Neural MP: A Policy Learning Framework for Generalist Neural Motion Planners

Murtaza Dalal* **Jiahui Yang*** **Russell Mendonca**
Youssef Khaky **Ruslan Salakhutdinov** **Deepak Pathak**
Carnegie Mellon University

Abstract: The current paradigm for motion planning generates solutions from scratch for every new problem, which consumes significant amounts of time and computational resources. For complex, cluttered scenes, motion planning approaches can often take minutes to produce a solution, while humans are able to accurately and safely reach any goal in seconds by leveraging their prior experience. We seek to do the same by applying data-driven learning at scale to the problem of motion planning. Our approach builds a large number of complex scenes in simulation, collects expert data from a motion planner, then *distills* it into a reactive generalist policy. We then combine this with lightweight optimization to obtain a safe path for real world deployment. We perform a thorough evaluation of our method on 64 motion planning tasks across four diverse environments with randomized poses, scenes and obstacles, in the real world, demonstrating an improvement of 23%, 17% and 79% motion planning success rate over state of the art sampling, optimization and learning based planning methods. All code, models and datasets will be released on acceptance. Video results available at `mihdalal.github.io/neuralmotionplanner`.

Keywords: Neural Motion Planning, Sim2real, Procedural Scene Generation

*Equal contribution.

1 Introduction

Motion planning is a longstanding problem of interest in robotics, with previous approaches ranging from potential fields [1, 2, 3], sampling (RRTs and Roadmaps) [4, 5, 6, 7, 8, 9, 10], search (A*) [11, 12, 13] and trajectory optimization [14, 15, 16, 17]. Despite being ubiquitous, these methods are often slow at producing solutions since they largely plan from scratch at test time, re-using little to no information outside of the current problem and what is engineered by a human designer. On the other hand, humans can generate motions in a closed loop manner, move quickly, react to various dynamic obstacles, and generalize across a wide distribution of problem instances. Rather than planning from scratch, people draw on their vast amounts of experience interacting with their environment while reactively adjusting their movements in order to quickly and efficiently move about the world. How can we create motion planners with similar properties? In this work, we argue that *distillation* at scale is the answer: we can *distill* the planning process into a **reactive, generalist** neural policy.

Our core contribution is a SOTA motion planner that runs zero-shot on any environment, with more accuracy and significantly less execution time. We demonstrate that large scale data generation in simulation can enable training generalist policies that can be successfully deployed for real-world tasks. To our knowledge, Neural MP is the first work to demonstrate that such a neural policy can generalize to a broad set of out-of-distribution of real-world environments, generalizing across tasks with significant variation across poses, objects, obstacles, backgrounds, scene arrangements, in-hand objects, and start/goal pairs. Specifically, we propose a simple, scalable approach for training and deploying fast, general purpose neural motion planners: 1) **large-scale procedural scene generation** with diverse environments in realistic configurations, 2) **multi-modal sequence modeling** for fitting

8th Conference on Robot Learning (CoRL 2024), Munich, Germany.

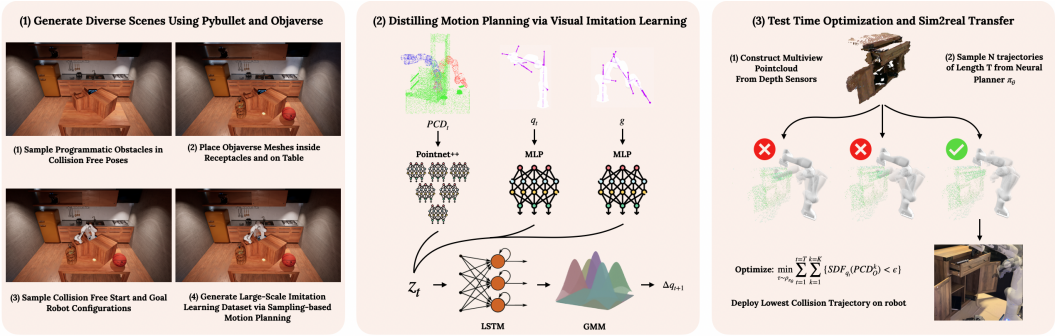


Figure 1: **Method Overview**: We present Neural Motion Planners, which consists of 3 main components. **Left**: Large Scale data generation in simulation using expert planners **Middle**: Training deep network models to perform fast reactive motion planning **Right**: Test-time optimization at inference time to improve performance.

to sampling-based motion planning data and 3) **lightweight test-time optimization** to ensure safe, and reliable deployment in the real world. We execute a thorough real-world empirical study of motion-planning methods, evaluating our approach on **64** real world motion planning tasks across **four** diverse environments, demonstrating a motion planning success rate improvements of **23%** over sampling-based, **17%** over optimization-based and **79%** over neural motion planning methods.

2 Neural Motion Planning

Our approach enables generalist neural motion planners, by leveraging large amounts of training data generated in simulation via expert planners. The policies can generalize to out-of-distribution settings by using powerful deep learning architectures along with diverse, large-scale training data. To further improve the performance of these policies at deployment, we leverage test time optimization to select the best path out of a number of options. We now describe each of these pieces in more detail.

2.1 Large-scale Data Generation

One of the core lessons of the deep learning era is that the quality and quantity of data is crucial to train broadly capable models. We leverage simulation to generate vast datasets for training robot policies. Our approach generates assets using programmatic generation of primitives and by sampling from diverse meshes of common objects. These assets are combined to create complex scenes resembling real world scenarios (Fig. E.1). Please check Alg. 1 in Appendix for more details.

Complex scene generation To generate a large enough number of diverse environments, we take the approach of *procedural* scene generation, using a set of six *parametrically variable* categories - shelves, cubbies, microwaves, dishwashers, open boxes, and cabinets. These categories are representative of a large set of objects in everyday scenarios that robots encounter and have to avoid colliding with. Each category instance is constructed using a combination of primitive cuboid objects and is parameterized by category specific parameters which define the asset. In addition to this, we augment our dataset with objects sampled from the recently proposed large-scale 3D object dataset, Objaverse [18]. This dataset contains a wide variety of objects that the neural planner is likely to observe during deployment, such as comic books, jars, record players, etc. This further extends the diversity and realism of our dataset.

Motion Planning Experts: To collect expert data in the diverse generated scenes, we leverage SOTA sampling-based motion planners due to their (relative) speed as well as ease of application to a wide array of tasks. Specifically, we use Adaptively Informed Trees [10] (AIT*), an almost-surely asymptotically optimal sampling-based planner to produce high-quality plans using privileged information, namely access to a perfect collision checker in simulation. In order to obtain challenging expert trajectories that maneuver around obstacles, we generate tight-space configurations by sampling end-effector poses from specific locations (*e.g.*, inside a cubby or microwave) and by using inverse kinematics to derive the joint pose. Tight-space configurations are sampled 50% of the time, to ensure that we collect trajectories where the robot moves around obstacles, as opposed to taking straight line

paths between nearby free space points. Additionally, we spawn objects grasped in the end-effectors, with significant randomization including boxes, cylinders, spheres or even Objaverse meshes.

2.2 Generalist Neural Policies

We would like to obtain agents that can use diverse sets of experiences to plan efficiently in new settings. In order to build generalist neural motion planning policies, we need an observation space amenable to sim2real transfer, and utilize an architecture capable of absorbing vast amounts of data.

Observations: We begin by addressing the sim2real transfer problem, which requires considering the observation and action spaces of the trained policy. With regards to observation, point-clouds are a natural representation of the scene for transfer [19, 20, 21, 22, 23], as they are 3D points grounded in the base frame of the robot and therefore view agnostic, and largely consistent between sim and real. We include proprioceptive and goal information in the observations, consisting of the current joint angles q_t, the target joint angles g, in addition to the point-cloud PCD.

Network Architecture: We require an architecture capable of scaling with data while performing well on multi-modal sequential control problems, *e.g.* motion planning. To that end, we design our policy π (visualized in Fig. 1) to be a sequence model to imitate the expert using a notion of history which is useful for fitting privileged experts using partially observed data [24]. In principle, any sequence modeling architecture could be used, but in this work, we opt for LSTMs for their fast inference time and comparable performance to Transformers on our datasets (see Appendix). We operate the LSTM policy over joint embeddings of PCD_t, q_t, and g with a history length of 2. We encode point-clouds using PointNet++ [25], while we use MLPs to encode q_t and g_t. We follow the design decisions from MπNets regarding point-cloud observations: we segment the robot point-cloud, obstacle point-cloud and the target robot point-cloud before passing it to PointNet++. For each time-step, we concatenate the embeddings of each of the observations together into one vector and then pass them into the LSTM for action prediction. For the output of the model, note that sampling-based motion planners such as AIT* are heavily multi-modal: for the same scene they may give entirely different plans for different runs. As a result, we require an expressive, multi-modal distribution to effectively capture such data, for which we use a Gaussian Mixture Model (GMM). Specifically, Neural MP predicts a GMM distribution over delta joint angles (Δq_{t+1}), which are used to compute the next target joint way-point during deployment: $q_{t+1} = q_t + \Delta q_{t+1}$. As we show in our experiments, for fitting to sampling-based motion planning, minimizing the negative log-likelihood of the GMM outperforms the PointMatch loss from MπNets, Diffusion [26] and Action-chunking [27]

2.3 Deploying Neural Motion Planners

Test time Optimization While our base neural policy is capable of solving a wide array of challenging motion planning problems, we would still like to ensure that these motions are safe to be deployed in real environments. We enable this property by combining our learned policy with a simple light-weight optimization procedure at inference time. This relies on a simple model that assumes the obstacles do not move and the controller can accurately reach the target way-points. Given world state $s = [q, e]$ (e is the environment state), the predicted world state is $s' = [q + \hat{a}$, e] where $\hat{a}$ is the policy prediction. With this forward model, we can sample N trajectories from the policy using the initial scene point-cloud to provide the obstacle representation and estimate the number of scene points that intersect the robot using the linear forward model. We then optimize for the path with the least robot-scene intersection in the environment, using the robot Signed Distance Function (SDF). Specifically, we optimize the following objective at test time: $\min_{\tau \sim \rho_{\pi_\theta}} \sum_{t=1}^{t=T} \sum_{k=1}^{k=K} \mathbb{1}\{SDF_{q_t}(PCD_O^k) < \epsilon\}$, in which ρ_{π_θ} is the distribution of trajectories under policy π_θ with a linear model as described above, PCD_O^k is the kth point of the obstacle point-cloud (with max $K = 4096$ points) and SDF_{q_t} is the SDF of the robot at the current joint angles. In practice, we optimize this objective with finite samples in a single step, computing the with minimal objective value by selecting the path with minimal objective value across 100 trajectories. We include a detailed analysis of the properties of our proposed test-time optimization approach in the Appendix.

3 Experimental Results

To guide our evaluation, we pose a set of experimental questions. 1) Can a single policy trained in simulation learn to solve complex motion planning tasks in the real world? 2) How does Neural MP compare to SOTA neural planning, sampling-based and trajectory optimization planning approaches? 3) How well does Neural MP extend to motion planning tasks with objects in-hand? 4) Can Neural MP perform dynamic obstacle avoidance?

Free Hand Motion Planning In this set of experiments, we evaluate motion planning the robot's hand is empty (Table 2). We find that our base policy on its own performs comparably to AIT*-80s (66.67% vs. 72.92%) while only using 1s of planning time. When we include test-time optimization (3s of planning), we find that across all three tasks, Neural MP achieves the best performance with a 95.83% success rate. In general, we find that Bins is the easiest task, with the sampling/optimization-based methods performing well, Shelf is a bit more difficult as it requires simultaneous vertical and horizontal collision avoidance, while Articulated is the most challenging task as it contains a diverse set of obstacles and tight spaces. Neural MP performs well across each task as it is trained with a diverse set of parametric objects that cover the types of real-world obstacles we encounter and it also incorporates complex meshes which cover the irregular geometries of the additional objects we include.

	Bins	Shelf	Articulated	Average
Sampling-based Planning:				
`AIT*-80s` [10]	93.75	75.0	50.0	72.92
`AIT*-10s (fast)` [10]	75.0	37.5	25.0	45.83
Optimization-based Planning:				
`Curobo` [17]	93.75	81.25	62.5	79.17
Neural:				
MπNets [19]	18.75	25.0	6.25	16.67
Ours-Base Policy	81.25	75.0	43.75	66.67
Ours	**100**	**100**	**87.5**	**95.83**

Figure 2: Neural MP performs best across each scene free-hand motion planning task, demonstrating greater improvement as the task complexity grows.

In-Hand Motion Planning In this experiment, we extend our evaluation to motion planning with objects in-hand, a crucial capability for manipulation. We evaluate Neural MP against running the neural policy without test time optimization and without including any Objaverse data, achieving 81% performance vs. 31% and 44%. We visualize an example trajectory in Fig. G.2. Our method performs well on in-distribution objects such as the book and board game, but struggles on out of distribution objects such as the toy sword, which is double the size of objects at training time. We additionally deploy our method on significantly out of distribution objects such as the bookcase (Fig. G.2c) and find that Neural MP generalizes well to in-hand motion planning tasks such as inserting the book in the right rung.

This experiment also serves as an ablation of our method, demonstrating the importance of test time optimization on out of distribution scenarios. For these tasks, the base policy performance results in a large number of collisions as two of the in-hand objects are out of distribution (sword and board game), but the optimization step is able to largely remove them and produce clean behavior that reaches the target without colliding. Additionally, this experiment demonstrates that the Objaverse data is crucial for the success of our method in the real world. Models trained only on cuboid-based parametric assets fail to generalize to the complexity of the real world (43.75%) while those trained on Objaverse perform well (81.25%), highlighting the importance of incorporating Objaverse meshes into scene generation.

Dynamic Motion Planning In many real-world scenarios, the environment may be changing as the motion planner is acting. We test how well Neural MP can motion plan in such settings by introducing obstacles into the environment while the motion planner is moving to a goal. We evaluate the motion planner on four different goals with three different added obstacles (drawer, monitor and pot). To handle dynamic obstacles, we run the neural motion planner closed loop and perform single-step test-time optimization. We compare against MπNets and find that Neural MP performs 53% better (63.33% vs. 10%), performing particularly well on the drawer and pot object while struggling on the monitor object which is significantly taller. We also qualitatively evaluate Neural MP on two significantly more challenging tasks in which we continuously move the obstacle into the robot's path and demonstrate that it can adjust its behavior to avoid collisions while reaching the goal.

Acknowledgments

We thank Shikhar Bahl, Ananye Agarwal, and Mihir Prabhudesai for their insightful discussions and feedback. We additionally thank Kenny Shaw, Rishi Veerapaneni, Ananye Agarwal, and Shikhar Bahl for feedback on early drafts of this paper. This work was supported in part by the NSF Graduate Fellowship, Apple, ONR MURI N00014-22-1-2773 and AFOSR FA9550-23-1-0747.

References

[1] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[2] C. W. Warren. Global path planning using artificial potential fields. In *1989 IEEE International Conference on Robotics and Automation*, pages 316–317. IEEE Computer Society, 1989.

[3] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807. IEEE, 1993.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[5] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[6] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1, 2000. doi:10.1109/ROBOT.2000.844107.

[7] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

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

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074, 2015. doi:10.1109/ICRA.2015.7139620.

[10] M. P. Strub and J. D. Gammell. Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3191–3198. IEEE, 2020.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] M. Likhachev, G. J. Gordon, and S. Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003.

[13] S. Koenig and M. Likhachev. A new principle for incremental heuristic search: Theoretical results. In *ICAPS*, pages 402–405, 2006.

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

[15] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[16] A. D. Dragan, N. D. Ratliff, and S. S. Srinivasa. Manipulation planning with goal sets using constrained trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 4582–4588. IEEE, 2011.

[17] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. Curobo: Parallelized collision-free robot motion generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8112–8119. IEEE, 2023.

[18] M. Deitke, D. Schwenk, J. Salvador, L. Weihs, O. Michel, E. VanderBilt, L. Schmidt, K. Ehsani, A. Kembhavi, and A. Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023.

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

[20] S. Christen, W. Yang, C. Pérez-D'Arpino, O. Hilliges, D. Fox, and Y.-W. Chao. Learning human-to-robot handovers from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[21] Y. Jiang, C. Wang, R. Zhang, J. Wu, and L. Fei-Fei. Transic: Sim-to-real policy transfer by learning from online correction. *arXiv preprint arXiv: Arxiv-2405.10315*, 2024.

[22] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

[23] C. Wu, X. Bi, J. Pfrommer, A. Cebulla, S. Mangold, and J. Beyerer. Sim2real transfer learning for point cloud segmentation: An industrial application case on autonomous disassembly. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4531–4540, 2023.

[24] M. Dalal, A. Mandlekar, C. Garrett, A. Handa, R. Salakhutdinov, and D. Fox. Imitating task and motion planning with visuomotor transformers. 2023.

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

[26] K. Saha, V. Mandadi, J. Reddy, A. Srikanth, A. Agarwal, B. Sen, A. Singh, and M. Krishna. Edmp: Ensemble-of-costs-guided diffusion for motion planning. *arXiv preprint arXiv:2309.11414*, 2023.

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

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

[29] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[30] A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

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

[32] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[33] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[34] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.

[35] A. Agarwal, A. Kumar, J. Malik, and D. Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on robot learning*, pages 403–415. PMLR, 2023.

[36] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

[37] X. Wang, C. Yeshwanth, and M. Nießner. Sceneformer: Indoor scene generation with transformers. In *2021 International Conference on 3D Vision (3DV)*, pages 106–115. IEEE, 2021.

[38] A. Chang, W. Monroe, M. Savva, C. Potts, and C. D. Manning. Text to 3d scene generation with rich lexical grounding. *arXiv preprint arXiv:1505.06289*, 2015.

[39] A. Chang, M. Savva, and C. D. Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 2028–2038, 2014.

[40] D. Ritchie, K. Wang, and Y.-a. Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6182–6190, 2019.

[41] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi, and R. Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

[42] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.

[43] P. Katara, Z. Xian, and K. Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023.

[44] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[45] H. Wang, X. Du, J. Li, R. A. Yeh, and G. Shakhnarovich. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12619–12629, 2023.

[46] C. Chamzas, C. Quintero-Pena, Z. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki. Motionbenchmaker: A tool to generate and benchmark motion planning datasets. *IEEE Robotics and Automation Letters*, 7(2):882–889, 2021.

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

[48] A. H. Qureshi, J. Dong, A. Choe, and M. C. Yip. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.

[49] 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)*, pages 1916–1923. IEEE, 2023.

[50] B. Ichter, P. Sermanet, and C. Lynch. Broadly-exploring, local-policy trees for long-horizon task planning. *arXiv preprint arXiv:2010.06491*, 2020.

[51] S. Huang, Z. Wang, P. Li, B. Jia, T. Liu, Y. Zhu, W. Liang, and S.-C. Zhu. Diffusion-based generation, optimization, and planning in 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16750–16761, 2023.

[52] J. J. Johnson, L. Li, F. Liu, A. H. Qureshi, and M. C. Yip. Dynamically constrained motion planning networks for non-holonomic robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6937–6943. IEEE, 2020.

[53] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[54] A. Murali, A. Mousavian, C. Eppner, A. Fishman, and D. Fox. Cabinet: Scaling neural collision detection for object rearrangement with procedural scene generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1866–1874. IEEE, 2023.

[55] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox. Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017. IEEE, 2021.

Appendix

A Table of Contents

- **Related Work** (Appendix B): Other works that are related to Neural MP's approach.
- **Additional Real World Experiment Analysis** (Appendix C): Additional analysis of our real world results, with more detailed experiments.
- **Ablations** (Appendix D): Ablations and analyses of Neural MP, demonstrating the effectiveness of our design decisions.
- **Procedural Scene Generation Details** (Appendix E): Description of all the environments we use in this work.
- **Network Training Details** (Appendix F): Full details on how Neural MP is trained, specifically the hyper-parameters used for training and network architectures.
- **Real World Task Details** (Appendix G): Details regarding the real world setup and analysis.
- **Discussion and Limitations** (Appendix H): Brief summary of Neural MP and address its limitations.

B Related Work

Approaches for Training General-Purpose Robot Policies Prior work on large scale imitation learning using expert demonstrations [28, 29, 30, 24, 31, 32] has shown that large models trained on large datasets can demonstrate strong performance on challenging tasks and some varying levels of generalization. On the other hand, sim2real transfer of RL policies trained with procedural scene generation has demonstrated strong capabilities for producing generalist robot policies in the locomotion regime [33, 34, 35, 36]. In this work, we combine the strengths of these two approaches to produce powerful neural motion planning policies. We propose a method for procedural scene generation in simulation and combine it with large scale imitation learning to produce strong priors which we transfer directly to over 64 motion planning problems in the real world. **Procedural Scene Generation for robotics** Automatic scene generation and synthesis has been explored in vision and graphics [37, 38, 39, 40] while more recent work has focused on embodied AI and robotics settings [41, 42, 43, 24]. In particular, methods such as Robogen [42] and Gen2sim [43] use LLMs to propose tasks and build scenes using existing 3D model datasets [18] or text-to-3D [44, 45] and then decompose the tasks into components for RL, motion-planning and trajectory optimization to solve in simulation. Our method is instead rule-based rather than LLM-based, is designed specifically for generating data to train neural motion planners (see Sec. 2.1), and demonstrates that policies trained on its data can indeed be transferred to the real world. MotionBenchmaker [46], on the other hand, is similar to our data generation method in that it autonomously generates scenes using programmatic assets. However, the datasets generated by MotionBenchmaker are not realistic: floating robots, a single major obstacle per scene and primitive objects that are spaced far apart. By comparison, the scenes and data generated by our work (Fig. E.1) are considerably more diverse, containing additional programmatic assets that incorporate articulations (microwave, dishwasher), multiple large obstacles per scene (up to 5), complex meshes sampled from Objaverse [18], and tightly packed obstacles.

Neural Motion Planning Finally, there is a large body of recent work [47, 19, 48, 49, 50, 26, 51] focused on imitating motion planners in order to accelerate planning. MPNet [47, 52, 48] trains a network to imitate motion planners, then integrates this prior into a search procedure at test time. Our method leverages large scale scene generation and sequence modeling, enabling it to use a faster optimization process at test time while obtaining strong results across a diverse set of tasks. MπNets [19] trains the SOTA neural motion planning policy using procedural scene generation and demonstrates transfer to the real world. Our approach is similar, albeit with 1) much more diverse scenes via programmatic asset generation and complex real-world meshes, 2) a more powerful learning architecture and multi-modal output distributions and 3) test-time optimization to improve performance at deployment, enabling significantly improved performance over MπNets.

C Additional Real World Results and Analysis

Figure C.1: **Neural Motion Planning at Scale in the Real World** Our approach enables a *single*, *generalist* neural network policy to solve motion planning problems across diverse setups; Neural MP can generate collision free motions for a wide array of unseen tasks significantly faster and with higher success than traditional as well as learning-based motion planning approaches.

C.1 Detailed Free Hand Motion Planning Results

In this section we perform additional analysis of the free hand motion planning results from the main paper. We include a more detailed version of Tab. **??** in the Appendix shown below (Tab. C.2). In this table, we additionally include the average (open loop) planning time per method and the average rate of safety violations. Safety violations are defined to occur where there are collisions, the robot hits its joint limits or there are torque limit errors. The open loop planning time for neural methods such as ours or MπNets involves simply measuring the total time taken for rolling out the policy and test time optimization (TTO). We find that sampling-based planners in general never collide when executed. If they produce a safety violation, it is only because they find a trajectory that is infeasible for the robot to execute on the hardware, due to joint or torque limit errors. Neural motion planning methods have much higher collision rates, though Neural MP has a significantly lower collision rate than MπNets, which we attribute to test-time optimization pruning out bad trajectories. We also note that not all collisions are created equal: some are slight, lightly grazing the environment objects while still achieving the goal, while others can be catastrophic, colliding heavily into the environment. In general, we found that our method tends to produce trajectories that may have slight collisions, though most of these are pruned out by TTO. With regards to planning time, MπNets is the fastest method, as our method expends additional compute rolling out 100x more trajectories and then selecting the best one using SDF-based collision checking.

C.2 Detailed In-hand Motion Planning Results

In this section, we extend the in-hand results shown in the main paper (Fig. **??**) with additional baselines (AIT*-80s, AIT*-10s and MπNets). For this evaluation (see Tab. C.3, we consider two of the four in-hand motion planning objects, namely joystick and book. We find sampling-based methods are able to perform in-hand motion planning quite well, matching the performance of our base policy as well as our method without Objaverse data. We also see that MπNets is unable to

	Bins (↑)	Shelf (↑)	Articulated (↑)	Avg. Success Rate (↑)	Avg. Planning Time (↓)	Avg. Safety Viol. Rate (↓)
Sampling-based Planning:						
`AIT*-80s` [10]	93.75	75	50.0	72.92	80	**0**
`AIT*-10s (fast)` [10]	75.0	37.5	25.0	45.83	10	2.1
Neural:						
`M$\pi$Nets` [19]	18.75	25.0	6.25	16.67	**1.0**	18.75
`Ours`	**100**	**100**	**87.5**	**95.83**	3.9	4.2

Figure C.2: Neural MP performs best across tasks for free-hand motion planning, demonstrating greater improvement as the task complexity grows.

	Book (↑)	Joystick (↑)	Avg. Success Rate (↑)	Avg. Planning Time (↓)	Avg. Safety Viol. Rate (↓)
Sampling-based Planning:					
`AIT*-80s` [10]	50	50	50	80	**0**
`AIT*-10s (fast)` [10]	25	50	37.5	10	**0**
Neural:					
`M$\pi$Nets` [19]	0	0	0	1	37.5
Ours:					
`Ours` (no TTO)	25	**75**	50	**0.9**	50
`Ours` (no Objaverse)	50	50	50	3.9	50
`Ours`	**100**	**75**	**87.5**	3.9	12.5

Figure C.3: Neural MP performs best across tasks for in-hand motion planning, demonstrating greater improvement as the in-hand object becomes more challenging.

perform in-hand motion planning on any of the evaluated tasks. This is likely because that network was not trained on data with objects in-hand, demonstrating the importance of including in-hand data when training neural motion planners. Finally, there is a significant gap in performance between our method with and without test-time optimization; pruning out colliding trajectories at test time is crucial for achieving high success rates on motion planning tasks.

C.3 Comparisons to Learning-based Motion Planners

In this section we evaluate how Neural MP compares to two additional learning-based methods, MPNets [47] and EDMP [26] (a Diffusion-based neural motion planner) as well as MπNets [19] in simulation. We compare these three neural motion planning methods in simulation trained on the same dataset (from MπNets) of 3.27 million trajectories. We train policies on the Global expert data and the Hybrid datasets and then evaluate on 5400 test problems across the Global, Hybrid and Both solvable subsets. We include numerical results Tab. C.4, with numbers for the baselines taken from the EDMP and MπNets papers. We find that across the board, Neural MP is the best learning-based motion planning method. We attribute this to the use of sequence modelling, the ability of the GMM to fit multimodal data and test-time optimization to prune out any collisions.

	Global	Hybrid	Both	Average
`MPNet` [47]				
Hybrid Expert	41.33	65.28	67.67	58.09
`M$\pi$Nets` [19]				
Global Expert	75.06	80.39	82.78	79.41
Hybrid Expert	75.78	95.33	95.06	88.72
`EDMP` [26]				
Global Expert	71.67	82.84	82.79	79.10
Hybrid Expert	75.93	86.13	85.06	82.37
`Ours`				
Global Expert	**77.93**	85.50	87.67	83.70
Hybrid Expert	76.33	**97.28**	**96.78**	**90.13**

Figure C.4: Performance comparison of neural motion planning methods across 5400 test problems in the MπNets dataset in simulation. Neural MP achieves the state-of-the-art results on these tasks.

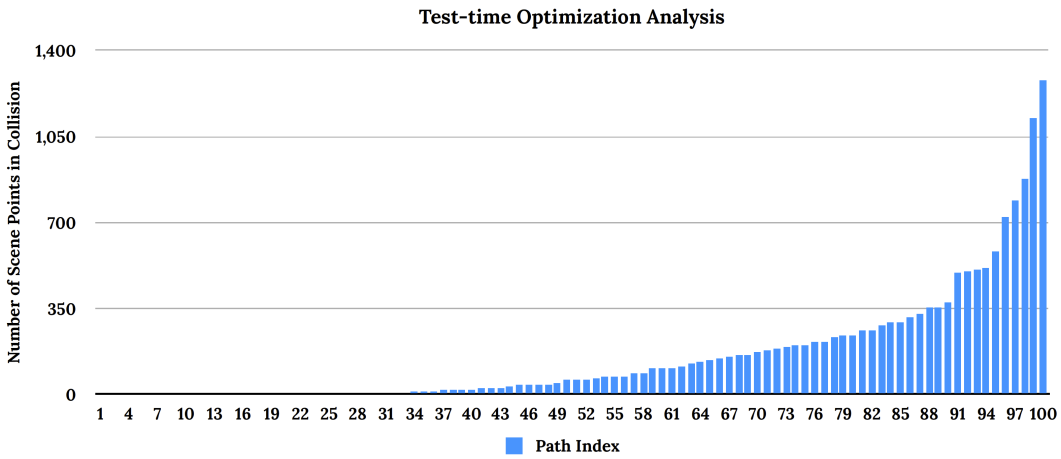


Figure C.5: **Test-time Optimization Analysis** For the Bins Scene 1 task, we plot the number of points in collision across 100 sampled trajectories from the model. 25% of the trajectories are completely collision free and we select a trajectory execute from that subset.

C.4 Test-time Optimization Analysis

To analyze what the test-time optimization procedure is doing, we first note that the base policy can sometimes produce slight collisions with the environment due to the imprecision of regression. As a result, when sampling from the policy, it is often likely that the policy will lightly graze objects which will count as failures when motion planning. We visualize a set of trajectories sampled from the policy here on our website for the real-world bins task. Observe that for some of the trajectories, the policy slightly intersects with the bin which would cause it to fail when executing in the real world, while for others it simply passes over the bin completely without colliding. We estimate the robot-scene intersection of all of these trajectories by comparing the robot SDF to the scene point-cloud and plot the range of values in Fig. C.5. We observe that 25% of trajectories do not collide with the environment, and we select for those. In principle, one could further optimize by selecting the trajectory that is furthest from the scene (using the SDF). In practice, we did not find this necessary and that selecting the first trajectory among those with the fewest expected collisions performed quite well in our experiments.

D Ablations

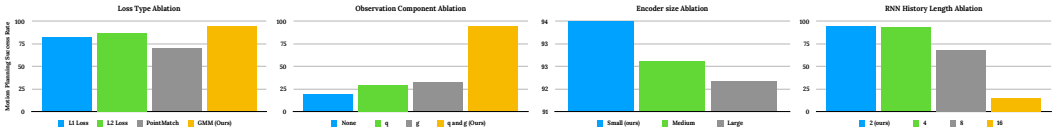


Figure D.1: **Ablation Results** We evaluate four different components of Neural MP, loss type (*left*), observation components (*middle left*), encoder sizes (*middle right*), and RNN history length (*right*). We validate that our design decisions produce measurable improvements in motion planning success rates.

We run additional ablations analyzing components of our method in simulation using a subset of our dataset (100K trajectories) and include additional details for experiments discussed in the main paper.

Loss Types For training objective, we evaluate 4 different options: GMM log likelihood (ours), MSE loss, L1 loss, and PointMatch loss (MπNets). PointMatch loss involves computing the l2 distance between the goal and the predicted end-effector pose using 1024 key-points. We plot the results on held out scenes in Fig. D.1. We find that GMM (ours) outperforms L2 loss, L1 loss, and PointMatch Loss (MπNets) by (7%, 12%, and 24%) respectively. One reason this may be the case is that sampling-based motion planners produce highly multi-modal trajectories: they can output entirely different trajectories for the same start and goal pair when sampled multiple times. Since Gaussian Mixture Models are generally more capable of capturing multi-modal distributions, they can hence fit our dataset well. At the same time, the PointMatch [19] loss struggles significantly on our data: it cannot distinguish between 0 and 180 degree flipped end-effector orientations, resulting in many failures due to incorrect end-effector orientations.

Observation Components We evaluate whether our choice of observation components impacts the Neural MP's performance. In theory, the network should be able to learn as well from the point-cloud alone as when the proprioception is included, as the point-cloud contains a densely sampled point-cloud of the current and goal robot configurations. However, in practice, we find that this is not the case. Instead, removing either q or g or both severely harms performance as seen in Fig. D.1. We hypothesize that including the proprioception provides a richer signal for the correct delta action to take.

RNN History Length In our experiments, we chose a history length of 2 for the RNN, after sweeping over values of 2, 4, 8, 16 based on performance. From Fig. D.1 we see history length 2 achieves the best performance at 94%, while using lengths 4, 8 and 16 achieve progressively decreasing success rates (92.67, 68, 14.67). One possible reason for this is that since point-clouds are already very dense representations that cover the scene quite well, the partial observability during training time is fairly low. A shorter history length also leads to faster training, due to smaller batches and fewer RNN unrolling steps.

Encoder Size Finally, we briefly evaluate whether encoder size is important when training large-scale neural motion planners. We train 3 different size models: small (4M params), medium (8M params) and large (16M params). From the results in Fig. D.1, we find that the encoder size does not affect performance by a significant margin (94%, 93%, 92%) respectively and that the smallest model in fact performs best. Based on these results, we opt to use the small, 4M param model in our experiments.

Neural MP-MLP	Neural MP-LSTM	Neural MP-Transformer	Neural MP-ACT
65.0	82.5	**85.0**	47.5

Table D.1: Ablation of different architecture choices for the action decoder. We find that LSTMs and Transformers comparably while LSTMs boast faster inference times.

Architecture Ablation In this experiment, we evaluate how different sequence modelling methods (Transformers and ACT [27]) and simpler action decoders such as MLPs compare against our design choice of using an LSTM. All methods are trained with the same dataset (of 1M trajectories), with the same encoder and GMM output distribution (with the exception of ACT which uses an L1 loss

as per the ACT paper). We then evaluate them on held out motion planning tasks (Fig. D.1 which are replicas of our real-world tasks (Bins and Shelf). We note several findings: 1) ACT performs poorly, largely due to its design choice of using an L1 loss which prevents it from handling planner multi-modality effectively, 2) Neural MP with an MLP action decoder also performs significantly worse than LSTMs and Transformers, as it is unable to use history information effectively to reason about the next action 3) Transformers and LSTMs perform similarly, with the Transformer variant performing marginally better, but with significantly slower inference time (2x). Hence we opt to use LSTM policies for our experimental evaluation, but certainly our method is amenable to any choice of sequence modeling architecture that performs well and has fast inference.

Neural MP-MotionBenchMaker	**Neural MP-MπNets**	**Neural MP**
0	32.5	82.5

Table D.2: Comparing different methods for generating datasets for motion planning. We find that policies trained on our data generalize best to held out scenes.

Dataset Ablation Finally, we evaluate the quality of different dataset generation approaches for producing generalist neural motion planners. We do so by training policies on three different datasets (Neural MP, MπNets [19], and MotionBenchMaker [46]) and evaluated on held out motion planning tasks in simulation. We train each model to convergence for 10K epochs and then execute trajectories on two held out tasks that mirror our real world tasks: RealBins and RealShelf. For fairness, we do not include any Objaverse meshes in these tasks, since MPiNets and MotionBenchMaker only have primitive objects. Still, we find that our dataset performs best by a wide margin (Tab. D.2). In general, we found that policies trained on MotionBenchMaker do not generalize well. As mentioned in the related works section, this dataset lacks the realism and diversity necessary to train policies that can generalize to held out motion planning scenes.

Data Scaling In order to understand the scaling of our method with data we evaluate how performance changes with dataset size. To do so, we train models with 1K trajectories, 10K trajectories and 100K trajectories. In these experiments, we train with subsets of our overall dataset and evaluate on held out simulation environments which are not sampled from the training distribution. While performance with a thousand trajectories is weak (15%), we find rapid improvement as we increase the orders of magnitude of data (10K - 50%, 100K - 65%), with the model trained on 1M trajectories achieving 80% success rate on entirely held out tight-space shelf and bin configurations, demonstrating that our method scales and improves with data.

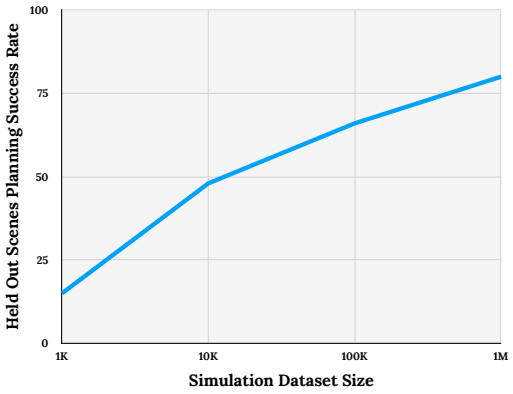


Figure D.2: Performance of our model on held out scenes, poses and configurations as we increase the dataset size.

E Procedural Scene Generation Details

In this section we provide additional details regarding the data generation methods we develop for training large scale neural motion planners.

Figure E.1: **Visualization of Diverse Simulation Training Environments**: We train Neural MP on a wide array of motion planning problems generated in simulation, with significant pose, procedural asset, and mesh configuration randomization to enable generalization.

E.1 Procedural Scene Generation

We formalize our procedural scene generation as a composition of randomly generated parameteric assets and sampled Objaverse meshes in Alg. 1

Procedural assets generation details We have six parametrically variable asset categories - shelves, cubbies, microwaves, dishwashers, open boxes, and cabinets. Each category instance g is comprised of N cuboids $g = \{x_0..x_i...x_N\}$, which satisfy the category level constraint given by $C(g)$. For controlled variation within each category, we make use of *parametric* category specific generation functions $X(\mathbf{p}) = \{x_0..x_i.x_N\}$, s.t. $C(X(\mathbf{p}))$, where p specifies the size and scale of each of the cuboids, their relative positions, and specific axes of articulation. The constraint C(.) relates to the relative positions, scales and orientations of the different cuboids, *e.g* for the microwave category the constraint ensures each of the walls are of the same height, and that the microwave has a hinge door.

Algorithm 1 Complex scene generation

Require: Asset category generators $\{X_i(\mathbf{p})\}_{0,1..G}$
Require: Number of scenes N
Require: Max objects per scene K
Require: Collision checker Q

```
for scene 1: N do
    Initialize scene S = {}
    Sample number of assets k ~ [1, ...K]
    for asset 1:k do
        Sample asset category g ~ [0, ..N]
        Sample asset parameter p
        Sample asset x ~ X_g(p)
        while Q(S, x) do
            for each asset s_i in S do
                n_i = collision normal b/n x and s_i
            end for
            Effective collision normal n = Σ n_i
            Update p so X_g(p) center is shifted along n
        end while
        Add asset x to scene S
    end for
    yield scene S
end for
```

Objaverse sampling details The Objaverse are sampled in the task-relevant sampling location of the programmatic asset(s) in the scene, such as between shelf rungs, inside cubbies or within cabinets. Similar to the programmatic assets, these Objaverse assets are also sampled from a category generator $X_{obj}(\mathbf{p})$. Here the parameter p specifies the size, position, orientation of the object as well as task-relevant sampling location of the object in the scene, such as between shelf rungs, inside cubbies or within cabinets. As discussed in the main paper, we propose an approach that iteratively adds assets to a scene by adjusting their position using the effective collision normal vector, computed from the existing assets in the scene. We detail the steps for doing this in Alg. 1.

E.2 Motion Planner Experts

We use three techniques to improve the data generation throughput when imitating motion planners at scale.

Hindsight Relabeling Tight-space to tight-space problems are the most challenging, particularly for sampling-based planners, often requiring significant planning time (up to 120 seconds) for the planner to find a solution. For some problems, the expert planner is unable to find an exact solution and instead produces approximate solutions. Instead of discarding these, note that we use a goal-conditioned imitation learning framework, where we can simply execute the trajectories in simulation and relabel the observed final state as the new goal.

Reversibility We further improve our data generation throughput by observing that since motion planners inherently produce collision-free paths, the process is reversible, at least in simulation. This allows us to double our data throughput by reversing expert trajectories and re-calculating delta actions accordingly. Additionally, for a neural motion planner to be useful for practical manipulation tasks, it must be able to generate collision free plans for the robot even when it is holding objects. To enable such functionality, we augment our data generation process with trajectories where objects are spawned between the grippers of the robot end effector. There are transformed along with the end-effector during planning in simulation. We consider the object as part of the robot for collision checking and for the sake of our visual observations. In order to handle diverse objects that the robot might have to move with at inference time, we perform significant randomization of the in-hand object that we spawn in simulation. Specifically, we sample this object from the primitive categories of boxes, cylinders or spheres, or even from Objaverse meshes of everyday articles. We randomize the scale of the object between 3 and 30 cm along the longest dimension, and sample random starting locations within a 5cm cube around the end-effector mid-point between grippers.

Smoothing Importantly, we found that naively imitating the output of the planner performs poorly in practice as the planner output is not well suited for learning. Specifically, plans produced by AIT* often result in way-points that are far apart, creating large action jumps and sparse data coverage, making it difficult to for networks to fit the data. To address this issue, we perform smoothing using cubic spline interpolation while enforcing velocity and acceleration limits. The implementation from MπNets performs well in practice, smoothing to a fixed 50 timesteps with a max spacing of 0.1 radians. In general, we found that smoothing is crucial for learning performance as it ensures action limits for each time-step transition.

E.3 Data Pipeline Parameters and Compute

In Table E.1, we provide a detailed list of all the parameters used in generating the data to train our model.

Hyper-parameter	**Value**
General Motion Planning Parameters	
collision checking distance	1cm
tight space configuration ratio	50%
dataset size	1M trajectories
minimum motion planning time	20s
maximum motion planning time	80s
General Obstacle Parameters	
in hand object ratio	0.5
in hand object size range	[[0.03, 0.03, 0.03], [0.3, 0.3, 0.3]]
in hand object xyz range	[[-0.05, -0.05, 0.], [0.05, 0.05, 0.05]]
min obstacle size	0.1
max obstacle size	0.3
table dim ranges	[[0.6, 1], [1.0, 1.5], [0.05, 0.15]]
table height range	[-0.3, 0.3]
num shelves range	[0, 3]
num open boxes range	[0, 3]
num cubbys range	[0, 1]
num microwaves range	[0, 3]
num dishwashers range	[0, 3]
num cabinets range	[0, 3]

Hyper-parameter	Value
Objaverse Mesh Parameters	
scale range	[0.2, 0.4]
x pos range	[0.2, 0.4]
y pos range	[-0.4, 0.4]
number of mesh objects per programmatic asset	[0, 3]
number of mesh objects on the table	[0, 5]
Table Parameters	
width range	[0.8, 1.2]
depth range	[0.4, 0.6]
height range	[0.35, 0.5]
thickness range	[0.03, 0.07]
leg thickness range	[0.03, 0.07]
leg margin range	[0.05, 0.15]
position range	[[0, 0.8], [-0.6, 0.6]]
z axis rotation range	[0, 3.14]
Shelf Parameters	
width range	[0.5, 1]
depth range	[0.2, 0.5]
height range	[0.5, 1.2]
num boards range	[3, 5]
board thickness range	[0.02, 0.05]
backboard thickness range	[0.0, 0.05]
num vertical boards range	[0, 3]
num side columns range	[0, 4]
column thickness range	[0.02, 0.05]
position range	[[0, 0.8], [-0.6, 0.6]]
z axis rotation range	[-1.57, 0]
Open Box Parameters	
width range	[0.2, 0.7]
depth range	[0.2, 0.7]
height range	[0.3, 0.5]
thickness range	[0.02, 0.06]
front scale range	[0.6, 1]
position range	[[0.0, 0.8], [-0.6, 0.6]]
z axis rotation range	[-1.57, 0.0]
Cubby Parameters	
cubby left range	[0.4, 0.1]
cubby right range	[-0.4, 0.1]
cubby top range	[0.85, 0.35]
cubby bottom range	[0.0, 0.1]
cubby front range	[0.8, 0.1]
cubby width range	[0.35, 0.2]
cubby horizontal middle board z axis shift range	[0.45, 0.1]
cubby vertical middle board y axis shift range	[0.0, 0.1]
board thickness range	[0.02, 0.01]
external rotation range	[0, 1.57]
internal rotation range	[0, 0.5]
num shelves range	[3, 5]
Microwave Parameters	
width range	[0.3, 0.6]
depth range	[0.3, 0.6]
height range	[0.3, 0.6]
thickness range	[0.01, 0.02]
display panel width range	[0.05, 0.15]
distance range	[0.5, 0.8]
external z axis rotation range	[-2.36, -0.79]
internal z axis rotation range	[-0.15, 0.15]

Hyper-parameter	Value
Dishwasher Parameters	
width range	[0.4, 0.6]
depth range	[0.3, 0.4]
height range	[0.5, 0.7]
control panel height range	[0.1, 0.2]
foot panel height range	[0.1, 0.2]
wall thickness range	[0.01, 0.02]
opening angle range	[0.5, 1.57]
distance range	[0.6, 1.0]
external z axis rotation range	[-2.36, -0.79]
internal z axis rotation range	[-0.15, 0.15]
Cabinet Parameters	
width range	[0.5, 0.8]
depth range	[0.25, 0.4]
height range	[0.6, 1.0]
wall thickness range	[0.01, 0.02]
left opening angle range	[0.7, 1.57]
right opening angle range	[0.7, 1.57]
distance range	[0.6, 1.0]
external z axis rotation range	[-2.36, -0.79]
internal z axis rotation range	[-0.15, 0.15]

Table E.1: **Data Generation Hyper-parameters** We provide a detailed list of hyper-parameters used to procedurally generate a vast variety of scenes in simulation.

Compute In order to collect a vast data of motion planning trajectories, we parallelize data collection across a cluster of 2K CPUs. It takes approximately 3.5 days to collect 1M trajectories.

F Network Training Details

We first describe additional details regarding our neural policy, and then discuss how it is trained. Following the design decisions of MπNets [19], we construct a segmented point-cloud for the robot, consisting of the robot point-cloud, the target goal robot point-cloud and the obstacle point-cloud. Here we note two key differences from MπNets: 1) our network conditioned on the target joint angles, while MπNets only does so through the segmented point-cloud, 2) we condition on the target joint angles, not end-effector pose, decisions that we found improved adherence to the overall target configuration. For in-hand motion planning, we extend this representation by considering the object in-hand as part of the robot for the purpose of segmentation.

We include a hyper-parameter list for our neural motion planner in Table F.1. We train a 20M parameter neural network across our dataset of 1M trajectories. The PointNet++ encoder is 4M parameters and outputs an embedding of dimension 1024. We concatenate this embedding with the encoded q_t and g vectors and pass this into the 16M parameter LSTM decoder. The decoder outputs weights, means, and standard deviations of the 5 GMM modes. We then train the model with negative log likelihood loss for 4.5M gradient steps, which takes 2 days on a 4090 GPU with batch size of 16.

Hyper-parameter	**Value**

PointNet++ Architecture	`PointnetSAModule(`
	`  npoint=128,`
	`  radius=0.05,`
	`  nsample=64,`
	`  mlp=[1, 64, 64, 64],`
	`)`
	`PointnetSAModule(`
	`  npoint=64,`
	`  radius=0.3,`
	`  nsample=64,`
	`  mlp=[64, 128, 128, 256],`
	`)`
	`PointnetSAModule(`
	`  nsample=64,`
	`  mlp=[256, 512, 512],`
	`)`
	`MLP(`
	`  Linear(512, 2048),`
	`  GroupNorm(16, 2048),`
	`  LeakyReLU,`
	`  Linear(2048, 1024),`
	`  GroupNorm(16, 1024),`
	`  LeakyReLU,`
	`  Linear(1024, 1024)`
	`)`

LSTM	1024 hidden dim, 2 layers
Inputs	q_t, g, PCD_t
Batch Size	16
Learning Rate	0.0001
GMM	5 modes
Sequence Length (seq length)	2
Point Cloud Parameters	
Number of Robot / Goal Point-cloud Points	2048
Number of Obstacle Point-cloud Points	4096

Table F.1: Hyper-parameters for the model

G Real World Setup Details

In this section, we describe our real world robot setup and tasks in detail and perform analysis on the perception used for operating our policies.

G.1 Real Robot Setup

Hardware For all of our experiments, we use a Franka Emika Panda Robot, which is a 7 degree of freedom manipulator arm. We control the robot using the manimo library (https://github.com/AGI-Labs/manimo) and perform all of experiments using their joint position controller with the default PD gains. The robot is mounted to a fixed base pedestal behind a desk of size .762m by 1.22m with variable height. For sensing, we use four extrinsically calibrated depth cameras, Intel Realsense 435 / 435i, placed around the scene in order to accurately capture the environment. We project the depth maps from each camera into 3D and combine the individual point-clouds into a single scene representation. We then post-process the point-cloud by cropping it to the workspace, filtering outliers and denoising, and sub-sampling a set of 4096 points. This processed point-cloud is then used as input to the policy.

Algorithm 2 Open-Loop Execution of Neural MP

Input: Neural MP π_θ, segmentor $\mathcal{S}$, initial angles q_0, scene point-cloud PCD_{full}, goal g, horizon H
Output: Executed trajectory on the robot
Initialize: Timestep $t \leftarrow 0$
Initialize: Trajectory $\tau \leftarrow \{\}$
$PCD_0 \leftarrow \mathcal{S}(PCD_{full}) \cup PCD_{q_0} \cup PCD_g$
while goal g not reached **and** $t < H$ **do**
  $a_t \sim \pi_\theta(q_{t-1}, PCD_{t-1}, g)$
  $q_t \leftarrow q_{t-1} + a_t$
  $PCD_t \leftarrow (PCD_{t_1} \setminus PCD_{q_{t-1}}) \cup PCD_{q_t}$
  $t \leftarrow t + 1$
  $\tau \leftarrow \tau + a_t$
end while
Execute the τ open loop on the robot.

Representation Collision Checking and Segmentation In order to perform real world collision checking and robot point-cloud segmentation, we require a representation of the robot to check intersections with the scene (collision checking) and to filter out robot points from the scene point-cloud (segmentation). While the robot mesh is the ideal candidate for these operations, it is far too slow to run in real time. Instead, we approximate the robot mesh as spheres (visualized in Fig. G.1) as we found this performs well in practice while operating an order of magnitude faster. We use 56 spheres in total to approximate the links of the robot as well as the end-effector and gripper. These have radii ranging from 2cm to 10cm and are defined relative to the center of mass of the link. This representation is a conservative one: it encapsulates the robot mesh, which is desirable for segmentation as this helps account for sensing errors which would place robot points outside of the robot mesh.

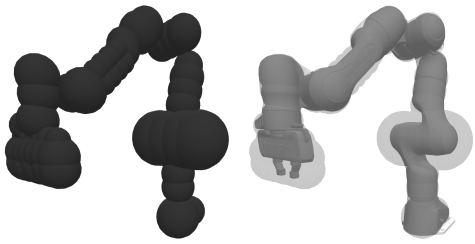


Figure G.1: We visualize the spherical representation on the left and overlay it on the robot mesh on the right.

Robot Segmentation In order to perform robot segmentation in the real world, we use the spherical representation to filter out robot points in the scene, so only the obstacle point-cloud remains. Doing so requires computing the Signed Distance Function (SDF) of the robot representation and then checking the scene point-cloud against it, removing points from the point-cloud in which SDF value is less than threshold ϵ. For the spherical representation, the SDF computation is efficient: for a sphere with center C and radius r, the SDF of point x is simply $||x - C||_2 - r$. In our experiments, we use a threshold ϵ of 1cm. We then replace the removed points with points sampled from the robot mesh of the robot. This is done by pre-sampling a robot point-cloud from the robot mesh at the default configuration, then performing forward kinematics using the current joint angles q_t and transforming the robot point-cloud accordingly. Replacing the real robot point-cloud with this sampled point-cloud ensures that the only difference between sim and real is the obstacle point-cloud.

Real-world Collision Checking Given the SDF, collision checking is also straightforward, we denote the robot in collision if any point in the scene point-cloud (this is after robot segmentation) has SDF value less than 1cm. Note this means that first state is by definition collision free. Also, this technique will not hold if performing closed loop planning, in that case this method would always denote the state as collision free as the points with SDF value less than 1cm would be segmented out for each intermediate point-cloud.

Open Loop Deployment For open-loop execution of neural motion planners, we execute the following steps: 1) generate the segmented point-cloud at the first frame, 2) predict the next trajectory way-point by computing a forward pass through the network and sampling an action, 3) update the current robot point-cloud with mesh-sampled point-cloud at the predicted way-point, and 4) repeat until goal reaching success or maximum rollout length is reached. The entire trajectory is then executed on the robot after the rollout. Please see Alg. 2 for a more detailed description of our open-loop deployment method.

G.2 Experimental Setup

In our experiments, we consider motion planning in four different real world environments containing obstacles (see Appendix G.3). Importantly, these are not included as part of the training set, and thus the policy needs to generalize to perform well on these settings. We begin by describing our environment design, then each of the environments, and finally our evaluation protocol and comparisons.

(a) Sampling-based planners struggle with tight spaces, a regime in which Neural MP performs well.

(b) Our method is able to motion plan with objects in-hand, a crucial skill for manipulation.

(c) Our policy has not been trained on this bookcase, yet it is able to insert the book into the correct location.

Figure G.2: Emergent Capabilities of Neural MP

Environment Design We evaluate our motion planner on tabletop motion planning tasks which we subdivide into *environments*, *scenes*, and *configurations*. We evaluate on four different environments, with each environment containing 1-2 large receptacles that function as the primary obstacles. For each environment, we have four different scenes which involve significant pose variation (over the entire tabletop) of the primary obstacles, table height randomization, as well as randomized selection, pose and orientation of objects contained within the receptacles. For each environment, we have two scenes with obstacles and two without obstacles. For each scene, we evaluate on four different types of start (q_0) and goal (g) angle pairs: 1) free space to free space, 2) free space to tight space 3) tight space to free space 4) tight space to tight space. Free space configurations do not have an obstacle in the vicinity of the end-effector, while tight space configurations generally have obstacles on most sides of the end-effector. Our four environments are 1) **Bins**: moving in-between, around and inside two different industrial bins 2) **Shelf**: moving in-between and around the rungs of a black shelf 3) **Articulated**: moving inside and within cubbies, drawers and doors 4) **in-hand**: moving between rungs of a shelf while holding different objects.

Evaluation Protocol We evaluate all methods on open loop planning performance for fairness, though our method, is capable of planning in a closed loop manner. For neural planners such as our method and MπNets, this involves generating an open loop path by passing the agent's predictions back into itself using a linear model for the next state. We then execute the plans on the robot, recording the goal reaching success rate, collision rate and the time taken to reach the goal. We follow MπNets' definition of success rate: reaching within 1cm and 15 degrees of the goal end-effector pose

of the target goal configuration while also not colliding with anything in the scene. In practice, our policy achieves orientation errors significantly below this threshold, 2 degrees or less.

Comparisons We propose three baselines for real-world comparisons to evaluate different aspects of our method's capabilities. We compare against sampling-based motion planning, which is expensive to run but has strong guarantees on performance. The first baseline is the expert we use to train our model, AIT* with 80 seconds of planning time. We run this planner with the same real world collision checker used by our method. AIT*-80s is impractical to deploy in most settings due to its significant planning time. Thus, we compare to a faster variant of AIT* with 10 seconds of planning time, which uses comparable time to our method (Note: AIT*-3s is unable to find a plan for any real world task). Next, we compare against Curobo [17], a SOTA motion-generation method which performs GPU-parallelized optimization and is much faster than AIT*. We run this baseline with a voxel-based collision checker and optimize its voxel resolution per task due to its sensitivity to that parameter. Finally, we compare against the SOTA neural motion planning approach, MπNets.

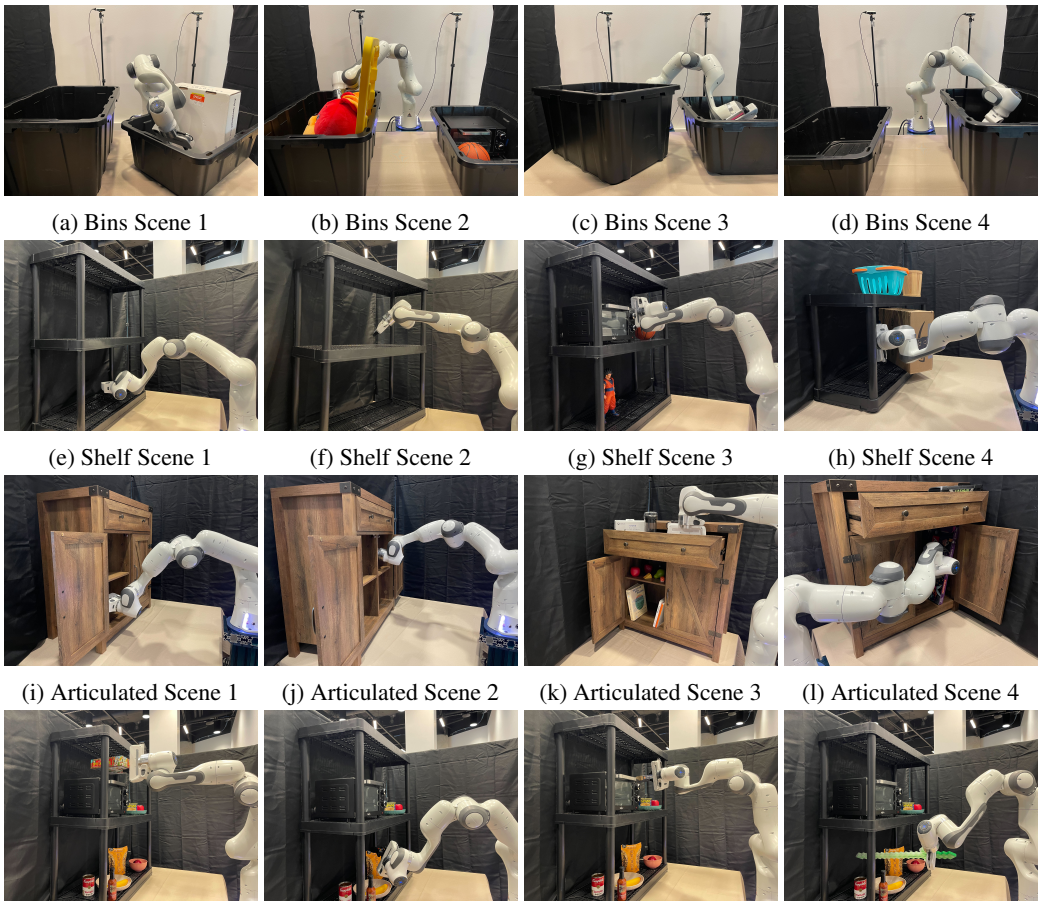
G.3 Tasks

Bins This task requires the neural planner to perform collision avoidance when moving in-between, around and inside two different industrial bins pictured in the first row of Fig. G.3. We randomize the position and orientation of the bins over the table and include the following objects as additional obstacles for the robot to avoid: toaster, doll, basketball, bin cap, and white box. The small bin is of size 70cm x 50cm x 25cm. The larger bin is of size 70cm x 50cm x 37cm. The bins are placed at two sides of the table. Between tasks, we randomize the orientation of the bins between 0 and 45 degrees and we swap the bin ordering (which bin is on the left vs. the right). The bins are placed 45cm in front of the robot, and shifted 60cm left/right.

Shelf This task tests the agent's ability to handle horizontal obstacles (the rungs of the shelf) while maneuvering in tighter spaces (row two in Fig. G.3). We randomize the size of the shelf (by changing the number of layers in the shelf from 3 to 2) as well as the position and orientation (anywhere at least .8m away from the robot) with 0 or 30 degrees orientation. The obstacles for this task include the toaster, basketball, baskets, an amazon box and an action figure which increase the difficulty. The shelf obstacle itself is of size 35cm x 80cm x 95cm.

Articulated We extend our evaluation to a more complex primary obstacle, the cabinet, which contains one drawer and two doors and tight internal spaces with small cubby holes (row three of Fig. G.3). We randomize the position of the entire cabinet over the table, the joint positions of the drawer and doors and the sizes of the cubby holes. The obstacles for this task are xbox controller box, gpu, action figure, food toy, books and board game box. The size of the cabinet is 40cm x 75cm x 80cm. The size of the top drawer is 30cm x 65cm x 12cm. The size of the cubbies is 35cm x 35cm x 25cm. The drawer has an opening range of 0-30cm and the doors open between 0 and 180 degrees.

In-Hand Motion Planning In this task (shown in row four of Fig. G.3), the planner needs to reason about collisions with not only the robot and the environment, but the held object too. We initialize the robot with an object grasped in-hand and run motion planning to reach a target configuration. For this task, we fix the obstacle (shelf) and its position (directly 80cm in front of the robot), instead randomizing across in-hand objects and configurations. We select four objects that vary significantly in size and shape: Xbox controller (18cm x 15cm x 8cm), book (17cm x 23cm x 5cm), toy sword (65cm x 10cm x 2cm), and board game (25cm x 25cm x 6cm). For this evaluation, we assume the object is already grasped by the robot, and the robot must just move with the object in-hand while maintaining its grasp.



(a) Bins Scene 1 (b) Bins Scene 2 (c) Bins Scene 3 (d) Bins Scene 4

(e) Shelf Scene 1 (f) Shelf Scene 2 (g) Shelf Scene 3 (h) Shelf Scene 4

(i) Articulated Scene 1 (j) Articulated Scene 2 (k) Articulated Scene 3 (l) Articulated Scene 4

(m) In Hand Object 1 (n) In Hand Object 2 (o) In Hand Object 3 (p) In Hand Object 4

Figure G.3: Images of our 16 evaluation scenes.

G.4 Perception Visualization and Analysis

We compare point-clouds from simulation and the real world for the Bins and Shelf task and analyze their properties. We replicate Bins Scene 4 and Shelf Scene 1 in simulation: simply measure the dimensions and positions of the real world objects and set those dimensions in simulation using the OpenBox and Shelf procedural assets. As seen in Fig. G.4, simulated point-clouds are far cleaner than those in the real world, which are noisy and perhaps more importantly, partial. The real-world point-clouds often have portions missing due to camera coverage as for large objects it is challenging to cover the scene well while remaining within the depth camera operating range. However, we find that our policy is still able to able operate well in these scenes, as PointNet++ is capable of handling partial point-clouds and is trained on a diverse dataset containing many variations of boxes and shelves with different types and number of components as well as sizes, which may enable the policy to generalize to partial boxes and shelves observed in the real world.

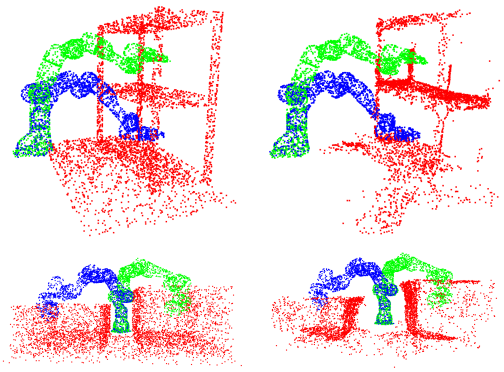


Figure G.4: **Visualization of Sim and Real point-clouds**: We visualize point-clouds of the Bins and Shelf task in sim and real, in the same poses. Due to noise in depth sensing, the real world point-clouds have significantly more deformations, yet our policy generalizes well to these tasks.

H Discussion and Limitations

In this work, we present Neural MP, a method that builds a data-driven policy for motion planning by scaling procedural scene generation, distilling sampling-based motion planning and improving at test-time via refinement. Our model demonstrably improves over the sampling-based planning in the real world, operating 2.5x-20x faster than AIT* while improving by over 20% in terms of motion planning success rate. Notably, our model generalizes to a wide distribution of task instances and demonstrates favorable scaling properties. At the same time, there is significant room for future work to improve upon, our model 1) is susceptible to point-cloud quality, which may require improving 3D representations via implicit models such as NeRFs [53], 2) does not still handle tight spaces well, a capability which could be potentially acquired via RL fine-tuning of the base policy and 3) is slower than simply running the policy directly due to test-time optimization, which can be addressed by leveraging learned collision checking [54, 55].