# Adaptive Experience Sampling for Motion Planning using the Generator-Critic Framework

Yiyuan Lee, Constantinos Chamzas, and Lydia E. Kavraki

***Abstract*—Sampling-based motion planners are widely used for motion planning with high-DOF robots. These planners generally rely on a uniform distribution to explore the search space. Recent work has explored learning biased sampling distributions to improve the time efficiency of these planners. However, learning such distributions is challenging, since there is no direct connection between the choice of distributions and the performance of the downstream planner. To alleviate this challenge, this paper proposes APES, a framework that learns sampling distributions optimized *directly* for the planner's performance. This is done using a *critic*, which serves as a differentiable surrogate objective modeling the planner's performance – thus allowing gradients to circumvent the non-differentiable planner. Leveraging the differentiability of the critic, we train a *generator*, which outputs sampling distributions optimized for the given problem instance. We evaluate APES on a series of realistic and challenging high-DOF manipulation problems in simulation. Our experimental results demonstrate that APES can learn high-quality distributions that improve planning performance more than other biased sampling baselines.**

***Index Terms*—Motion and Path Planning; Learning from Experience**

## I. Introduction

*Motion planning* [1] is a core component of reasoning in robotics. Growing fields such as Task and Motion Planning [2] and collaborative robotics [3] demand the need for more performant motion planning. Here, motion planning is used as a time-critical subroutine – called repeatedly by other high-level modules. Sampling-based motion planners [4], [5] are widely used today [6], and perform relatively well for practical systems. These planners have been shown to efficiently approximate the connectivity of high-dimensional search spaces using a small number of configuration samples [4]. Such planners typically explore the search space by drawing samples from a uniform distribution. However, they still face difficulties in certain high-dimensional problems [7]. To improve their efficiency, manually defined criteria can be used to construct a biased sampling distribution [8]–[10]. However, identifying such criteria may be challenging for certain problems. On the other hand, recent approaches have explored using experience to learn high-quality sampling distributions [11]–[13].

Manuscript received: February, 24, 2022; Revised May, 21, 2022; Accepted June, 26, 2022.

This paper was recommended for publication by Editor Hanna Kurniawati upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF 2008720, NSF-GRFP 1842494, and Rice University Funds.

YL, CC and LEK are with the Department of Computer Science, Rice University, Houston, TX, USA {yiyuan.lee,chamzas,kavraki}@rice.edu

Digital Object Identifier (DOI): see top of this page.

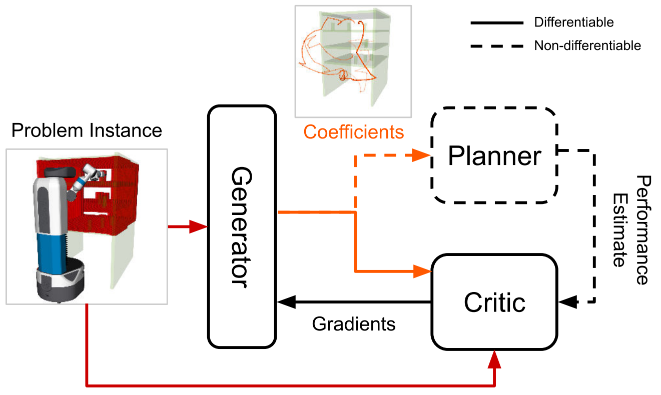


Figure 1: Overview of the training procedure for APES. The problem instance is represented as an occupancy grid, a start configuration, and a goal configuration. Given a problem instance, a *generator* proposes a set of coefficients to instantiate a sampling distribution (orange). This sampling distribution is represented as a basis of paths, weighted by the coefficients (trajectories shown in the top-most subfigure). The sampling distribution is used to bias a given planner's search. After planning, we retrieve a performance estimate, which is used to update a *critic* via supervision loss. The critic acts as a differentiable surrogate objective (solid arrows) for the generator. This allows the training gradients to circumvent the non-differentiable (dashed arrows) planner, optimizing the generator *directly* for downstream planning performance.

Learning such sampling distributions is challenging because there is no direct connection between the choice of a sampling distribution and the planner's performance. As such, existing methods seek to optimize the sampling distributions against other indirect objectives, such as to reconstruct bottlenecks in offline roadmaps [14], and past solution paths [15]. However, these indirect learning objectives may not accurately reflect the core objective of maximizing the planner's performance. To the best of our knowledge, no method exists that optimizes the distributions *directly* for the performance of non-differentiable sampling-based planners.

Towards this goal, we present **A**da**p**tive **E**xperience **S**ampling (APES), a framework that learns sampling distributions optimized directly for a given planner's performance. Using APES (Fig. 1), the planner uses the learned distributions to sample along a *basis* of paths – where the probability of producing a sample along each path is described by a set of *coefficients*. The core of APES is a *generator*, which maps a given problem instance into a set of high-quality coefficients. The generator is trained via the recent *generator-critic* framework [16], where a *critic* is built from past experience to model the planner's performance. The critic is used as a differentiable surrogate objective to circumvent the non-differentiable planner. This allows the generator to be optimized *directly* for planning performance via gradient-based methods.

We applied APES to planning with RRTConnect [17] for a series of realistic and challenging high-DOF manipulation problems in simulation. In our experiments, we compare APES with baselines that learn biased sampling distributions using heuristics. We show that APES discovers better sampling distributions – by considering the true performance objective (i.e., minimizing the number of planner iterations required). Additionally, we also investigate which components of the problem specification are most important for learning high-quality distributions, via an ablation study over the generator's inputs.

Overall, the main contributions of this work are as follows. 1) We present APES, a framework that can optimize sampling distributions *directly* for planning performance. 2) We show that the representation of sampling distributions as a weighted path basis is effective for learning. 3) Experimentally validate the effectiveness of APES on realistic, high-DOF motion planning problems. 4) We investigate, through an ablation study, the importance of available problem information for performance.

## II. Related Work

Over the years, several methods have been proposed to improve planning performance of sampling-based planners, through biased sampling. Some approaches utilize manually defined criteria, such as Gaussian Sampling [8], Bridge Sampling [9], and Medial-axis Sampling [10], where a configuration sample is only accepted if it passes a user-defined check. Another line of work utilizes workspace decompositions – such as the Ball Decomposition framework of [18] or the Delaunay Decompositions employed by [19]. More recently, [20] introduced different section patterns to effectively explore narrow passages for high-DOF robots.

Instead of handcrafted criteria, many methods have proposed learning the sampling distributions from a robot's past experience. One such class of methods attempt to learn fixed sampling distributions that exploit problem invariants. For example, [21] uses Kernel Density Estimation and [22] a Gaussian Mixture Model (GMM) to model such distributions based on past solution paths. Although easy to implement, it is difficult for such methods to generalize across variations in workspaces and start and goal configurations, since the learned sampling distributions are not adaptive – they are unable to exploit problem-specific information to maximize performance.

Instead of fixing the distributions, some methods exploit start and goal specifications to use past experience for planning. For example, library-based methods store experience in the form of paths [23] or roadmaps [24], which are later retrieved with handcrafted similarity functions defined over the start and goal of new problems. The retrieved paths, if invalid, can be used to seed optimization-based planners [25], or can be reused in tandem with sampling-based planners [23], [24], [26]. In contrast to our method, the retrieval of these paths are not optimized directly for performance, but rather based on similarity to past experience.

Another class of methods utilizes only workspace features to infer good sampling distributions. For example, [27], [28] learn important workspace regions which are subsequently transformed into $\mathcal{C}$-space configuration samples through inverse kinematics. The works of [12], [29] construct databases of local samplers, which are queried by handcrafted workspace similarity functions to synthesize sampling distributions. However, these methods ignore the start and goal information, and either apply only to low-dimensional robots, or require handcrafting features of the workspace.

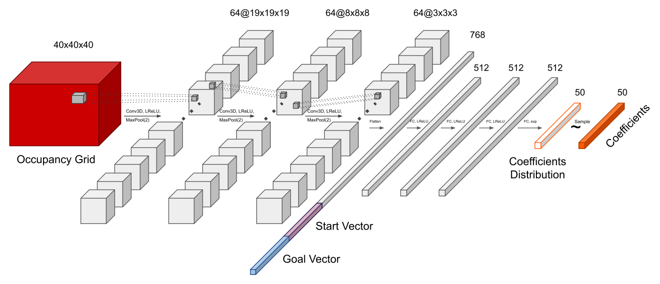
Finally, some methods generate sampling distributions that are conditioned on all available information, i.e., workspace, start, and goal – by leveraging the representational capabilities of deep neural networks. The work of [15] uses a conditional variational autoencoder (CVAE) to reconstruct past solution paths as a form of sampling distribution. This was adapted in [11], [14], where the reconstruction of the CVAE is used to identify critical $\mathcal{C}$-space regions via graph-based methods. In contrast, APES does not use such reconstruction heuristics, but learns a sampling distribution by optimizing *directly* for downstream planning performance. Similar to our method, [30] predicts weights of a GMM but defines a similarity cost over depth images based on reconstruction. These existing methods optimize against a handcrafted heuristic objective which does not necessarily correspond to the true performance metric (of minimizing the planner iterations required). Instead, APES can optimize against the chosen performance objective directly, by making use of the differentiable critic.

## III. Problem Definition

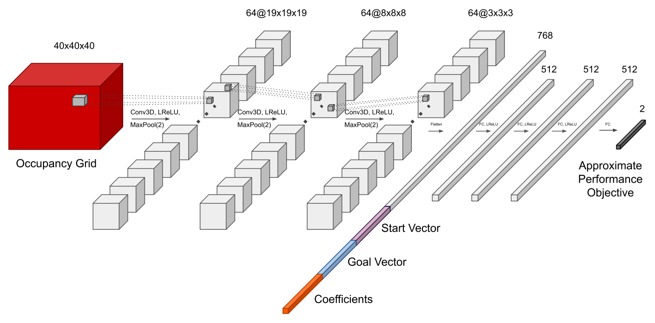
We consider a motion planning *problem*, which includes: 1) the geometry and kinematics of the robot we wish to plan for, 2) the set of obstacles and the distribution of possible placements for each obstacle, 3) the distribution of possible start configurations for the robot, and 4) the distribution of possible goal configurations for the robot. Each *problem instance* $c$ defines a specific placement for each obstacle, a specific choice of start configuration, and a specific choice of goal configuration. In particular, such a problem instance $c$ is represented as a triplet of: 1) a 3D *occupancy grid* of the workspace, 2) a real vector corresponding to the start configuration in $\mathcal{C}$-space, and 3) a real vector corresponding to the goal configuration in $\mathcal{C}$-space.

For every problem instance $c$, we seek to produce a sampling distribution, represented as a Gaussian Mixture Model (GMM) in the $\mathcal{C}$-space of the robot. The components of the GMM are constructed from a fixed *basis* of $K$ known solution paths from previously seen problem instances. The number of solution paths $K$ is tuned manually. These solution paths are represented as a sequence of points in $\mathcal{C}$-space. Every point along these $K$ paths induces a component centered at that point, and is given a fixed variance. Such a GMM is parameterized by a set of *coefficients* $w = (w_1, \dots, w_K)$ over the paths. The points along each path $i$ are weighted equally and sum up to the path's weight $w_i$.

Given a problem instance $c$, the choice of GMM, described by a set of coefficients $w$, is used to bias the chosen sampling-based planner, RRTConnect [17]. In every iteration of RRTConnect, we sample a new configuration from the GMM, which is used as a target to extend the bidirectional trees. The performance of



(a) Generator network



(b) Critic network

Figure 2: Neural network architectures of the generator and the critic. (a) The generator passes the occupancy grid (red) through three layers of 3D convolutions – each using 64 $3 \times 3 \times 3$ filters with leaky ReLU activation and max pooling (over a $2 \times 2 \times 2$ window). The hidden units are flattened into a vector of 1728 real entries. This is concatenated with the start vector (purple) and goal vector (blue), which vary in length depending on the robot used. The result is passed through three fully-connected layers with 512 hidden units each, using leaky ReLU activation. Finally, the output layer produces a vector of $K = 50$ real numbers. This real vector is used to parameterize a Dirichlet distribution over coefficients (specifically, by rectifying the entries to be $> 0$ via the exp function, and using the result as the concentration parameter of the Dirichlet distribution). Coefficient samples (of size $K = 50$) are then drawn from this distribution. (b) The critic similarly passes the occupancy grid (red) through three layers of 3D convolutions – each using 64 $3 \times 3 \times 3$ filters with leaky ReLU activation and max pooling (over a $2 \times 2 \times 2$ window). The hidden units are flattened into a vector of 1728 real entries. This is concatenated with the start vector (purple) and goal vector (blue), which vary in length depending on the robot used; and a choice of $K = 50$ coefficients (orange). The result is passed through three fully-connected layers with 512 hidden units each, using leaky ReLU activation. Finally, the output layer produces a vector of two real numbers, which is used to parameterize a Normal distribution approximating the performance objective (specifically, one entry is used as the mean parameter and the other is rectified to be $> 0$ via the exp function and used as the scale parameter of the Normal distribution).

this planning is described by a *performance objective* $V(c, w)$, which is defined as the number of the planner's internal planning iterations until a solution is found. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. We select this performance objective since it directly correlates to runtime efficiency. We note that since the planner is stochastic, this performance objective is a random variable.

The goal of this work is to find, for each problem instance $c$, a set of coefficients $w$ that maximize the expected planning performance $\mathbb{E}[V(c, w)]$.

## IV. Methodology

### A. Overview of APES

Using APES (Fig. 1), the problem instance $c$ is passed to a *generator* $G_\theta$ (Fig. 2a), which is a neural network with parameters $\theta$. The generator outputs $G_\theta(c)$, a Dirichlet distribution over coefficients. We use a distributional output instead of a single output to allow for better exploration [31], [32]. Coefficient samples $w = (w_1, \ldots, w_K)$ are drawn from the Dirichlet distribution, with the property that $w_i \geq 0$ and $w_1 + \cdots + w_K = 1$. These coefficients induce a GMM, which is used to bias the sampling of the downstream sampling-based planner.

The key challenge is to optimize the generator *directly* for the performance objective $V(c, w)$, which cannot be differentiated through the planner (RRTConnect). To do this, we adapt the recent *generator-critic* framework [16], which was first used to learn macro-actions for POMDP planning. In [16], a similar problem was identified – the performance of online POMDP planning, with respect to the choice of macro-actions, cannot be differentiated through the selected POMDP planner (since the POMDP planner is not differentiable). To circumvent this, the authors built a *critic* – which is a neural network that models the performance of the POMDP planner. The critic is used as

**Algorithm 1:** Generator training

```
1  $\theta \leftarrow$ InitGeneratorNetWeights()
2  $\psi \leftarrow$ InitCriticNetWeights()
3  $D \leftarrow$ InitReplayBuffer()
4  for $i = 1, \ldots, W$ do parallel
5      while TrainingIncomplete() do
6          $c \leftarrow$ SampleProblemInstance()
7          $w \sim G_\theta(c)$
8          $v \sim$ CallPlanner$(c, w)$
9          $D \leftarrow$ Append$(D, (c, w, v))$
10         $\{(c_i, w_i, v_i)\}_{i=1}^{M} =$ Sample$(D, M)$
11         $\psi \leftarrow \psi - \frac{1}{M}\nabla_\psi J_{\text{critic}}(\psi)$
12         $\theta \leftarrow \theta - \frac{1}{M}\nabla_\theta J_{\text{generator}}(\theta)$
13         $\alpha \leftarrow \alpha - \frac{1}{M}\nabla_\alpha J_{\text{entropy}}(\alpha)$
```

a *differentiable surrogate objective* to pass training gradients into the macro-actions.

We were inspired by [16] to develop APES. Here, our approach learns sampling distributions optimized for sampling-based planners. We similarly construct a *critic* $\hat{V}_\psi$ (Fig. 2b), which is a neural network parameterized by $\psi$, to approximate the performance objective $V(c, w)$. The critic $\hat{V}_\psi$ takes the problem instance $c$ and a choice of coefficients $w$. It returns $\hat{V}_\psi(c, w)$, a normal distribution that seeks to approximate the true downstream performance objective $V(c, w)$[1]. During training, both the critic and the generator are learned simultaneously. The critic $\hat{V}_\psi(c, w)$ is used as a *differentiable surrogate* of the performance objective $V(c, w)$, to allow approximate gradients from $V(c, w)$ to flow back to the generator. This optimizes the generator *directly* for the performance objective.

[1]Note that the performance objective is a random variable. Hence, both $V(c, w)$ and $\hat{V}_\psi(c, w)$ are distributions.

## B. Offline training pipeline

The training pipeline is outlined in Alg. 1. The weights for the generator and the critic are randomly initialized (Line 1 - Line 2). An empty *replay buffer* is created (Line 3) to store recent experience for training. $W$ asynchronous workers are run (Line 4) to collect data and to perform training. The workers share access to the same generator and critic weights and replay buffer, which are access-controlled by a set of global mutexes.

Each worker runs a data collection and training loop (Line 5). In each round, it first samples (Line 6) a random problem instance $c$. It then invokes (Line 7) the generator $G_\theta$ to produce a Dirichlet distribution over coefficients, from which a set of coefficients $w$ is sampled. A GMM is constructed from the coefficients $w$, and is used to bias the RRTConnect planner (Line 8). This is done by sampling new points from the GMM when extending the bidirectional trees maintained within RRTConnect. After planning, we retrieve the number of planning iterations used internally by the planner to find a solution. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. This value corresponds to a sample $v$ of the performance objective $V(c, w)$. The experience $(c, w, v)$ is added to the replay buffer (Line 9).

The worker proceeds to update the networks. A mini-batch with a fixed size of $M$ data points is sampled uniformly (Line 10) from the replay buffer (without repetition within each mini-batch). This is first used to update (Line 11) the critic by minimizing the negative log-likelihood loss (with respect to $\psi$):

$$J_{\text{critic}}(\psi) = \sum_{i=1}^{M} -\log p_\psi(v \mid c_i, w_i) \tag{1}$$

where $p_\psi(v \mid c_i, w_i)$ denotes the p.d.f of observing $v$ using the model $\hat{V}_\psi(c_i, w_i)$ produced by the critic. Then, the critic is used as a differentiable surrogate objective to update (Line 12) the generator, by minimizing the objective (with respect to $\theta$):

$$J_{\text{generator}}(\theta) = \sum_{i=1}^{M} \mathop{\mathbb{E}}_{w \sim G_\theta(c_i)} \left[ \mathbb{E}\left[ \hat{V}_\psi(c_i, w) \right] \right] - \alpha \cdot \mathcal{H}(G_\theta(c_i)). \tag{2}$$

The first term is a *planning term* representing the expected planner's performance over the coefficients distribution output by the generator. The second term is an *entropy regularization term* which prevents the generator from converging prematurely, by regularizing the entropy $\mathcal{H}(G_\theta(c_i))$ of its outputs. The weight $\alpha > 0$ of the entropy regularization term is then adjusted (Line 13) by minimizing the objective (with respect to $\alpha$):

$$J_{\text{entropy}}(\alpha) = \sum_{i=1}^{M} \alpha \cdot (\mathcal{H}(G_\theta(c_i)) - \mathcal{H}_0) \tag{3}$$

where we constrain $\alpha > 0$ through a rectifier[2]. This effectively enforces a minimum amount of entropy $\mathcal{H}_0$ in the coefficients produced by the generator, using the same scheme as [16], [32]. This finally completes a round of data collection and training for the worker, and the process is repeated up to an allowed number of training rounds.

[2]In particular, $\alpha = e^\beta$ with $\beta \in \mathbb{R}$.

## C. Reparameterization trick and chain rule

The planning term of Eqn. 2 does not have an analytical closed-form, since we are unable to compute exactly the expectation over $G_\theta(c)$. Instead, we approximate its gradient via the reparameterization trick [33]. The planning term can be rewritten as

$$\mathop{\mathbb{E}}_{w \sim G_\theta(c_i)} \left[ \mathbb{E}\left[ \hat{V}_\psi(c_i, w) \right] \right] \approx \mathop{\mathbb{E}}_{\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{1})} \left[ \mathbb{E}\left[ \hat{V}_\psi(c_i, f_\theta(\epsilon; c_i)) \right] \right] \tag{4}$$

where we reparameterize $G_\theta(c)$ as a deterministic function $f_\theta(\epsilon; c)$ – which takes a random seed vector $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and outputs a coefficient vector.

The value of Eqn. 4 can be estimated in terms of samples of $\epsilon$. Thus, using an estimate derived from a single sample of $\epsilon$, we apply the chain-rule to finally derive an approximate gradient of the planning term:

$$\begin{aligned} &\nabla_\theta \mathop{\mathbb{E}}_{w \sim G_\theta(c_i)} \left[ \mathbb{E}\left[ \hat{V}_\psi(c_i, w) \right] \right] \approx \\ &\nabla_w \mathbb{E}\left[ \hat{V}_\psi(c_i, w) \right] \Big|_{w = f_\theta(\epsilon; c_i)} \nabla_\theta f_\theta(\epsilon; c_i) \end{aligned} \tag{5}$$

where we evaluate the derivative at $w = f_\theta(\epsilon; c_i)$.

## V. Experiments

### A. Setup

**Problem specifications.** We apply APES to a series of motion planning problems for realistic robots in simulation – CAGE (Fig. 3a), BOOKSHELF (Fig. 3b), and TABLE (Fig. 3c), which were generated with MOTIONBENCHMAKER [7]. Within each problem, every instance contains variations in the position of the workspace objects, the start configuration, and the goal configuration. For CAGE, the 3D position of the cage varies by $\pm 10$ cm along all three axes, while its rotational offset from the robot varies by $\pm 30°$. The 2D position of the cube in the cage varies by $\pm 25$ cm along both axes, with orientation varying by $\pm 90°$. For BOOKSHELF, the 2D position of the bookshelf varies by $\pm 10$ cm along both axes, while its rotational offset from the robot varies by $\pm 30°$. Each cylinder has a fixed depth within its allocated compartment, while its lateral position varies by $\pm 45$ cm. In TABLE, the lateral position of each bar varies by $\pm 10$ cm. There is also variation in whether the robot is required to tuck its arms crossed or uncrossed.

For each problem, we pre-generate 500 instances for training and 500 instances for testing. We use RRTConnect with OMPL default parameters for planning, and limit the maximum number of iterations internally allowed by the planner to $1000, 10000$, and $100000$ respectively. The TABLE is given a much larger limit since it is substantially harder – the planner has to plan for two 7 DOF arms under a small space allowance.

**Training and evaluation procedure**. APES was applied to each problem as follows. First, we built a fixed path basis of size $K = 50$, using the solution paths of 50 randomly selected problem instances in the training set. Then, we train the generator using Alg. 1, for a total of $20000, 50000, 20000$ training rounds respectively across $W = 8$ asynchronous workers. A replay buffer of size 5000 was used, with mini-batch sizes of $M = 64$. These hyperparameters were selected

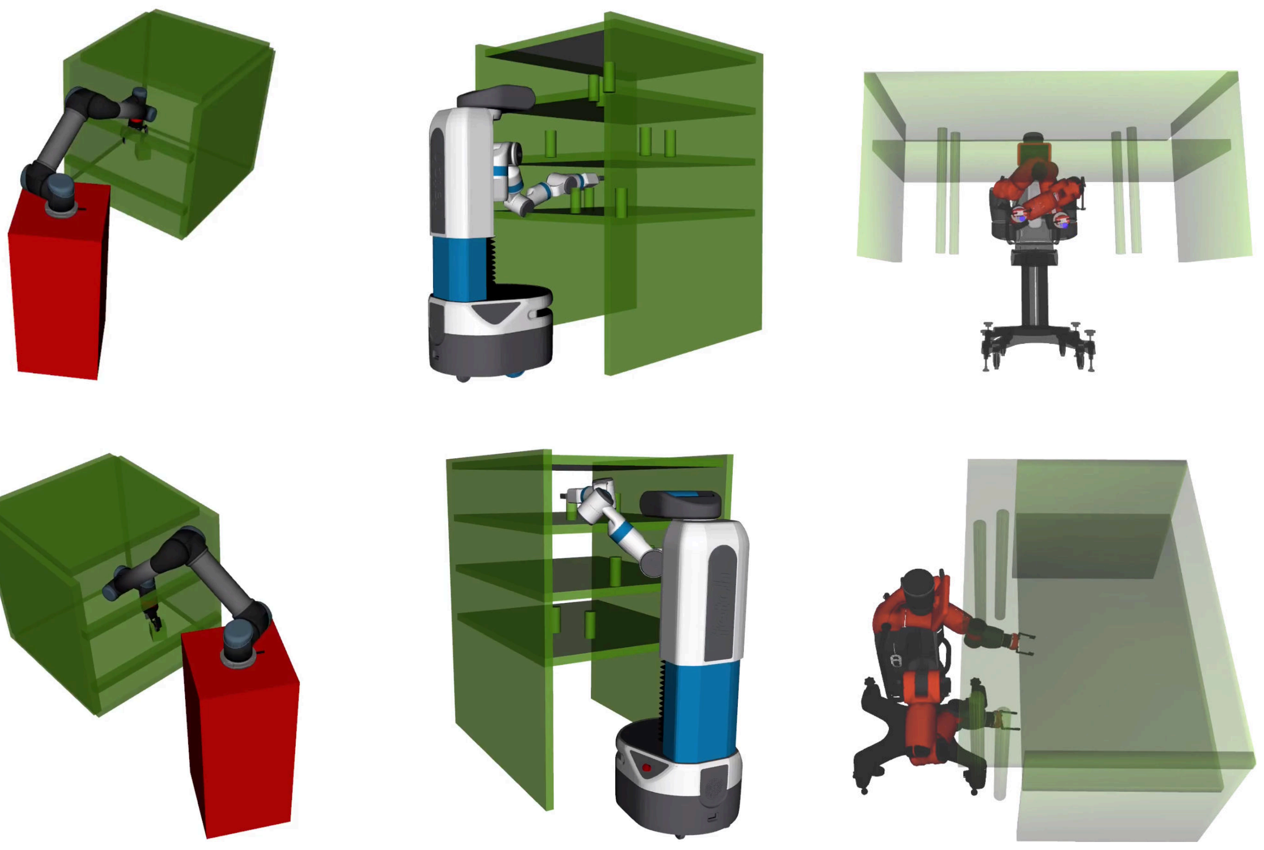

(a) CAGE (b) BOOKSHELF (c) TABLE

Figure 3: Samples of problem instances used in our experiments. The positions and orientations of the objects in the scene are randomized relative to the robot. The start configurations are uniformly sampled from the robot's collision-free $\mathcal{C}$-space. The goal configurations are randomly selected according to the problem. For illustration purposes, we show only the goal configurations. (a) CAGE problem: a UR5 (6-DOF) robot reaches for a box inside a constrained cage. (b) BOOKSHELF problem: a Fetch (8-DOF) robot reaches deep inside a bookshelf to pick the cylinder at the back. (c) TABLE problem: a Baxter (14-DOF) robot tucks both its arms between the bars under the table, given a small space allowance. In some problem instances, the arms need to be tucked crossed (top).

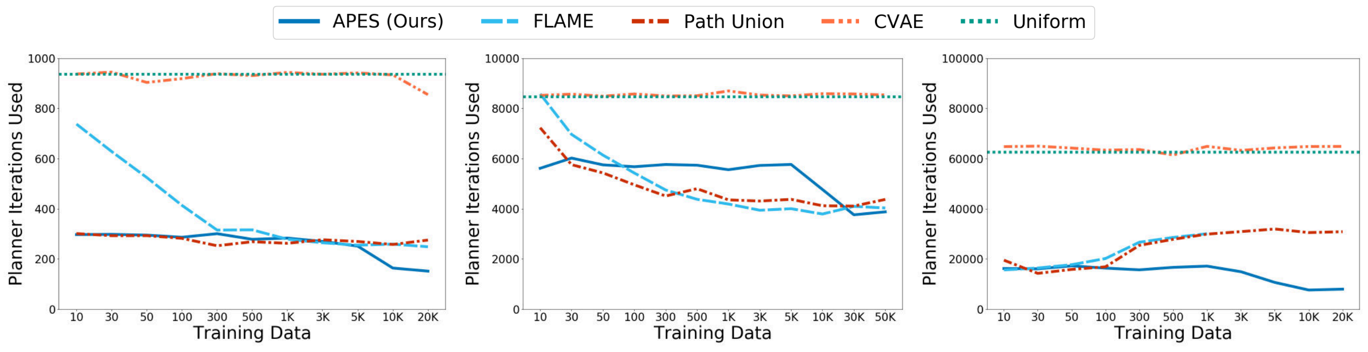
through manual tuning, as well as through consideration of compute limitations. After training, the fixed path basis and trained generator are evaluated on the set of 500 problem instances. All experiments are run on an Intel Xeon Gold 6130 (32x 3.7GHz) with 32GB of RAM and a NVIDIA RTX 3060.

**Baselines.** APES was compared to $4$ baselines. These include CVAE [15], which learns to reconstruct past solutions – conditioned on the workspace, start, and goal information – using a conditional variational autoencoder; FLAME [12], which builds a database of local samplers from path experience which are queried with a handcrafted similarity function over local workspace information; PATHUNION, which simply assigns equal weights to all waypoints in a set of path experience to build a GMM (thus learning *problem invariants* similar to Repetition Sampling in [22]); and vanilla UNIFORM which runs the planner with the typical uniform distribution. All baselines similarly use RRTConnect for planning with the same maximum allowed number of planner iterations. While APES uses only 500 training instances, it calls the planner repeatedly on these instances during training, up to a total of $20000, 50000, 20000$ invocations respectively. To maintain fairness, we provide $20000, 50000, 20000$ training instances to the baselines. We also tuned the hyperparameters for the baselines for optimal performance.

## B. *Training performance*

The training performance of APES compared to the baselines are shown in Fig. 4. In this work, we seek to optimize the generator for the downstream performance objective – which is to minimize the number of planning iterations used internally by the planner until a solution is found. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. This metric was chosen over the metric of time taken, since it correlates directly to sampling efficiency. On the other hand, the metric of running time is affected by other factors such as size of the trees maintained internally by RRTConnect, or overhead from preprocessing.

**Overall training performance.** Starting with a randomly initialized generator, APES was able to optimize the generator to produce high-quality coefficients. Given sufficient data, APES eventually outperformed the baselines in CAGE and TABLE, while achieving similar performance as FLAME in BOOKSHELF. Despite our best efforts in tuning CVAE, it had similar performance as UNIFORM. We believe that this is due to the multimodality of the motion planning problem [34] which cannot be captured by the unimodal output of CVAE. Additionally, in this work, we used a 3D occupancy grid to represent the workspace, instead of the geometric representations used in prior works for CVAE [12], [14]. This



(a) CAGE (b) BOOKSHELF (c) TABLE

Figure 4: Performance of APES against learning baselines with increasing amounts of training data. We evaluated the number of planning iterations used internally by the planner until a solution is found, averaging the results over $500$ test problems. Each planner iteration involves drawing a sample from the learned sampling distributions, to extend the bidirectional trees maintained in RRTConnect. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. The performance of UNIFORM is shown as a horizontal line for comparison. The missing evaluations for FLAME in the TABLE problem could not be done within the 32GB memory limit, and are omitted.

| | CAGE | | | BOOKSHELF | | | TABLE | | |
|---|---|---|---|---|---|---|---|---|---|
| | Iterations | Success Rate | Time (s) | Iterations | Success Rate | Time (s) | Iterations | Success Rate | Time (s) |
| APES (Ours) | **152 (12)** | **0.94 (0.01)** | **42 (3)** | **3765 (190)** | 0.73 (0.02) | **59 (3)** | **8014 (820)** | **0.98 (0.01)** | **61 (7)** |
| CVAE | 855 (13) | 0.15 (0.02) | 113 (2) | 8494 (130) | 0.27 (0.02) | 152 (2) | 61493 (4500) | 0.41 (0.02) | 2397 (90) |
| FLAME | 249 (12) | **0.94 (0.01)** | 97 (5) | **3796 (130)** | **0.89 (0.01)** | 104 (2) | 15670 (1200) | 0.93 (0.01) | 156 (11) |
| PATHUNION | 253 (12) | **0.93 (0.01)** | 89 (5) | 4113 (160) | 0.81 (0.02) | 122 (4) | 14310 (1000) | **0.96 (0.01)** | 150 (11) |
| UNIFORM | 936 (8) | 0.14 (0.02) | 87 (2) | 8470 (120) | 0.30 (0.02) | 173 (3) | 62610 (2100) | 0.41 (0.02) | 2559 (96) |

Table I: Detailed performance of APES and baselines across the test problems. *Iterations* refer to the number of planning iterations used internally by the planner until a solution is found. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. *Success rate* refers to the probability of finding a solution within the iteration budget. Numbers indicate the mean, with standard errors in braces. *Time* refers to the time taken to find a solution or to use up the iteration budget. Bold entries indicate the best-performing results.

likely made it harder to learn a mapping from workspace to $\mathcal{C}$-space.

**Comparing data usage.** The baselines generally took much fewer training data to converge compared to APES. However, APES is able to utilize larger amounts of data to improve performance. For the case of using FLAME for the TABLE problem, there is a noticeable memory overhead which makes it infeasible to execute for very large amounts of data. Interestingly, FLAME and PATHUNION seem to degrade in performance with more data for the TABLE problem. A possible explanation is that for TABLE, the solutions require exploring small regions in a concentrated manner. Using less data, FLAME and PATHUNION were able to concentrate the planner on such small regions of the $\mathcal{C}$-space. With more data, the sampling distributions spread out over a larger volume, causing it to dilute this concentration.

## C. Planning results

In Table I, we analyzed the performance of the learned generator against the other baselines. In addition to the metric of planning iterations used internally by the planner, we also show the success rate of finding a solution in the allowed number of iterations, and the time taken.

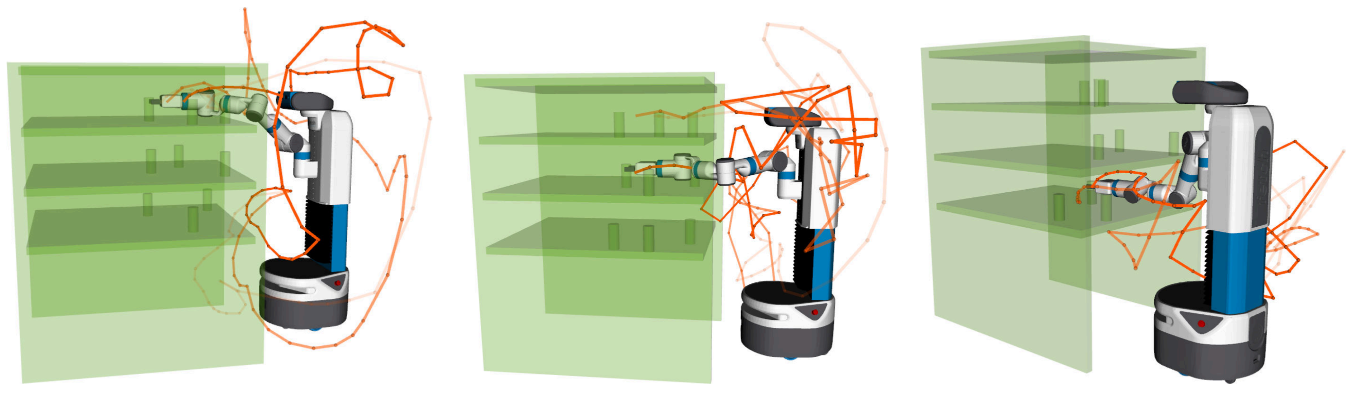
**Overall planning performance.** Our results show that APES produces the best sampling distributions for RRTConnect, in terms of the number of planning iterations used internally by the planner. We also achieved the highest success rate for CAGE and TABLE. APES also achieves the lowest planning time across all problems.

**Trade-off between metrics.** In CAGE and TABLE, APES achieves a success rate similar to FLAME and PATHUNION, but uses fewer iterations. This indicates that APES was able to identify regions of the $\mathcal{C}$-space that allow for fewer planning iterations despite providing the same success rate. On the other hand, in BOOKSHELF, APES achieves a similar number of planning iterations used while having a lower success rate compared to FLAME. This indicates that APES has favored riskier regions which cut down planning iterations. We also note that the metric of planning time time appears disproportionate. This is because the time taken is affected by various additional factors. For example, RRTConnect maintains an internal structure of the trees grown for nearest-neighbour lookup, which grows with more samples added to the tree. Since we are primarily concerned with the efficiency of samples, we focus on the number of planning iterations instead of the time taken.

**What have we learned?** To illustrate the effect of learning using APES, we visualize the learned coefficients in Fig. 5 for the BOOKSHELF problem. We select three problem instances where the robot has to reach into different compartments of the bookshelf. For each instance, we call the learned generator and visualize the top three paths with the highest coefficients. From Fig. 5, it can be seen that depending on the goal, the proposed paths lead towards the corresponding compartment of the bookshelf.

## D. Ablation study

In Table II, we conducted an additional ablation study to investigate how much the information of the workspace, the



(a) Upper compartment (b) Middle compartment (c) Lower compartment

Figure 5: Visualization of the sampling distributions learned by APES, for different problem instances of BOOKSHELF. The paths visualized are taken from the path basis used by APES. We show only the top three paths with the highest coefficients. The relative value of a path's coefficient is indicated by its opacity. (a-c) It can be seen from the visualizations that depending on the object the robot needs to pick, the most prominent paths guide the robot towards the corresponding compartment of the bookshelf.

start, and the goal affect performance. We use variants of the generator that take only the workspace information or the start and goal information when generating coefficients. Additionally, we included an unconditioned version where we learned a fixed coefficient distribution for all workspaces, start, and goals.

**Conditioning generally improves performance.** Our results show that in all problems, information of all of the workspace, the start, and the goal is beneficial to achieving the best results using APES. In CAGE, both workspace and start and goal information appear equally important. In BOOKSHELF and TABLE, information about the start and goal appears to be more important.

**Comparing with baselines.** We note that APES generally performs better than the baselines (in Table I) when given the same amount of information – CVAE uses workspace, start, and goal; FLAME uses only workspace information; while PATHUNION is unconditioned. This indicates that by optimizing directly for planning performance, APES can utilize the given information more efficiently. The exception is the unconditioned version of APES applied to BOOKSHELF, which performs worse than PATHUNION. Indeed, unconditioned APES simply resolves to a weighted variant of PATHUNION where each path is given a learned set of weights. Since PATHUNION is constructed from many more paths (compared to a basis of just 50 paths in APES), it is likely that the wide coverage of paths in PATHUNION was able to compensate for the lack of optimization of path weights.

## VI. Discussion

In this paper, we proposed APES – a learning method that can be used to informatively guide a sampling-based planner using a biased sampling distribution. The key advantage of APES is that it can directly optimize for the desired performance objective – in our case, the number of planning iterations used internally by the planner until a valid path is found. Additionally, APES is able to utilize all the available information to condition the sampling distribution. We demonstrate the benefits of APES in 3 challenging manipulation problems where it outperforms other baselines.

| | CAGE | BOOKSHELF | TABLE |
|---|---|---|---|
| All Information | **152 (12)** | **3765 (190)** | 8014 (820) |
| Workspace Only | 219 (14) | 5458 (190) | 10694 (1000) |
| Start & Goal Only | 212 (14) | 4218 (190) | 9275 (930) |
| Unconditioned | 241 (14) | 5363 (180) | 9216 (880) |

Table II: Ablation study using partial problem information. We show the number of planning iterations used internally by the planner until a solution is found. The maximum allowed number of iterations is taken if the planner fails to find a solution within the given budget. Numbers indicate the mean, with standard errors in braces. Bold entries indicate the best performing results.

One of the limitations of our current approach is that the selection of the path basis greatly affects the performance of the learned distributions. Currently, the path basis is randomly selected and fixed, and it is not clear how to optimize it. In our future work, we would like to investigate ways to improve upon this. Other future work could investigate using APES with different planners and performance objectives. It could also include a deeper investigation into the generalization capabilities of APES, such as across different problems, planners, and robots.

This work could also aid in investigating the connection between the choice of 1) sampling-based planner, 2) performance metric, and 3) sampling distribution. For example, is it possible that different planners have different optimal sampling distributions? Is it possible that a sampling distribution which maximizes the success probability is completely different from one which minimizes the iterations used? Indeed, we can easily swap these components in APES, with minimal changes to the training pipeline. Overall, we hope that our work will serve as a general tool to better understand the relationship between sampling-based planners and the sampling distributions which they use.

## References

[1] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, 2005.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 4, pp. 265–293, 2021.

[3] D. Le and E. Plaku, “Cooperative multi-robot sampling-based motion planning with dynamics,” in *Int. Conf. on Automated Planning and Scheduling*, 2017.

[4] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. M. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robot. Autom. Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[7] 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 Robot. Autom. Letters*, vol. 7, no. 2, pp. 882–889, 2022.

[8] V. Boor, M. H. Overmars, and a. V. D. Stappen, “The gaussian sampling strategy for probabilistic roadmap planners,” in *IEEE Int. Conf. Robot. Autom.*, 1999.

[9] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin, “On finding narrow passages with probabilistic roadmap planners,” in *Int. Wksp. on the Algorithmic Foundations of Robotics*, 1998.

[10] J.-M. Lien, S. L. Thomas, and N. M. Amato, “A general framework for sampling on the medial axis of the free space,” in *IEEE Int. Conf. Robot. Autom.*, 2003.

[11] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust, “Learned critical probabilistic roadmaps for robotic motion planning,” in *IEEE Int. Conf. Robot. Autom.*, 2020.

[12] C. Chamzas, Z. Kingston, C. Quintero-Peña, A. Shrivastava, and L. E. Kavraki, “Learning sampling distributions using local 3D workspace decompositions for motion planning in high dimensions,” in *IEEE Int. Conf. Robot. Autom.*, 2021.

[13] P. Lehner and A. Albu-Schaeffer, “The repetition roadmap for repetitive constrained motion planning,” *IEEE Robot. Autom. Letters*, vol. 3, no. 3, pp. 3884–3891, 2018.

[14] R. Kumar, A. Mandalika, S. Choudhury, and S. S. Srinivasa, “LEGO: Leveraging experience in roadmap generation for sampling-based planning,” *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2019.

[15] B. Ichter, J. Harrison, and M. Pavone, “Learning sampling distributions for robot motion planning,” in *IEEE Int. Conf. Robot. Autom.*, 2018.

[16] Y. Lee, P. Cai, and D. Hsu, “MAGIC: Learning macro-actions for online POMDP planning,” in *Robotics: Science and Syst.*, 2021.

[17] J. J. Kuffner and S. M. LaValle, “RRT-Connect: An efficient approach to single-query path planning,” in *IEEE Int. Conf. Robot. Autom.*, 2000.

[18] O. Brock and L. E. Kavraki, “Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces,” in *IEEE Int. Conf. Robot. Autom.*, 2001.

[19] H. Kurniawati and D. Hsu, “Workspace importance sampling for probabilistic roadmap planning,” in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2004.

[20] A. Orthey and M. Toussaint, “Section patterns: Efficiently solving narrow passage problems in multilevel motion planning,” *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1891 – 1905, 2021.

[21] T. F. Iversen and L.-P. Ellekilde, “Kernel density estimation based self-learning sampling strategy for motion planning of repetitive tasks,” in *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2016.

[22] P. Lehner and A. Albu-Schaeffer, “Repetition sampling for efficiently planning similar constrained manipulation tasks,” *IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, 2017.

[23] D. Berenson, P. Abbeel, and K. Goldberg, “A robot path planning framework that learns from experience,” in *IEEE Int. Conf. Robot. Autom.*, 2012.

[24] D. Coleman, I. A. Sucan, M. Moll, K. Okada, and N. Correll, “Experience-based planning with sparse roadmap spanners,” in *IEEE Int. Conf. Robot. Autom.*, 2015.

[25] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, “Memory of motion for warm-starting trajectory optimization,” *IEEE Robot. Autom. Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.

[26] E. Pairet, C. Chamzas, Y. R. Petillot, and L. E. Kavraki, “Path planning for manipulation using experience-driven random trees,” *IEEE Robot. Autom. Letters*, vol. 6, no. 2, p. 3295–3302, 2021.

[27] M. Zucker, J. Kuffner, and J. A. Bagnell, “Adaptive workspace biasing for sampling-based planners,” *IEEE Int. Conf. Robot. Autom.*, 2008.

[28] D. Molina, K. Kumar, and S. Srivastava, “Learn and link: Learning critical regions for efficient planning,” in *IEEE Int. Conf. Robot. Autom.*, 2020.

[29] C. Chamzas, A. Shrivastava, and L. E. Kavraki, “Using local experiences for global motion planning,” in *IEEE Int. Conf. Robot. Autom.*, 2019.

[30] E. Y. Puang, P. Lehner, Z.-C. Marton, M. Durner, R. Triebel, and A. Albu-Schäffer, “Visual repetition sampling for robot manipulation planning,” in *IEEE Int. Conf. Robot. Autom.*, 2019.

[31] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *Int. Conf. on Learning Representations*, 2016.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Int. Conf. on Machine Learning*, 2018.

[33] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *Int. Conf. on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2014.

[34] W. Merkt, V. Ivan, T. Dinev, I. Havoutis, and S. Vijayakumar, “Memory clustering using persistent homology for multimodality- and discontinuity-sensitive learning of optimal control warm-starts,” *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1649–1660, 2021.