

Appendices to “Embodied Lifelong Learning for Task and Motion Planning”

Anonymous Author(s)

A Experimental Details

This section provides additional details about the experiments in Section 5 in the main paper.

A.1 Network Architectures and Diffusion Models

All network architectures used throughout our work were simple multi-layer perceptrons with two hidden layers of 256 nodes each. For our nested models, the hidden layers were shared between the generative model p_ψ and the auxiliary predictor f , and only the output layer was separate—note that there was no sharing of parameters between specialized p_ℓ and generic p samplers. All training proceeded for 1,000 epochs over the training data in mini-batches of size 512 (when that many samples were available, and a single batch otherwise), using an Adam optimizer with the default hyperparameters of PyTorch, including a learning rate of 10^{-3} . The one exception was the replay model, which used 100 epochs of training during model adaptation.

To train the diffusion models, for each point (s, ϕ) , a time step $t \in [1, T]$ was randomly chosen for $T = 100$, and a sample was drawn from the forward process $\phi_t = \phi\sqrt{\bar{\alpha}_t} + \epsilon\sqrt{1 - \bar{\alpha}_t}$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, I)$ is standard Gaussian noise and $\bar{\alpha}_t$ is a scaling constant obtained by expanding the expression of $q(\cdot)$. The loss function then measured how closely $\epsilon_\psi(\phi_t, \mathbf{x}, t)$ approximated the true noise ϵ : $\mathcal{L}(s, \phi, t, \epsilon) = \|\epsilon_\psi(\phi_t, \mathbf{x}, t) - \epsilon\|$. Once trained, the planner sampled from the model by simulating the reverse process $p_\psi(\phi_{T:0} | s)$.

To process the input composed of three vectors (ϕ_t, \mathbf{x}, t) , the time index t was first processed using sinusoidal positional embeddings [6] of the same dimension as \mathbf{x} . Then, the three vectors were concatenated into a single input to the network. Since the auxiliary predictor shared the same base layers of the network, we used $t = 0$ as a constant input to f .

All inputs and outputs were scaled to $[0, 1]$, except when the range of the variable was less than 1, in which case the given variable was shifted to 0, but not rescaled.

A.2 2D Domain Experimental Setting

We now provide the details of our evaluations on the 2D domains we created.

A.2.1 Domain Descriptions

We created five different 2D domains, specially crafted to require distinct sampling distributions across objects. Figure A.1 depicts the simulated domains.

- **Books** Various rectangular books of sides $w_{\text{book}} \in [0.5, 1]$, $l_{\text{book}} \in [1, 1.5]$ are scattered in a room and must be picked and placed on a rectangular shelf of sides $w_{\text{shelf}} \in [2, 5]$, $l_{\text{shelf}} \in [5, 10]$.
- **Cups** Square cups with sides $l_{\text{cup}} \in [0.5, 1]$ must be picked *by the handle* (one specific side) and palced in a cupboard of sides $w_{\text{cupboard}} \in [2, 5]$, $l_{\text{cupboard}} \in [5, 10]$. The cupboard is always against a wall.
- **Boxes** Boxes of sides $w_{\text{box}} \in [0.5, 1]$, $l_{\text{box}} \in [1, 1.5]$ must be placed *on pockets at the extremes* of a tray of width $w_{\text{tray}} \in [3, 5]$ and length $l_{\text{tray}} \in [11, 13]$.

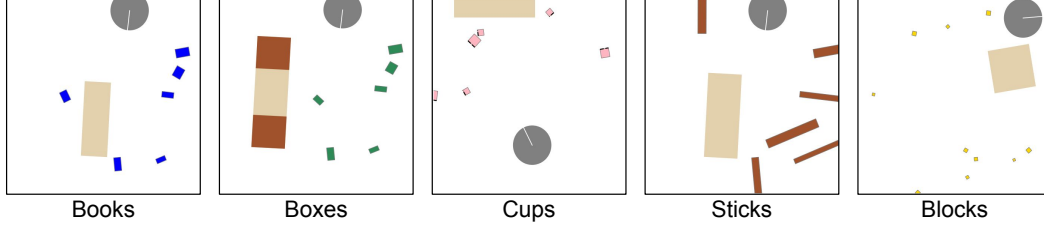


Figure A.1: 2D domains used to evaluate our sampler learning approaches. The objects in each domain have properties that ensure that samplers must generate diverse candidate action parameters to solve the tasks.

- **Books** Small square blocks of sides $l_{\text{block}} \in [0.25, 0.5]$ must be placed in a square bin of sides $l_{\text{bin}} \in [4, 6]$. While previous tasks contain $n \in [4, 5]$ objects, these require placing $n \in [9, 10]$ blocks.
- **Sticks** Long sticks of sides $w_{\text{stick}} \in [0.5, 1]$, $l_{\text{stick}} \in [5, 6]$ in a container of sides $w_{\text{container}} \in [3, 5]$, $l_{\text{container}} \in [7, 10]$.

The robot has three controllers that it can execute: `NAVIGATETo(object)`, parameterized by the relative target coordinates normalized by the object’s size; `PICK(object)`, parameterized by the length to extend the robot’s gripper and the angle to hold the object at; and `PLACE(object, container)`, parameterized by the gripper extension.

A.2.2 Auxiliary Geometric Signals

This section describes the auxiliary signals we used to train our predictors f , and we later describe how those were used to construct the mixture distributions for our samplers. We used the following auxiliary signals for each form of controller:

- **NAVIGATETo**: distance to the nearest point between the robot and the object—in the case of the cup, this signal measured distance to the handle, and in the case of the tray, it measured distance to the nearest pocket; coordinates of the nearest point on the object’s boundaries (in the object’s frame); target coordinates (in the world frame); and target coordinates (in the object’s frame)
- **PICK**: position of the gripper’s point (in the world frame); and position of the gripper’s point (in the object’s frame)
- **PLACE**: position of the center of mass of the object (in the world frame); and position of the center of mass of the object (in the container’s frame).

Note that all these signals measure the *intended effects* of an action, but cannot measure the actual attained effect, which would require knowledge of all objects in the world. However, by measuring the agent’s accuracy in predicting these signals, we can assess how well-trained it is in the neighboring region of the current state, and use that as a measure of how well its samples may generalize.

A.2.3 Constructing the Mixture Distribution

In these 2D domains, we created mixture distributions over three mixture components: a generic sampler trained on all object types, a specialized sampler for each object type, and a fixed uniform sampler over the parameter space of the controller. We used the inverse of the root mean square error as the assessment of reliability, ρ . For this, we first computed (offline) the average prediction error for random guessing via simulation, and assigned this fixed error value to the uniform sampler. Then, we used this value to normalize prediction errors across the various signals.

521 A.2.4 Lifelong Training Details

522 In the lifelong setting, upon facing a new problem, the agent used its mixture sampler to generate
523 samples for any previously seen object type. For unknown types, the agent used a mixture over the
524 generic and the uniform sampler with 0.5 weight for each. At the very beginning, the samplers were
525 initialized with a uniform distribution over the parameter space.

526 A.2.5 Evaluation Protocols

527 In the offline setting, we generated 50 test tasks for each of 10 trials, with varying random seeds
528 controlling the sizes of objects and their placements.

529 In the lifelong setting, each trial shuffled the order of the domains using the random seed, and
530 presented the agent with a sequence of 500 tasks from each domain. Instead of updating the models
531 after each task, which would render most updates very minor, we updated the models at intervals of
532 50 tasks, resulting in a total of 10 model updates per domain.

533 In both settings, we used Fast-Downward as the skeleton generator, getting a single skeleton for each
534 task (i.e., $N = 1$) and setting the maximum total number of samples to $B = 10,000$. During search,
535 a maximum of $M = 100$ samples were attempted at any given state before backtracking. We did not
536 impose a timeout for these experiments.

537 A.3 BEHAVIOR Domain Experimental Setting

538 Next, we describe the precise details of our lifelong learning evaluation on BEHAVIOR tasks.

539 A.3.1 Domain Descriptions

540 We considered 10 different BEHAVIOR tasks: `boxing_books_up_for_storage`,
541 `collecting_aluminum_cans`, `locking_every_door`, `locking_every_window`,
542 `organizing_file_cabinet`, `polishing_furniture`, `putting_leftovers_away`,
543 `re-shelving_library_books`, `throwing_away_leftovers`, and `unpacking_suitcase`.
544 We followed the evaluation setting from prior work to adapt BEHAV-
545 IOR domains to the TAMP setting [23]. In particular, only actions with
546 `NAVIGATE_TO(object)`, `GRASP(object, surface)`, `PLACE_ON_TOP(object, surface)`,
547 `PLACE_INSIDE(object, surface)`, `PLACE_UNDER(object, surface)`, and
548 `PLACE_NEXT_TO(object, target, surface)` controllers were implemented at the continu-
549 ous level, while other actions (e.g., `CLEAN_DUSTY` or `OPEN`) were implemented only at the abstract
550 level and assumed to always succeed.

551 A.3.2 Auxiliary Geometric Signals

552 The auxiliary signals that we used to assess each sampler’s reliability were:

- 553 • `NAVIGATE_TO`: sine and cosine of the robot’s yaw; distance to target; nearest point on the
554 object’s bounding box (in the object’s frame); distance to the nearest point in the object’s
555 bounding box; and robot position (in the object’s frame).
- 556 • `GRASP`: sine and cosine of the Euler angles of the robot’s gripper; distance of the gripper
557 to the target and the surface; distance of the gripper to the nearest point on the target’s and
558 surface’s bounding boxes; position, and sine and cosine of the Euler angles, of the gripper’s
559 pose in the target and surface coordinate frames.
- 560 • `PLACE...`: distance from hand and object to surface; nearest points from hand and object
561 to surface’s bounding box; nearest point from hand to object’s bounding box; distances to
562 these nearest points; positions, and sines and cosines of Euler angles, of the gripper’s and
563 object’s poses in the surface’s coordinate frame. For `PLACE_NEXT_TO`, additionally compute
564 the relevant distances, nearest points, and relative coordinates with respect to the target
565 object.

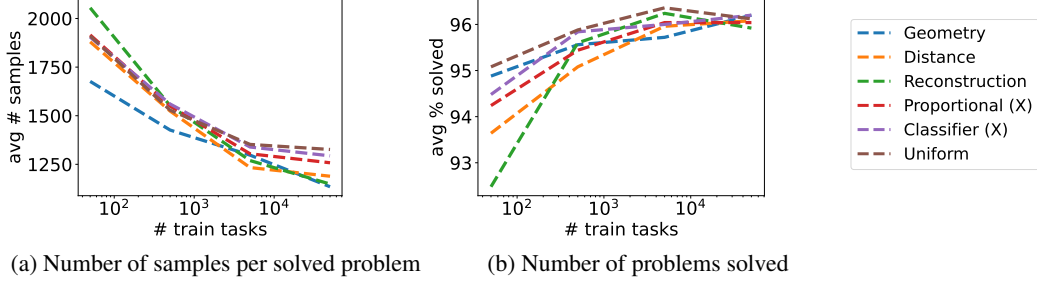


Figure A.2: Alternative choices of the mixture strategy to generate samples for TAMP in the 2D domains. Our geometric prediction method is most effective in the low-data setting.

566 Like in the 2D domains, these signals measure only intended effects, but have no means to effectively
 567 measure if those effects are attained (e.g., due to collisions with unforeseen objects).

568 A.3.3 Constructing the Mixture Distribution

569 In BEHAVIOR domains, we only considered the trained specialized and generic samplers as mixture
 570 components, since computing the uniform sampler’s error like in the 2D case would have required
 571 precomputing the error of random predictors via simulation, which was prohibitively expensive for
 572 BEHAVIOR. In consequence, we used the root mean square error directly (without normalization) to
 573 weight the two mixture components.

574 A.3.4 Lifelong Training Details

575 In the lifelong setting, we only used the learned samplers for exploration when both generic and
 576 specialized samplers had been trained. In consequence, whenever a new object type was encountered,
 577 hand-crafted samplers were used. At the start of the robot’s lifetime, all samplers were initialized
 578 to hand-crafted distributions from prior work [23]—note that, for BEHAVIOR domains, a uniform
 579 distribution would never complete tasks within any reasonable time limit.

580 A.3.5 Evaluation Protocols

581 We repeated the BEHAVIOR experiments over 8 trials with varying random seeds, which controlled
 582 both the order of BEHAVIOR tasks and the sampled problems within each. We trained the agent
 583 sequentially on six randomly chosen tasks in each trial. We presented the robot with 96 tasks of each
 584 type in sequence, and updated models every 48 tasks.

585 We again used Fast-Downward as the skeleton generator with $N = 1$. We set the sample bound to
 586 $B = 1,000$, with up to $M = 10$ samples at each state before backtracking. We did not use a timeout
 587 for these experiments.

588 B Additional Experiments

589 In this section, we present ablations and additional experiments to those presented in Section 5 in the
 590 main paper.

591 B.1 Evaluating the Mixture Weight Construction

592 While the results of our main experiments strongly support the choice of using mixture distributions
 593 for generating samples for TAMP, we were interested in more clearly understanding the choices of
 594 how to construct those mixture distributions. For this purpose, we implemented and evaluated five
 595 alternative strategies for constructing the mixture distribution from our nested models:

- **Distance** Our first alternative mixture still followed the process of training auxiliary models but, unlike our main implementation, uses only a single auxiliary variable: distance to target. This allows us to verify whether a collection of auxiliary signals is necessary, or a single one may suffice
- **Reconstruction** We similarly create an auxiliary model for directly reconstructing the state features \mathbf{x} . With this, we check the usefulness of including the action parameters ϕ in the auxiliary tasks.
- **Uniform** This strategy simply uses a uniform mixture distribution. The purpose of evaluating this technique is to check whether all the gains of mixture distributions come from the mere fact of using a mixture, or the weighting plays an important role.
- **Proportional** This *cheating* method observes the outcomes of the uniform mixture over all test tasks, and computes the proportion of successful samples that were drawn from each mixture component. We include this strategy to check whether there may exist some fixed choice of mixture weights that works *across all states*.
- **Classifier** This additional *cheating* method also observes the outcomes of the uniform mixture and trains a classifier to generate the mixture weights. This enables us to study whether it may be possible to train a model to directly choose which sampler to use, as an alternative to using auxiliary tasks as an assessment of reliability.

The results across the five 2D domains are shown in Figure A.2. While all mixture choices indeed perform well, in the low-data regime, which is crucial in lifelong settings, our geometric predictions lead to the highest efficiency across all mixture choices. Notably, the uniform mixture solves the largest fraction of problems. This is expected, given that our mixtures include the uniform sampler over the action space, which is guaranteed to eventually find a successful sample; only the uniform mixture guarantees that this sampler is used sufficiently often to guarantee solving most problems (albeit less efficiently than other samplers). The reconstruction error performs worst of all in the low-data setting, but eventually matches the performance of our geometry-prediction implementation; this validates the importance of including the action parameters ϕ as part of the auxiliary signal computation. Neither of the strategies that cheat is especially strong, indicating that 1) fixed mixture weights are not sufficiently flexible and 2) directly predicting which sampler to use, given the state, is difficult.

B.2 Replay Training Matches Full Retraining in Lifelong Setting on 2D Domains

To validate our use of balanced replay instead of full retraining, with $10\times$ gains in training speed due to starting from previously trained models, we compared the performance of the two methods on a lifelong sequence of 2D domains. Results in Figure B.3 validate this claim.

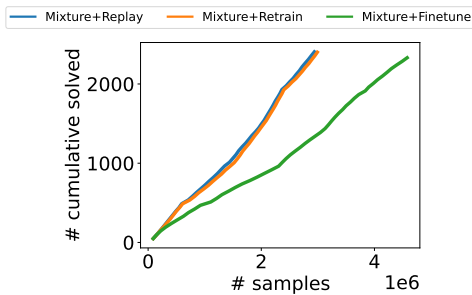


Figure B.3: Comparison of retraining vs replay on the lifelong learning evaluation in 2D domains. Replay (which is more efficient) matches the performance of retraining over the sequence of problems.