st\_cond}, \text{end\_cond})$, number of training steps N, diffusion timestep T
for $i = 1 \rightarrow N$ **do**
  $\tau^0 \sim D$, sample clean trajectory from dataset
  $\tau^0_{1:K} \leftarrow$ divide τ^0 to K overlapping chunks
  $\tau^t_{1:K} \leftarrow$ add noise to $\tau^0_{1:K}$, $t \sim T$, following DDPM
  # Objective for noisy chunk condition
  $\mathcal{L}_{\text{nbr}} = \|\epsilon - \epsilon_\theta(\tau^t_k, t \mid \tau^t_{k-1}, \tau^t_{k+1})\|^2,\ k \sim [2, K-1]$
  # Objective for start / end state condition
  $\mathcal{L}_{\text{start}} = \|\epsilon - \epsilon_\theta(\tau^t_1, t \mid q_s, \tau^t_2)\|^2,\ q_s = \tau^0_1[0]$
  $\mathcal{L}_{\text{end}} = \|\epsilon - \epsilon_\theta(\tau^t_K, t \mid \tau^t_{K-1}, q_g)\|^2,\ q_g = \tau^0_K[-1]$
  $\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{nbr}} + \mathcal{L}_{\text{start}} + \mathcal{L}_{\text{end}}$
  Backprop to update $\epsilon_\theta(.)$ using $\mathcal{L}_{\text{all}}$
end for
return $\epsilon_\theta(.)$

Algorithm 2 Autoregressive Trajectory Sampling

Models: trained diffusion denoiser model $\epsilon_\theta(\tau, t \mid \text{st\_cond}, \text{g\_cond})$
Input: start state q_s, goal state q_g, number of composed trajectories K
Initialize K trajectories $\tau_{1:K} \sim \mathcal{N}(0, I)$
for $t = T \rightarrow 1$ **do**
  # Denoise τ_1 conditioned on q_s and τ^t_2
  $\tau^{t-1}_1 = \epsilon_\theta(\tau^t_1, t \mid q_\text{s}, \tau^t_2)$
  # Denoise intermediate trajectories τ_2 to τ_{K-1}
  for $k = 2 \rightarrow K-1$ **do**
    $\tau^{t-1}_k = \epsilon_\theta(\tau^t_k, t \mid \tau^{t-1}_{k-1}, \tau^t_{k+1})$
  end for
  # Denoise τ_K conditioned on τ^{t-1}_{K-1} and q_g
  $\tau^{t-1}_K = \epsilon_\theta(\tau^t_K, t \mid \tau^{t-1}_{K-1}, q_\text{g})$
end for
τ_comp = Merge the denoised trajectories $\tau^0_{1:K}$
return τ_comp

VII. Environment and Stitching Datasets

In this paper, we directly evaluate our method on public stitching datasets introduced in two recent papers [6] and OGBench [7]. In this section, we provide detailed descriptions of each dataset along with qualitative examples of trajectories in these datasets.

A. Stitching Datasets in [6]

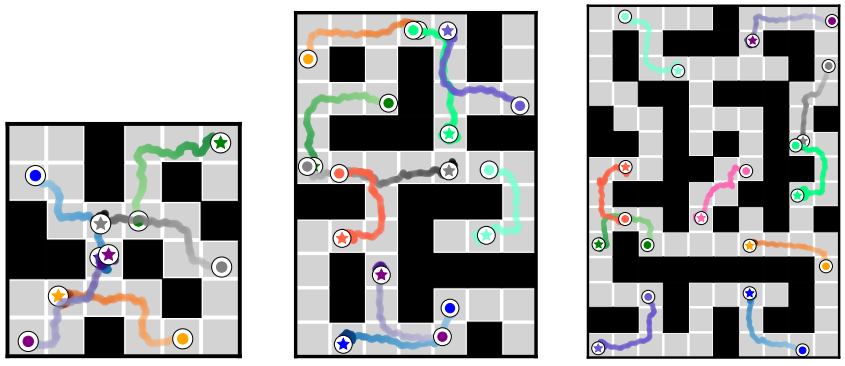
This paper [6] divides each evaluation environment into several small regions and each demonstration trajectory in the training datasets can only navigate within a specific region. There is a small overlap (one block) between each region, which can be used to stitch trajectories across regions. Therefore, to complete test-time goals, the agent needs to conduct effective reasoning based on the given start state and goal state and identify the corresponding overlap joints. The division of regions is visualized in the original paper [6]. We use the environments and datasets from their official implementation release at https://github.com/RajGhugare19/stitching-is-combinatorial-generalisation.

B. OGBench Datasets

OGBench is a comprehensive benchmark designed for offline goal-conditioned RL. Since our focus is to evaluate the trajectory stitching ability of CompDiffuser, we use the `Stitch` and `Explore` dataset types in OGBench.

In `Stitch` datasets, trajectories are constrained to navigate no more than 4 blocks in the environment. The start and goal state of each trajectory can be sampled from the entire environment provided that the travel distance between the start and goal is within 4 blocks. Qualitative examples of the trajectories in the `Stitch` dataset are shown in Figure 7.

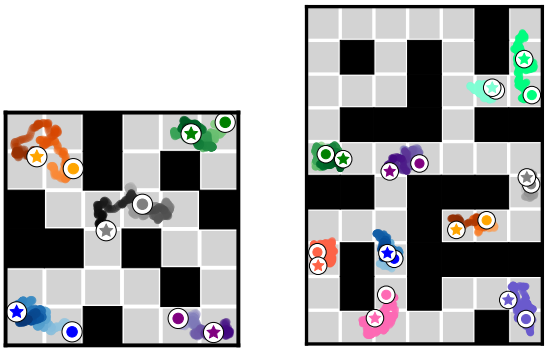
In `Explore` datasets, trajectories are of extremely low-quality though high-coverage. The data collection policy contains a large amount of action noise and will randomly re-sample a new moving direction after every 10 steps. Hence, each demonstration trajectory in the training dataset typically moves within only 2-3 blocks due to the random moving direction. These datasets might be even more challenging due to the large noisy and cluster-like trajectory pattern. Qualitative examples of the trajectories in the `Explore` dataset are shown in Figure 8.



AntMaze Medium Stitch AntMaze Large Stitch AntMaze Giant Stitch

Fig. 7: **Trajectory Examples in OGBench AntMaze Stitch Datasets.** Each Trajectory is limited to travel at most 4 blocks for dataset type `Stitch`, while at inference, the distance between the start and goal can be up to 30 in the `Giant` Maze.

We show the environment names, corresponding datasets, and the maximum environment steps for each evaluation episode in Table III. All methods are trained on the OGBench public release datasets. We slightly increase the environment steps for some environments due to the task difficulty (e.g., `Giant` Maze). For these environments, we follow the implementation in https://github.com/seohongpark/ogbench. and rerun all baselines with the increased maximum environment steps; for other environments, we directly adopt the reported success rates in the original paper.



AntMaze Medium Explore AntMaze Large Explore

Fig. 8: **Trajectory Examples in OGBench AntMaze Explore Datasets.** Trajectories in `Explore` datasets are of extremely low-quality though high-coverage. The data collection policy contains a large amount of action noise and will randomly re-sample a new moving direction after every 10 steps.

Environment	Type	Size	Dataset Name	Env Steps
`pointmaze`	`stitch`	`Medium`	pointmaze-medium-stitch-v0	1000
		`Large`	pointmaze-large-stitch-v0	1000
		`Giant`	pointmaze-giant-stitch-v0	1000
`AntMaze`	`Stitch`	`Medium`	antmaze-medium-stitch-v0	1000
		`Large`	antmaze-large-stitch-v0	2000
		`Giant`	antmaze-giant-stitch-v0	2000
`AntSoccer`	`stitch`	`Arena`	antsoccer-arena-stitch-v0	5000
		`Medium`	antsoccer-medium-stitch-v0	5000
`HumanoidMaze`	`Stitch`	`Medium`	humanoid-medium-stitch-v0	5000
		`Large`	humanoid-large-stitch-v0	5000
		`Giant`	humanoid-giant-stitch-v0	8000

TABLE III: **Our Evaluation Environments and Datasets in OGBench.**

VIII. BASELINES

We compare our approach with a wide variety of baselines, including diffusion-based trajectory planning algorithms, data augmentation based stitching algorithms, and goal-conditioned offline RL algorithms.

Particularly, we include the following methods:

- For generative planning methods, we include Decision Diffuser (DD) [2] for monolithic trajectory sampling and Generative Skill Chaining (GSC) [78] for trajectory stitching;
- For data augmentation based methods, we include stitching specific data augmentation [6] with RvS [80] and Decision Transformer (DT) [47]. Since the evaluation setting is identical, we directly adopt the reported numbers in the original paper [6].
- For offline reinforcement learning methods, we include goal-conditioned behavioral cloning (GCBC) [81], [82], goal-conditioned implicit V-learning (GCIVL) and Q-learning (GCIQL) [83], Quasimetric RL (QRL) [84], Contrastive RL (CRL) [85], and Hierarchical implicit Q-learning (HIQL) [86]. For these baselines, we follow the implementation setup established by OGBench [7] throughout our experiments.

We describe the implementation details of a few notable ones below.

Generative Skill Chaining (GSC) [78]. GSC is a recent diffusion model-based skill stitching method. We directly adopt its original score-averaging based stitching algorithm and apply it in our tasks. Specifically, when composing K trajectories $\tau_{1:K}$, GSC averages the scores of the overlapping segments between adjacent trajectories (in total $K-1$ overlapping segments in this case) prior to each denoising timestep. For a fair comparison, we re-use the same diffusion denoiser networks in CompDiffuser for every GSC experiment.

Hierarchical Implicit Q-Learning (HIQL) [86]. HIQL is a recently proposed Q-learning based method that employs a hierarchical framework for training goal-conditioned RL agents. It learns a goal conditioned value function and uses it to learn feature representations, high-level policy, and low-level policy. We follow the original implementation of the method in OGBench [7].

Goal-Conditioned Behavioral Cloning (GCBC) [81]. GCBC is a classic imitation learning-based method. In our experiments, GCBC trains an MLP that takes as input the observation state and a future goal state from the same offline trajectory and outputs a corresponding action for the agent. We use the same implementation as in OGBench [7].

IX. Additional Quantitative Results

In this section, we provide additional quantitative experiment results. In Section IX-E, we study how varying the number of composed trajectories K affects the performance of our method. In Section IX-F, we investigate the effect of inverse dynamic models of different horizons. Next, we provide a sampling time comparison of Decision Diffuser and the proposed parallel and autoregressive sampling schemes in Section IX-G. Following this, in Section IX-H, we analyze how the proposed autoregressive sampling scheme performs when using different denoising starting directions.

A. PointMaze

PointMaze in [6]. Following the original framework, the training data are curated by dividing each environment (here, maze) into several small regions, and feasible trajectories constrained within their respective regions are sampled. Possible stitching between segments is facilitated by a small overlap (one block) between different regions, which can be used to join trajectories across regions. More dataset details are provided in Appendix VII-A. We present the quantitative results on these datasets in Table IV, where we compare to goal-conditioned behavior cloning methods trained with data augmentation, Decision Diffuser, and GSC. Most baselines perform suboptimally, likely because they are unable to autonomously identify the small overlapping regions needed for trajectory stitching. In contrast, CompDiffuser successfully resolves all tasks across various maze sizes, demonstrating its ability to stitch trajectories even when the connecting regions are small.

PointMaze in OGBench [7]. In this particular setup, each trajectory in the training dataset is constrained to navigate no more than four blocks in the environment. The start and goal of each trajectory can be sampled from the entire environment provided that the travel distance between the start and goal is within four blocks. These trajectories are much shorter than the ones required for a feasible plan between a given start and goal at inference. More details about these datasets are provided in Appendix VII-B. We present the quantitative results in Table IV, where we compared our method to GSC and multiple offline RL baselines following the benchmark established in [7]. We observe that while GSC is able to perform on par with CompDiffuser in `Medium` and `Large` mazes, it struggles as the planning horizon further increases, as illustrated in the qualitative results in Figure 5. CompDiffuser significantly outperforms all baselines in `Giant` maze, demonstrating its efficacy in complex environments. Additional results are provided in Appendix X-A, X-B, and X-C.

Env	Size	RvS	RvS (SA)	RvS (GA)	DT	DT (SA)	DT (GA)	DD	GSC	Ours
PointMaze [6]	U-Maze	17±7	97±5	76±5	17±5	65±4	54±4	0±0	**100**±0	**100**±0
	Medium	1±2	55±3	21±3	20±2	55±3	62±2	30±1	93±1	**100**±0
	Large	3±4	38±5	31±5	22±2	35±2	39±5	0±0	99±2	**100**±0

Env	Type	Size	GCBC	GCIVL	GCIQL	QRL	CRL	HIQL	GSC	Ours
`PointMaze` [7]	`stitch`	`Medium`	23 ±18	70 ±14	21 ±9	80 ±12	0 ±1	74 ±6	**100**±0	**100**±0
		`Large`	7 ±5	12 ±6	31 ±2	84 ±15	0 ±0	13 ±6	**100**±0	**100**±0
		`Giant`	0 ±0	0 ±0	0 ±0	50 ±8	0 ±0	0 ±0	29±3	**68**±3

TABLE IV: **Quantitative Results on PointMaze Stitching Datasets in [6] and OGBench [7].** We compare CompDiffuser to baselines of multiple categories, including diffusion, data augmentation, and offline reinforcement learning. SA and GA stand for state augmentation and goal augmentation respectively, as described in [6]. Our results are averaged over 5 seeds and standard deviations are shown after the ± sign.

B. AntMaze and HumanoidMaze

AntMaze and HumanoidMaze. We conduct maze navigation experiments of multiple agents, ant and humanoid, using the pre-collected `Stitch` datasets provided in OGBench. The data collection strategy is identical to OGBench `PointMaze`, where each episode is constrained to travel at most 4 blocks, while at inference, a successful plan requires the agent to travel up to 30 blocks. We compare our method with the offline RL benchmark established in [7] along with the best-performing diffusion-based compositional stitching baseline GSC, as shown in Table I in the main paper. Both GSC and CompDiffuser generate plans in a planar *x-y* space while the agent follows the plans with a learned inverse dynamics model. We observe a pattern similar to the results in `PointMaze`, where CompDiffuser can consistently give high success rates as the planning horizon and complexity increase while other baselines start to collapse. To complete the study, we also conduct experiments where the full agent state is used for planning instead of the *x-y* space and provide the results in Section III-C.

AntMaze with Low-Quality Data. We evaluate CompDiffuser on OGBench `AntMaze` with a different data collection strategy, `Explore`. These datasets consist of extremely low-quality yet high-coverage data, where the data collection policy contains a large amount of action noise and will randomly re-sample a new moving direction after every 10 steps (Please see Figure 8 in Appendix for qualitative examples). Hence, each demonstration episode typically oscillates within only 2-3

blocks. Our planner needs to learn from these clustered trajectories to construct coherent plans that reach goals in large spatial distances. We present the success rate of each method in Table I.

C. AntSoccer

The `AntSoccer` environment in OGBench requires the ant agent to move a soccer ball to a designated goal in the environment, different from maze tasks that require the agent itself to reach the goal. OGBench provides two categories of trajectories in `AntSoccer Stitch` datasets: (1) ant navigates in the maze without the ball and (2) ant moves and dribbles the ball in the maze. The planner must stitch trajectories of these two skills to complete the goal-reaching objective, because the ant must reach the ball first and then dribble it to the given goal location (see Figure 14 and Figure 13).

We present experimental results in two distinct maze configurations, `Arena` and `Medium`, in Table V relative to the benchmarking provided in OGBench. Specifically, as an ablation study, we evaluate with two planner state space configurations: (1) a 4D planner consisting of the *x*-*y* location of the ant and the ball; (2) a 17D planner consisting of the *x*-*y* location of the ant and the ball along with all the joint positions of the ant. We observe that our compositional method outperforms all the baselines in both configurations. Notably, the 17D variant demonstrates slightly higher success rates, likely due to the ability of joint positions to provide more fine-grained information for ball dribbling.

Env	Size	GCBC	GCIVL	GCIQL	QRL	CRL	HIQL	GSC (4D)	Ours (4D)	GSC (17D)	Ours (17D)
antsoccer stitch	arena	34 ±4	21 ±3	5 ±2	2 ±1	2 ±1	23 ±2	41±4	55±6	65±3	**69**±3
	medium	2 ±1	1 ±0	0 ±0	0 ±0	0 ±0	8 ±2	5±2	13±1	12±2	**17**±3

TABLE V: **Quantitative Results on OGBench AntSoccer Stitch.** We evaluate two generative planners with different planning state dimensions: a 4D planner that operates on the *x*-*y* positions of the ant and the ball, and a 17D planner that additionally generates the 13 joint positions of the ant agent.

D. Ablation Studies

Replanning with CompDiffuser. Our method can also flexibly replan during a rollout, which enables the agent to recover from failure, such as when the agent fails to track the planned trajectory due to sub-optimal inverse dynamic actions. In practice, we replan if the distance between the agent's current observation and the synthesized subgoal is larger than a threshold. Please see Appendix XII-C for details. In Table VI, we present ablation studies of CompDiffuser with and without replanning in `PointMaze` and `AntMaze Stitch` datasets in OGBench. CompDiffuser outperforms the best performing baselines QRL and HIQL even without replanning in 5 out of 6 tasks. We also observe that w/ and w/o replanning yield similar performance in maze size `Medium` and `Large`, while offering significant performance boost in the more complex `Giant` maze.

Parallel vs. Autoregressive Sampling. We compare the performance of the two proposed compositional sampling schemes, parallel and autoregressive, as presented in Table VII. Autoregressive sampling consistently outperforms the parallel sampling across various tasks in plan quality, showing that the causal information flow, where each trajectory chunk is conditioned on the already-denoised (less noisy) version of previous chunk, leads to more coherent and physically consistent transitions between chunks. We include additional discussion and sampling time comparison in Appendix IX-G.

Env	Size	QRL	HIQL	Ours w/o Replan	Ours w/ Replan
PointMaze	Medium	80±12	74±6	**100**±0	**100**±0
	Large	84±15	13±6	**100**±0	**100**±0
	Giant	50±8	0±0	**53**±6	**68**±3
AntMaze	Medium	59±7	**94**±1	**92**±2	**96**±2
	Large	18±2	67±5	**76**±2	**86**±2
	Giant	0±0	21±2	**27**±4	**65**±3

TABLE VI: **Quantitative Results of CompDiffuser with and without Replanning.** We report the success rates on OGBench `PointMaze` and `AntMaze Stitch` datasets. For w/o replan, CompDiffuser only synthesizes one trajectory and executes the trajectory in a close-loop manner; for w/ replan, CompDiffuser will synthesize a new trajectory if the agent loses track of the current plan.

Env	Size	Replan	Parallel	AR
PointMaze	Large	✓	**100**±0	**100**±0
	Giant	✗	45±1	**53**±6
	Giant	✓	66±2	**68**±3
AntMaze	Large	✓	84±5	**86**±2
	Giant	✗	18±4	**27**±4
	Giant	✓	48±1	**65**±3

TABLE VII: **Quantitative Comparison of two Sampling Schemes: Parallel and Autoregressive (AR).** Parallel sampling performs on par with AR sampling in the easier `Large` maze, while AR sampling can generate trajectories of higher quality when constructing longer plans (i.e., composing more trajectories), likely due to its causal denoising strategy.

E. Number of Composed Trajectories

In Table VIII, we compare CompDiffuser with GSC over composing different numbers of trajectories (results are also shown in Figure 6). Our method performs steadily when composing different numbers of trajectories while GSC collapses.

	PointMaze Giant Stitch					
# Comp	7	8	9	10	11	12
GSC	$\mathbf{21}_{\pm 1}$	$\mathbf{21}_{\pm 3}$	$\mathbf{15}_{\pm 2}$	$\mathbf{2}_{\pm 1}$	$\mathbf{0}_{\pm 0}$	$\mathbf{0}_{\pm 0}$
CompDiffuser	$\mathbf{51}_{\pm 7}$	$\mathbf{53}_{\pm 6}$	$\mathbf{55}_{\pm 4}$	$\mathbf{56}_{\pm 2}$	$\mathbf{50}_{\pm 6}$	$\mathbf{50}_{\pm 3}$

TABLE VIII: **Quantitative Results over Different Numbers of Composed Trajectories.** We report success rates (w/o replanning) of composing 7 to 12 trajectories in the OGBench `PointMaze Giant Stitch` dataset. CompDiffuser can consistently construct feasible trajectories over various numbers of composed trajectories while GSC gradually collapses.

F. Inverse Dynamics Model

In this section, we evaluate our planner with inverse dynamics models of different horizons (results are also shown in Figure 6). Specifically, we use an MLP to implement the inverse dynamics model, which takes as input the start and goal state and outputs an action. We use the same training dataset (as to train the planner) to train the corresponding inverse dynamics model. As shown in Table IX, our method performs steadily across different inverse dynamics model horizons, showing that the subgoals generated by our planners are of high feasibility and are robust to various inverse dynamics models' configurations.

Env	Type	Size	8	12	16	20	24	28
`AntMaze`	`Stitch`	`Large`	$77_{\pm 4}$	$\mathbf{86}_{\pm 2}$	$80_{\pm 2}$	$85_{\pm 3}$	$76_{\pm 2}$	$77_{\pm 3}$
		`Giant`	$61_{\pm 5}$	$65_{\pm 3}$	$\mathbf{68}_{\pm 4}$	$63_{\pm 3}$	$60_{\pm 2}$	$63_{\pm 3}$

TABLE IX: **Quantitative Results of CompDiffuser with Inverse Dynamics Models of Different Horizons.** We present the success rates of CompDiffuser with 6 different inverse dynamics model horizons. In both environments, `Large` and `Giant`, our method performs consistently across all configurations, showing that the synthesized plans adhere to the transition dynamics and are easy to follow.

G. Sampling Time Comparison: Parallel vs. Autoregressive

In Table X, we compare the diffusion denoising sampling time of three methods: (1) monolithic model method, Decision Diffuser (DD), where we directly sample a long trajectory with the same horizon as the proposed compositional sampling method; (2) parallel compositional sampling, as shown in the left of Figure 3, where we denoise all trajectories $\tau_{1:K}$ in one batch; (3) autoregressive compositional sampling, as shown in the right of Figure 3, where we sequentially denoise each τ_k.

We observe that both parallel and AR sampling methods require more sampling time than DD probably due to (1) the the simpler and smaller denoiser network of DD; (2) time for condition encoding (our method will first encode the noisy adjacent trajectories τ_{k-1} and τ_{k+1} and feed the resulting latents to the denoiser ϵ_θ) and (3) the overhead of trajectory merging.

In addition, in our parallel sampling scheme, we stack all trajectories $\tau_{1:K}$ to one batch and feed it to the denoiser network ϵ_θ. While it indeed requires only one model forward, the batch size increases implicitly, which is probably the major reason that the sampling time of Ours (Parallel) does not proportionally decrease as the number of composed trajectories K, in comparison to Ours (AR).

Env	Type	Size	DD (Monolithic)	Ours (Parallel)	Ours (AR)
PointMaze	Stitch	Medium	0.23	1.02	1.54
		Large	0.23	1.67	3.39
		Giant	0.23	2.73	5.00

TABLE X: **Quantitative Comparison of Sampling Time** We report the time for sampling one (compositional) trajectory in three different PointMaze environments using one Nvidia L40S GPU (unit: second). The reported results are averaged over 20 sampling. In Parallel and AR (Autoregressive) mode, we use the proposed compositional sampling scheme as shown in Figure 3. Specifically, we compose 3, 6, and 9 trajectories in Maze Medium, Large, and Giant, respectively. In Decision Diffuser (DD), we directly sample one trajectory with identical length as the compositional counterparts.

H. Starting Direction of Autoregressive Compositional Sampling

In the proposed autoregressive sampling scheme described in Figure 3, inside each diffusion denoising timestep, the denoising starts from τ_1 and sequentially proceeds to τ_K (from left to right), which we denote as forward passing. Another implementation variant is to denoise in the reverse order, that is, first denoise τ_K and sequentially proceed to τ_1 (from right to left), which we denote as backward passing.

We provide a quantitative comparison of these two starting directions in Table XI. We observe that these two methods yield similar performance, demonstrating that either autoregressive sampling direction can enable effective information propagation and exchange.

Env	Type	Size	Ours (Forward)	Ours (Backward)
PointMaze	Stitch	Medium	$100_{\pm 0}$	$100_{\pm 0}$
		Large	$100_{\pm 0}$	$100_{\pm 0}$
		Giant	$55_{\pm 4}$	$56_{\pm 2}$

TABLE XI: **Quantitative Comparison of Forward and Backward Information Propagation.** We study the effect of the starting direction of the autoregressive sampling, from τ_1 vs. from τ_K. To directly study the trajectory quality, we report the results w/o replanning for both methods. Either Forward or Backward achieves similar performance, suggesting that our sampling method is robust to different sampling configurations.

X. Additional Qualitative Results

In this section, we present additional qualitative results of CompDiffuser. Videos and interactive demos are provided at our project website (see *Abstract* section for the link).

A. Composing Different Numbers of Trajectories

We present qualitative results of composing 8 to 11 trajectories in OGBench `PointMaze Giant Stitch` environments in Figure 9. We compositionally sample multiple trajectories to construct a long-horizon plan where the given start is at the bottom-left corner and the given goal is at the top-right corner. For clearer view, we present the results before applying trajectory merging (i.e., we show each individual trajectory of $\tau_{1:K}$) and use different colors to highlight different trajectories.

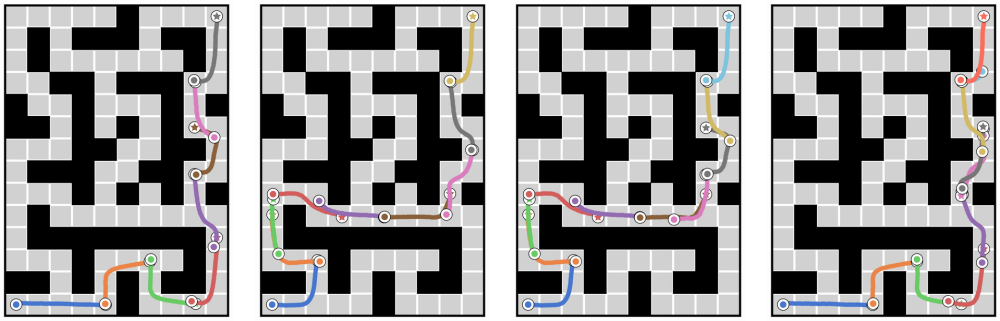


Fig. 9: **Composing Different Numbers of Trajectories at Inference.** Our method is only trained on short trajectory segments that travel at most 4 blocks, while is able to compositionally generate coherent long trajectories given the start (circle) and goal (star). When K is smaller and barely sufficient to reach the goal (e.g., 8), the length of the overlapping part between segments decreases so as to extend the travel distance of the overall compositional plan. In contrast, if given a larger K (e.g., 11), some parts of the compositional plan might travel back and forth to consume the extra length.

B. Diverse Trajectory Morphology

The proposed compositional sampling method allows direct generalization to long-horizon planning tasks at test time through its noisy-sample conditioning and bidirectional information propagation design. Meanwhile, this sampling approach also preserves the multi-modal nature of the diffusion model, enabling a diverse range of trajectory morphology. As shown in Figure 10, given a similar start and goal pair, our method can construct trajectories that reach the goal via various possible paths. With such multi-modal flexibility, the proposed sampling process can be further customized with specific preferences by integrating additional test-time steering techniques.

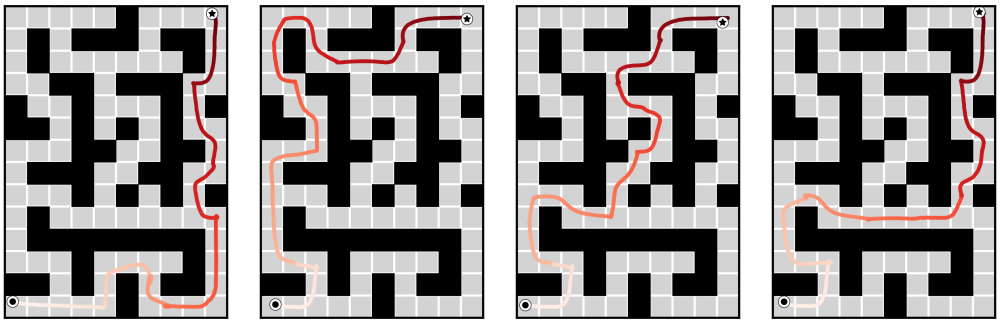


Fig. 10: **Diverse Trajectory Morphology.** We present four trajectories with similar start and goal in OGBench `PointMaze Giant Stitch`. All trajectories are generated by CompDiffuser, composing 9 trajectories. CompDiffuser preserves the multi-modal nature of diffusion models and is able to flexibly sample trajectories of diverse morphology.

C. Planning Results on Different Tasks.

In Section X-B above, we present multiple trajectories of Task 1 in OGBench `PointMaze Giant`. In this section, we present qualitative results of the following Task 2 to Task 5, as in Figure 11. We set the number of composed trajectories K

to 9 in all tasks. The state state is shown by the black circle and the goal state is shown by the black star. Our method successfully constructs feasible plans for various start-goal configurations across different spatial distances.

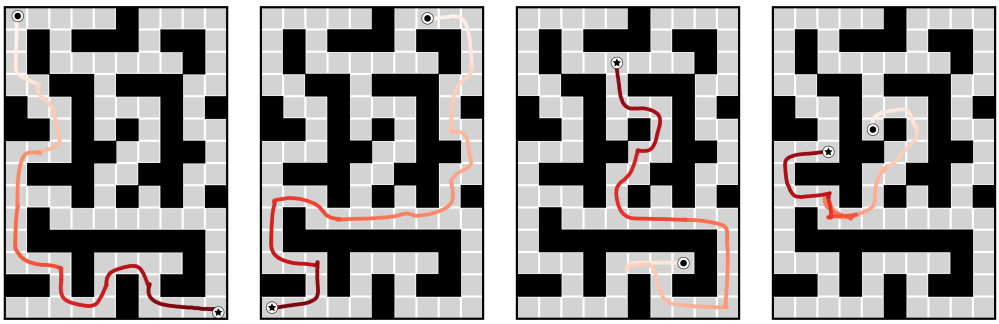


Fig. 11: **Different Tasks in OGBench PointMaze Giant Stitch.** We present qualitative plans by CompDiffuser on OGBench Task 2-5 (See Figure 10 for results of Task 1). We set the number of composed trajectories to 9 for all the tasks above. The black circle denotes the start state and the black star denotes the goal state. In task 2 and 3, our method effectively constructs long horizon plans that reach the goals on the opposite side of the environment (despite being trained only on trajectories that travel at most 4 blocks). In comparison, Task 4 and 5 feature a relatively smaller spatial gap between the start and goal, thus requiring a shorter planning horizon. Our method still generalizes to these tasks by generating plans that traverse additional distance or leveraging back-and-forth movements to consume the extra plan length.

D. Compositional Planning in High Dimensional Space

In this section, we present additional high dimensional trajectories generated by CompDiffuser in OGBench `AntMaze Large Stitch` environment. Similar to other experiments in the paper, the models are trained on the corresponding OGBench public release datasets.

AntMaze Large Stitch 29D. We train CompDiffuser on the state space of *x*-*y* position along with the ant's joint positions and velocities, resulting in a 29D planning task. Note that CompDiffuser is only trained on short trajectory segments (we set its horizon to 160), while at test time, we compose 5 trajectories to directly construct trajectory plans of horizon 584. We present additional qualitative results in Figure 12. Note that we uniformly sub-sample the length of the trajectory to 50 for clearer view. Corresponding quantitative results are reported in Table II.

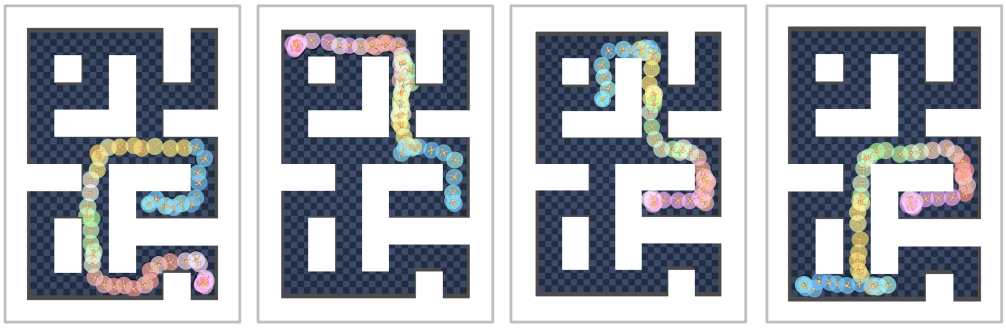


Fig. 12: **Compositional Planning in 29D on OGBench AntMaze Large Stitch Tasks.** Original trajectory plans are much denser and we uniformly sub-sample 50 states from the original generated trajectories for better view. Five trajectories are composed as shown by different colors: the blue one indicates the first trajectory τ_1 and the purple one indicates the fifth trajectory τ_5.

E. Compositional Planning in AntSoccer

We provide additional qualitative results in OGBench `AntSoccer Arena Stitch` environment in Figure 13 and OGBench `AntSoccer Medium Stitch` environment in Figure 14. In this task, the ant is initialized to the location of the blue circle and is tasked to move the ball to the goal location indicated by the pink circle. Hence, the ant needs to first reach the ball from the far side of the environment and dribble the ball to the goal.

However, such long-horizon trajectories (the ant reaches the ball and then dribbles the ball to the goal) do not exist in the training dataset. The training dataset only contains two distinct types of trajectories: (1) the ant moves in the environment without the ball, (2) the ant moves while dribbling the ball. Therefore, the planner needs to generalize and stitch in a zero-shot manner – constructing an end-to-end trajectory that first approaches the ball and then dribbles the ball to the goal.

We train CompDiffuser on the state space of the ant's *x*-*y* position and joint positions along with the *x*-*y* position of the ball, resulting in a 17D planning task. Similar to Figure 14, we present each individual trajectory $\tau_{1:K}$ in Figure 13 for clearer view. Corresponding quantitative results are provided in Table V.

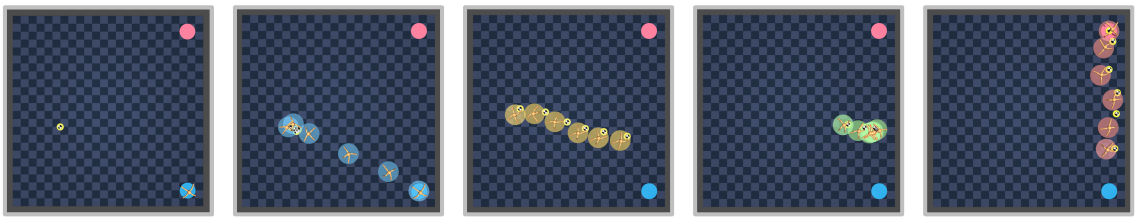


Fig. 13: **Compositional Planning on OGBench AntSoccer Arena Stitch.** We present the initial state for planning and each individual trajectories $\tau_{1:4}$ above. The start position of the ant is shown by the blue circle (bottom right) and the goal is to move the ball to the pink circle (upper right). We compositionally sample 4 trajectories (as shown from left to right), which will then be merged to form a long-horizon plan. The ball is highlighted with a small yellow circle. Our compositional sampling method effectively stitches two different types of trajectories and generalizes to more difficult tasks unseen in the training data.

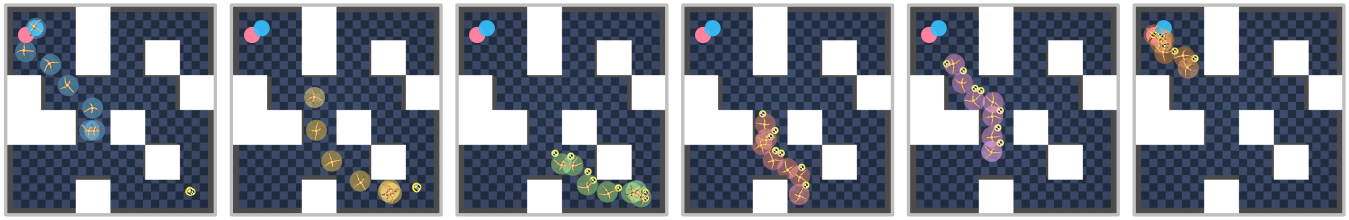


Fig. 14: **Qualitative Plan generated by CompDiffuser in OGBench AntSoccer Medium Stitch.** The initial position of the ant is shown by the blue circle and the goal is to move the ball to the pink circle. We plot each individual trajectory $\tau_{1:6}$ separately (as shown from left to right) and mark the ball with a yellow circle for clearer view. These generated trajectories will be merged to form a long-horizon plan τ_{comp}. Note that our model is only trained on two different types of short trajectories: 1) the ant moves without dribbling the ball; 2) the ant moves while dribbling the ball. At test time, the proposed compositional sampling method can stitch these two types of trajectories and construct end-to-end plans of longer horizon to solve the more difficult task – first navigate to the ball from the far side and then dribble the ball to the goal position.

XI. Failure Mode Analysis

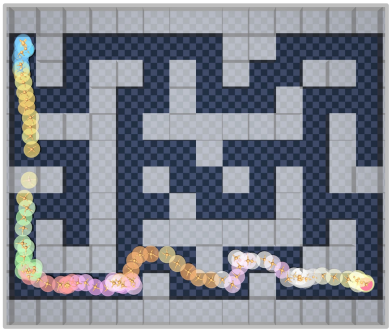
In this section, we present several failure cases of our method along with analyses and qualitative examples of these failure modes. Our proposed framework consists of two components: a generative planner and an inverse dynamics model. We outline and discuss several failure modes below.

A. Infeasible Generated Plan

As the number of composed trajectories K increases, our method may show inconsistencies in the trajectory plan, especially when planning in high dimensional state space. We observe that although the start and goal conditioning can consistently ensure that the composed plan begins at the initial state and terminates at the provided goal, the intermediate trajectories may include implausible transitions, such as trajectories that jump over walls, making the overall plan infeasible. However, empirically these failures usually occur at considerably higher K as compared to baseline methods (e.g., GSC, see Table IV and Table I).

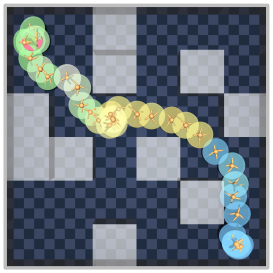
In Figure 15, we present a qualitative example of infeasible plan in OGBench `AntMaze Giant Stitch` when planning in 29D space (ant's x-y position, joint positions, and joint velocity) and composing 9 trajectories. The original white walls are rendered transparent for better view of infeasible states. Though the generated compositional plan is mostly valid, the second trajectory τ_2, as shown in yellow, is infeasible as it passes through walls. This is probably due to the suboptimal coordination among trajectories in the compositional sampling process.

As illustrated in the figure, certain trajectories (e.g., the neighboring blue, green, and red ones) barely proceeds, moving only 2-3 blocks. To bridge the resulting spatial gap, the intermediate yellow trajectory must span a much longer distance. However, the training data does not contain such long-horizon trajectories, making it difficult for a single trajectory to extend to such length, which finally results in an o.o.d sample that goes through walls. We could potentially mitigate this issue with rejection or guided sampling techniques to select feasible candidate plans.

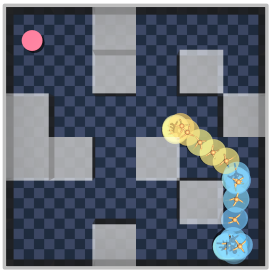


Infeasible Plan: Passing through Walls

Fig. 15: **Failure Mode: Infeasible Plan.** We increase the transparency of the original white walls for better view of infeasible states. The start state is at the top left corner and the goal state is at the bottom right corner. Nine trajectories are composed, highlighted by different colors. The second trajectory τ_2 (shown in yellow) is infeasible and passes through walls. This is probably due to the suboptimal coordination between trajectories in the compositional sampling process: the neighboring blue and green trajectories barely progress, leaving a long o.o.d gap for the yellow trajectory to fill.



Feasible Plan



Rollout: Suboptimal Actions

Fig. 16: **Failure Mode: Suboptimal Inverse Dynamics Actions.** We present a failure case of planning in 15D space (ant's x-y position and joint positions) in `AntMaze Medium Stitch`. Left: the compositional plan where three trajectories (represented in blue, yellow, and green) are composed. We sub-sample the plan to 36 states for visualization (the original plan is dense with over 300 states). Right: the corresponding environment execution rollout of the compositional plan. Though the synthesized plan is valid and successfully reaches the goal, the inverse dynamics model may fail, as illustrated on the right which gets stuck in a right turn and is unable to proceed. Further incorporating some effective replanning strategies or employing more robust inverse dynamics models could potentially mitigate these failure scenarios.

B. Suboptimal Inverse Dynamics Model

In our experiments, we observe that even if CompDiffuser synthesizes a feasible trajectory for the agent to follow, the agent might not successfully reach the goal due to the error by the inverse dynamics model. For example, the agent might bump into walls due to unstable locomotion or get stuck in some local region, yet the synthesized plan is collision-free and coherent.

In implementation, we use a simple MLP to parameterize the inverse dynamics model and train it with a regression MSE loss. We believe that more optimized model architecture or specific finetuning might further boost the performance of the inverse dynamics model, hence boosting the overall performance of our method. We deem that out of the scope of this work.

In Figure 16, we present a qualitative example of failure due to the inverse dynamics model in OGBench `AntMaze Giant Stitch`. The planning is in 15D space (ant's *x*-*y* position and joint positions) and composes 3 trajectories. While we observe that the synthesized plan is feasible and successfully reaches the goal, the environment execution rollout fails during the yellow trajectory. In this instance, the ant agent fails to execute the right turn, loses track of subsequent subgoals, and becomes trapped in a local region. To address this issue, incorporating effective replanning strategies could help the agent recover (since the new plan will start from the current trapped state). In addition, we deem that employing more robust or specialized inverse dynamics models may further mitigate such failure scenarios.

C. Suboptimal Number of Composed Trajectories K

In our current implementation, the number of composed trajectories K needs to be manually specified at test time. As shown by the quantitative results Table VIII and qualitative results Figure 9, a relatively larger K does not significantly affect the model performance as the extra plan length will be consumed by staying or circulating within certain valid regions in the environment. However, an aggressively smaller K might lead to failure plans – since the overall planning horizon becomes insufficient to cover the substantial spatial gap between the start and the goal.

In this section, we provide qualitative examples of infeasible plans due to impractical K. In Figure 17, we show each individual trajectory τ_k generated by our compositional sampling method when K ranges from 3 to 6. Note that a feasible horizon to reach the goal from the start (bottom left) to the goal (upper right) requires composing at least 8 trajectories, i.e., $K = 8$.

Therefore, while the generated trajectories can begin at the start state and terminate at the goal state, the intermediate segments become disconnected since there are too few trajectories to bridge the significant spatial gap (given that the training data contains only short trajectories). Nonetheless, we observe that the overall flow of the trajectories is directed toward the goal, and as K increases, the plan's structure gradually becomes more feasible.

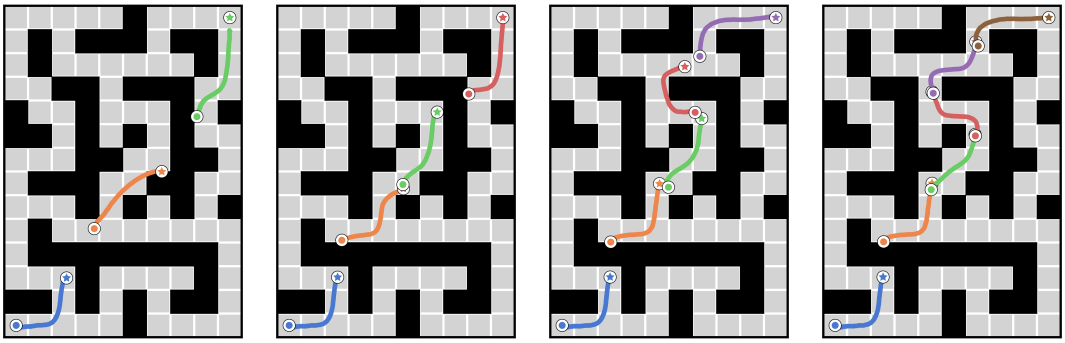


Fig. 17: **Failure Model: Suboptimal Number of Composed Trajectories K.** Our method may generate infeasible plans if K, the number of composed trajectories, is set significantly below the required minimum. For example, in the planning task illustrated above, the start state is located in the bottom left corner while the goal state is at the upper right corner. A feasible plan for this task typically requires composing at least 8 trajectories (i.e., $K = 8$). We present qualitative examples of plans when K is smaller than the minimum threshold. Though the first and last trajectory segments correctly attach to the start and goal states, the intermediate trajectories are disconnected due to the insufficient plan length. However, the overall plan still progresses towards the goal, which may provide useful guidance signals to the agent.

XII. Implementation Details

Software: The computation platform is installed with Ubuntu 20.04.6, Python 3.9.20, PyTorch 2.5.0.

Hardware: We use 1 NVIDIA GPU for each experiment.

A. Our Conditional Diffusion Model

In this section, we introduce detailed implementation of our conditional diffusion model $\epsilon_\theta(\tau^t, t \mid \text{st\_cond}, \text{end\_cond})$.

Model Inputs and Outputs. Our diffusion model takes as input the noisy trajectory τ^t, diffusion noise timestep t, and the start condition st_cond and end condition end_cond for τ^t. st_cond can be either noisy chunk or start state q_s and end_cond can be either noisy chunk or goal state q_g. We use the predicting τ^0 formulation to implement this network, i.e., the network directly predicts the clean sample τ^0.

Implementation of Noisy Chunk Conditioning. As described in Section II, we only train *one* diffusion model ϵ_θ that is able to condition on both the start state q_s, goal state q_g, and noisy chunk τ^t_{k-1}, τ^t_{k+1}, such that in test time we can use the proposed compositional sampling approach to construct long-horizon plans. This unified one model design eliminate the need for manual task subdivision (across multiple models) or reliance on predefined planning skeleton, thereby enabling holistic end-to-end planning.

In practice, we can implement the noisy neighbor conditioning $\mathcal{L}_{\text{nbr}}$ simply using the overlapping part between the two chunks instead of a full chunk τ_{k-1}/τ_{k+1} conditioning, because the overlapping part is sufficient to ensure the connectivity between two adjacent trajectories. Such design further simplifies the training procedure: instead of dividing a training trajectory τ to K chunks, we only need to sample a noisy sub-chunk from τ itself.

Specifically, assume τ^t is the noisy trajectory to be denoised and $\hat{\tau}^t$ is another independent noisy version of τ also at noisy timestep t. We denote the length of τ as h and the length of the overlapped part as h_{o}, then we can use $\hat{\tau}^t[0 : h_{\text{o}}]$ and $\hat{\tau}^t[h - h_{\text{o}} : h]$ as the noisy sub-chunks for st_cond and end_cond during training, respectively. Hence, when inference, the end_cond for τ_1 can be set to $\hat{\tau}^t_2[0 : h_{\text{o}}]$ (the front chunk of the second trajectory τ_2) and the st_cond for τ_2 can be set to $\hat{\tau}^t_2[0 : h_{\text{o}}]$ (the tail chunk of the first trajectory τ_1).

Model Architecture. For planning in 2D *x-y* space, we follow Decision Diffuser [2] (https://github.com/anuragajay/decision-diffuser/) and use a conditional U-Net as the denoiser network. For planning in high dimension state space, we use a DiT [87] based transformer [88] as the denoiser network (https://github.com/facebookresearch/DiT/).

Training Pipeline. We provide detailed hyperparameters for training our model on `PointMaze Giant Stitch` environment in Table XII. We do not apply any hyperparameter search or learning rate scheduler. Please refer to our codebase for more implementation details.

Hyperparameters	Value
Horizon	160
Diffusion Time Step	512
Probability of Condition Dropout	0.2
Iterations	1.2M
Batch Size	128
Optimizer	Adam
Learning Rate	2e-4
U-Net Base Dim	128
U-Net Encoder Dims	(128, 256, 512, 1024)

TABLE XII: Hyperparameters for Training on `PointMaze Giant Stitch` environment.

Inference Pipeline. In Table XIII, we present the single model horizon (length of individual τ_k) and the inference-time number of composed trajectories K corresponding to the reported results in Table IV, Table I and Table V.

For each evaluation problem, we generate B samples in a batch and we use a simple heuristic to select one sample as the output plan. Specifically, we compute the L2-distance of each overlapping parts in the generated trajectory segments $\tau_{1:K}$. The one with the smallest average distance will be adopted as the output plan, in the sense that a small distance in the overlapping parts indicates better coherency between adjacent trajectories. we deem that developing some more advanced

inference-time methods with CompDiffuser may be an interesting future research direction, such as probability or density based plan selection methods or compositional sampling with flexible preference steering.

Environment	Type	Size	Single Model Horizon	# of Composed Trajectories
PointMaze [6]	-	`U-maze`	40	5
		`Medium`	144	5
		`Large`	192	5
`pointmaze`	`stitch`	`Medium`	160	3
		`Large`	160	5
		`Giant`	160	8
`AntMaze`	`Stitch`	`Medium`	160	3
		`Large`	160	6
		`Giant`	160	9
`AntMaze`	`Explore`	`Medium`	192	5
		`Large`	192	6
`AntSoccer`	`stitch`	`Arena`	160	5
		`Medium`	160	6
`HumanoidMaze`	`Stitch`	`Medium`	336	4
		`Large`	336	6
		`Giant`	336	11

TABLE XIII: **Number of Composed Trajectories for Each Evaluation Environment.** Our diffusion models are trained with a short horizon as listed in the *Single Model Horizon* column. In test time, we compositionally generate multiple such short trajectories to enable trajectory stitching and construct plans of much longer horizon.

B. Trajectory Merging

As introduced in Method Section II and Algorithm 2, the generated trajectories $\tau_{1:K}$ are mutually overlapped and we merge these K trajectories to form a long-horizon compositional trajectory τ_{comp}. In this section, we describe the implementation of the exponential trajectory blending technique which we use for merging.

We directly leverage the classic exponential trajectory blending formulation. For simplicity, let τ_1 and τ_2 denote the trajectories to blend, $\tau_1[t]$ denote the t-th state in τ_1, t_{start} and t_{end} denote the start and end index of the region to apply blending. Note that, in practice, we *only* blend the overlapped part between two adjacent trajectories. We provide the equation for exponential blending below,

$$\tau_{\text{comp}}[t] = w(t) * \tau_1[t] + (1 - w(t)) * \tau_2[t], \quad \text{where} \quad w(t) = \frac{e^{-\beta\left(\frac{t - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}}\right)} - e^{-\beta}}{1 - e^{-\beta}} \tag{5}$$

We set $\beta = 2$ across all our experiments. In practice, various other trajectory blending techniques can also be directly applied, such as cosine blending and linear blending.

C. Replanning

In this section, we describe the detailed implementation of replanning. While our method is designed to directly generate an end-to-end trajectory from the given start state to the goal state, replanning can be performed at any given timesteps during a rollout. Specifically, we initiate replanning if the agent loses track of the current subgoal, i.e., the L2 distance between the agent and the subgoal is larger than a threshold.

In a larger maze, the required number of composed trajectories, denoted as K, is usually large due to the distance between the start state and the goal state. However, if we keep replanning with a similar large K even when the agent is already close to the goal, the generated trajectory might travel back and forth to consume the unnecessary intermediate length (see (2) and (3) in Figure 11), thus delaying the agent's progress toward the goal.

To address this, we use a receding scheme for K to encourage faster convergence to the goal. Let K denote the number of composed trajectories of the current plan (the plan that the agent is following), h_{comp} denote the length of the current

plan, h_{exe} denote the length of the current plan that the agent already executes in the environment. The number of composed trajectories used for replanning K_{replan} is given by

$$K_{\text{replan}} = \texttt{ceil}\Big(\ (1 - \frac{\max(h_{\text{exe}} - \delta, \ 0)}{h_{\text{comp}}}) * K \ \Big) \tag{6}$$

where δ is a hyper-parameter that controls the convergence speed to the goal, for example, K_{replan} will decrease faster if δ is set to a small (or negative) number while K_{replan} will decrease very slowly if δ is a large positive number.

ACKNOWLEDGMENT

We would like to thank Zilai Zeng for helpful early discussion on trajectory stitching and feedback of the manuscript.