SAFARI: Safe and Active Robot Imitation Learning with Imagination

Norman Di Palo
The Robot Learning Lab
Imperial College London
n.di-palo20@imperial.ac.uk

Edward Johns
The Robot Learning Lab
Imperial College London

Abstract

One of the main issues in Learning from Demonstration is the erroneous behavior of an agent when facing out-of-distribution situations. In this work, we tackle this problem by introducing a novel active learning and control algorithm, SAFARI. During training, it allows an agent to request further human demonstrations when these out-of-distributions are met. At deployment, it combines model-free acting using behavioural cloning with model-based planning to reduce state-distribution shift, using future state reconstruction as a test for state familiarity. We empirically demonstrate how this method increases the performance on a set of manipulation tasks by a substantial margin in both simulated and real robotics scenarios.

1 Introduction

To teach a robot a novel task, one promising avenue of research is **Learning from Demonstration** (LfD) (Osa et al. (2018); Hussein et al. (2017); Ghasemipour et al. (2020)), and in particular **Behavioural Cloning** (BC). One of the main drawbacks of this method is caused by the erroneous behavior of policy networks outside the training distribution (Pomerleau (1989); Ghasemipour et al. (2020)). This may be caused by both a poor state-space visitation in the demonstrations and test time drifting, caused by accumulated errors and noise.

In this work we propose a novel active imitation learning and control method that tackles the aforementioned problems: **SAFARI** (**SAF**e and **A**ctive **R**obot Imitation Learning with **I**magination). SAFARI is a method for imitation learning and control which allows the robot to actively ask for more informative demonstrations at training time, actively minimizing distributional shift at test time, and predicting test time failures to improve safety. As we show in this work, these methods all contribute to tackle the aforementioned issues from three different perspectives, resulting in a substantial improvement in performance and safety.

Specifically, SAFARI uses an interplay between a **policy network**, a **learned dynamics network** (also referred as **world model** in this work), and an **epistemic uncertainty network**. The policy network is trained to emulate the expert's behavior with Behavioural Cloning, while the dynamics network is trained to predict future states (Nagabandi et al. (2018)), enabling planning alongside model-free policy acting, and the uncertainty network Di Palo and Valpola (2018); Boney et al. (2019) is trained to estimate epistemic uncertainty/out-of-distribution states.

With "uncertainty" we will always refer to epistemic uncertainty: dealing with aleatoric uncertainty is outside the scope of this work.

We test our method both on simulated and real robotics environments, obtaining substantial improvements over baselines in both scenarios.

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

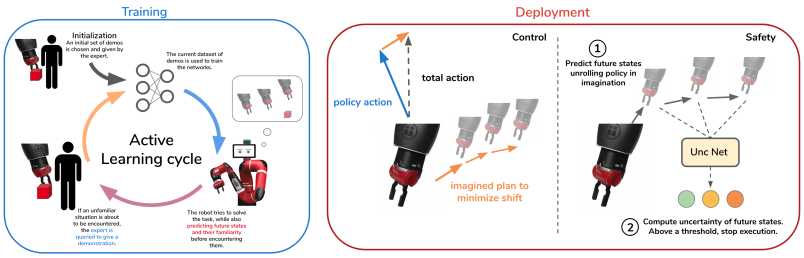

Figure 1: An overview of our training and control method, SAFARI.

2 Related Work

Learning from Demonstration with deep neural networks has shown impressive results in recent years in robotics. (Argall et al. (2009); Zhang et al. (2018); James et al. (2018); Gupta et al. (2019)). Active Learning (Hanczor (2018); Argall et al. (2009)) deals with the design of methods able to actively ask an expert for labels on unseen data. Ross et al. (2011) proposed DAgger, with several variants proposed over the years (Cronrath et al.; Hanczor (2018)), but these methods cannot be applied efficiently to real-world systems. We extend Di Palo and Johns (2019) with a novel control method, and testing the algorithm on a real robot. Filos et al. (2020) is the closest work to our method, but applied to a very different scenario: simulated autonomous vehicles. While proposing a similar approach, they don't use a policy network and don't learn a dynamics function, but compute a series of desired future states and assume the existence of an inverse dynamical model which can then infer actions.

3 Method

3.1 Active Imitation Learning

We propose to have an **iterative, interactive approach to gathering demonstrations**. Instead of receiving all the demonstration beforehand, the agent can request a demonstration by interacting with its environment and stopping when its uncertainty surpasses a certain threshold. The uncertainty estimation method is described in section 3.3. Our experiments, detailed in Section 4, demonstrate how we can train a better policy network using active demonstrations than receiving the same number of demonstrations passively beforehand.

3.2 Networks Models and Architectures

Inspired by previous works on Learning from Demonstration Lynch et al. (2019); Levine et al. (2016); Finn et al. (2016); Zhang et al. (2018), we use a feed-forward neural network f_θ to parametrize our policy, taking as input the current observation, and computing as output the action. The policy network is trained with Behavioural Cloning.

To model the uncertainty of the agent, we use Denoising Autoencoders (DAE), g_ϕ (Vincent et al. (2010); Arponen et al. (2017), as described in Boney et al. (2019). Different kinds of architectures have been tested, more information can be found in the Appendix.

As a learned dynamics model, we use a feed-forward network d_γ that takes as input the current state and action and predicts the one step difference $\Delta\hat{x}_{t+1}$ such that $\hat{x}_{t+1} = x_t + \Delta\hat{x}_{t+1}$ (Nagabandi et al. (2018)). This network is trained using the same demonstration trajectories collected by the expert.

All the networks are trained together on the same data (although used in different way) as described in Section 4 and Algorithm 1 in the Appendix.

3.3 Uncertainty Estimation with Imagined Rollouts

To compute the uncertainty on a particular state, we use an interplay of the three previously mentioned networks, as we show in Figure 3 (right, appendix).

From a state s_t, we use the Policy Network and the Dynamics Network to predict the future states that the agent will encounter following its current policy, $s_{t+1}, ..., s_{t+T}$. We use the DAE g_ϕ to compute the aforementioned error on each of those states, and then average the result. For brevity, we will refer to the uncertainty of a state as $u_t = u(s_t, f_\theta, d_\gamma, g_\phi)$, where s_t is the current state and $f_\theta, d_\gamma, g_\phi$ are policy, dynamics and uncertainty networks, that are used as shown in Figure 3 (right, appendix). This value is used as a proxy for the epistemic uncertainty of the policy on the current state. We show empirically in Section 5.1 (appendix) how this method can accurately predict failures several steps before they happen, surpassing even a supervised model, allowing to collect informative demonstrations at training time and adding safety at test time.

3.4 Online Planning for State Shift Minimization

Here we demonstrate how an interplay of the networks we introduced in this work can be used, in addition to obtaining a more efficient active imitation learning method, to reduce state-distribution shifting at test time. (Ross et al. (2011); Pomerleau (1989); Laskey et al. (2017); Ghasemipour et al. (2020)) At test time the agent plans for a series of actions $(a_0, a_1, ..., a_T)$ that minimize the uncertainty of the predicted future state visited by applying these actions, $\hat{s}_{t+T+1}$, similarly to Boney et al. (2019); Di Palo and Valpola (2018), that applied uncertainty regularization to Model-Based Reinforcement Learning. In our work, we decompose the learning of a model and the learning of a policy, tackling the problem of imitation learning with no explicit reward. The uncertainty of this state is computed as described in Section 3.3. Hence, at each time step the robot computes its action as

$$a_t = f_\phi(s_t) + \beta \cdot a_{p,0} \text{ where } a_{p,0:T} = argmin_a \; u(\hat{s}_{t+T+1}) \text{ subject to } \hat{s}_{t+1} = d_\gamma(s_t, a_t)$$

where a_p is a trajectory of actions that minimizes the uncertainty of the final predicted state, T is the length of the plan, and $a_{p,0}$ is the first action of such trajectory, following the Model Predictive Control framework.

4 Experiments

In this section we describe the experiments we designed to benchmark the performance of SAFARI on several manipulation tasks, both simulated and on a real robot. [1]:

Further discussion, ablation studies and a more detailed list of hyperparameters used in all the experiments can be found in the Appendix, where we also demonstrate the ability of our method to predict failure in an unsupervised way, and the impact of the dynamics model in reducing the number of steps needed.

Task / Method	**AL-PL**	**AL-ROP**	**AL-DART**
1-Cube Pick and Place	**88%**-12%	**80%**-20%	**80%**-20%
2-Cubes Stack	**83%**-17%	**73%**-27%	**67%**-33%
Nut and Peg	**75%**-25%	**63%**-37%	**71%**-29%

Table 1: Experimental results of Active Learning (AL) against several baselines as described in Section 4.1.1. We show the percentage of experiments, defined as a round of data collection, training and testing with a different random seed (Section 4.1), on which one method successfully completes more test instances of the task than the other.

4.1 Active Learning Performance at Test Time

We designed the following experiments to compare the final performance of the agent using Passive Learning, Active Learning, and two other baselines, reducing all possible influences of external factors.

N demonstrations are used to train a policy network, consisting of our Passive Learning baseline. For Active Learning, we select the initial $(1-\gamma)N$ (where $0 < \gamma < 1$) demonstrations, used as the initialization demonstrations, then collect additional γN demonstrations using our iterative method, described in section 3.1. We suppose that both $(1-\gamma)N$ and γN are integers. We then test the policy networks trained on these two sets of demonstrations on a common test set.

[1] Detailed hyperparameters, code and videos: `https://www.robot-learning.uk/safari`

We repeat the experiments, including data collection, training, and testing, on a minimum of 7 seeds. In Table 1 we show on what percentage of the seeds one method completed more test tasks than the other, while showing the normalized sum of the total solved test cases in Figure 2, right (appendix).

4.1.1 Simulated Environments - Cubes Manipulation and Nut and Peg Insertion

We start by describing our results on three simulated tasks. The three manipulation tasks are 1 Cube pick-and-place, 2 Cubes stacking, and a Nut and Peg insertion task (Figure 3, left, appendix). We compare our proposed active learning method against several baselines: Passive Learning (PL), randomly stopping during execution to ask for a demonstration (Rand. On Policy, ROP), and DART Laskey et al. (2017), an effective algorithm for robot imitation learning that tackles distribution shift. The results of Table 1 show how our method outperforms all the baselines considering several independent runs.

4.1.2 Real Sawyer Environment

To test the ability of our method to scale to real world scenarios, we designed a manipulation task using the Sawyer robot. Additional details can be found in the Appendix.

Active/Total Demos	**0/50 (PL)**	**25/50**	**40/50**
Sawyer - Obj. Manipulation w/ Vision	15/25	18/25	**20/25**

Table 2: Successfully completed tasks over the same test set of 25 tasks, varying the amount of active demonstrations. (PL being Passive Learning).

In these experiments, we provided a total of 50 demonstrations to the robot. For the active learning approach, we test to different ratios of active demonstrations, $\gamma = 0.5$ and $\gamma = 0.8$. As shown in Table 2, the active learning approach beats the passive learning approach in this environment as well. We also notice how a larger ratio of active demonstrations achieves the best results, as also happened in the experiments in Figure 2 (left, appendix). This validates our hypothesis that demonstrations collected using our method are generally more informative and useful for the policy network to generalize.

4.2 Test Performance of Hybrid Model-free and Online Planning

In this section we measure the influence of the planning method proposed in section 3.4 (Figure 1 - Figure 4, appendix) on the performance of the robot at test time.

We collected a series of expert demonstrations and trained the models described in this work, $f_\theta, g_\phi, d_\gamma$. We compare the performance of normal behavioural cloning using the policy network actions at test time, $a_t = f_\theta(s_t)$ against our hybrid acting approach (Section 3.4). As shown in Table 3, our proposed hybrid approach **outperforms behavioural cloning** on different environments.

Task / Method	**SAFARI (ours)**	**BC**
1-Cube Pick and Place	**8.8** $\pm$ 2.35	7.6 $\pm$ 2.35
2-Cubes Stack	**5.9** $\pm$ 4.0	3.4 $\pm$ 2.6

Table 3: Comparison of our proposed control method (Section 3.4) and behavioural cloning. Average number of solved test instances and standard deviation computed on 10 different seeds of 50 test cases each.

5 Conclusion

In this work we introduced SAFARI, a method that tackles some of the principal issues of Learning from Demonstrations in robotic scenarios. We demonstrated how to tackle these problems using a common, versatile approach, obtained with a combination of a policy network, a dynamics network and an uncertainty network.

References

T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

S. K. S. Ghasemipour, R. Zemel, and S. Gu. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277, 2020.

D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

N. Di Palo and H. Valpola. Improving model-based control and active exploration with reconstruction uncertainty optimization. *arXiv preprint arXiv:1812.03955*, 2018.

R. Boney, N. Di Palo, M. Berglund, A. Ilin, J. Kannala, A. Rasmus, and H. Valpola. Regularizing trajectory optimization with denoising autoencoders. In *Advances in Neural Information Processing Systems*, pages 2859–2869, 2019.

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

S. James, M. Bloesch, and A. J. Davison. Task-embedded control networks for few-shot imitation learning. *arXiv preprint arXiv:1810.03237*, 2018.

A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

M. Hanczor. *Improving Imitation Learning through Efficient Expert Querying*. PhD thesis, Carnegie Mellon University Pittsburgh, 2018.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

C. Cronrath, E. Jorge, J. Moberg, M. Jirstrand, and B. Lennartson. Bagger: A bayesian algorithm for safe and query-efficient imitation learning.

N. Di Palo and E. Johns. Active robot imitation learning with autoencoders and imagined rollouts. *NeurIPS 2019 Workshop on Robot Learning*, 2019.

A. Filos, P. Tigas, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? *arXiv preprint arXiv:2006.14911*, 2020.

C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. *arXiv preprint arXiv:1903.01973*, 2019.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

H. Arponen, M. Herranen, and H. Valpola. On the exact relationship between the denoising function and the data distribution. *arXiv preprint arXiv:1709.02797*, 2017.

M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Appendix

Algorithms

In this section of the Appendix we describe in more detail the proposed algorithms. Algorithm 1 presents the steps taken in the Active Learning procedure. We define *unc_rollout* as the function that computes the uncertainty using the Dynamics Network for predicting future states and their predicted uncertainty, describing it in Algorithm 2.

Algorithm 1 Active Learning with Uncertainty Networks - Autonomous Variant

Initialize demonstrations set $D = \{\}$, Policy, Dynamics and Uncertainty networks $f_\theta, d_\gamma, g_\phi$, total desired demos N, active learning demos ratio γ, steps-to-retrain μ.
 # Start collecting a fraction of the total desired demos in passive learning to train the networks.
for i in $N(1-\gamma)$ **do**
  Sample task instance.
  Collect demo in the form $d_i = (s_0, a_0, ..., s_T, a_T)$.
  $D \leftarrow D \cup d_i$
end for
Train $f_\theta, d_\gamma, g_\phi$ on D.
 # Collect demos with Active Learning. Retrain every μ steps, stop at N total demos.
for i in $\gamma N/\mu$ **do**
  for j in μ **do**
    Sample task instance.
    while Task is not finished/failed **do**
      Compute uncertainty $u_t = unc\_rollout(s_t, f_\theta, d_\gamma, g_\phi, steps)$, get action $a_t = f_\theta(s_t)$.
      if $u_t > u_{thr}$ **then**
        Stop execution, collect expert demo from s_t as $d_i = s_t, a_t, ..., s_T, a_T$.
        $D \leftarrow D \cup d_i$
        Break While.
      end if
      Execute action a_t, obtain observation s_{t+1}.
    end while
  end for
  Train $f_\theta, d_\gamma, g_\phi$ on D.
end for

Algorithm 2 Function *unc_rollout()*

INPUTS: current state s_i, Policy Network f_θ, Dynamics Network d_γ, Uncertainty Network g_ϕ, number of future steps $steps$.
Initialize $u_{tot} \leftarrow 0$.
for i in $steps$ **do**
  Predict uncertainty and add to total. $u_{tot} \leftarrow u_{tot} + g_\phi(s_i)$
  Predict action $a_i \leftarrow f_\theta(s_i)$.
  Predict next state $s_{i+1} \leftarrow s_i + d_\gamma(s_i, a_i)$
  $s_i \leftarrow s_{i+1}$
end for
Return $u_{tot}/steps$

Hyperparameters and details of experiments

In this section we describe the hyperparameters and architectures we used in our experiments. We used the OpenAI Gym Fetch environment (Brockman et al. (2016)) for our experiments, that was modified to add the 2 Cubes scenario, plus a simulated Sawyer Nut and Peg scenario from Yu et al. (2019). The 1 Cube scenario has an observation space of 31, describing positions, velocities and orientations of end-effector and objects and desired goal position, and an action space of 4,

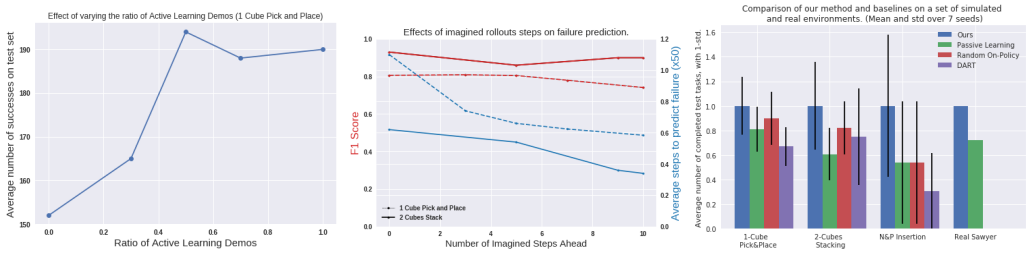

Figure 2: Left: How test-time performance varies as we modify the ratio of Active Learning examples over Passive Learning ones, γ. The plot represents the average number of successes over 13 different independent runs. Center: How failure prediction F1 score and steps needed to predict failures change when modifying the number of look ahead steps in imagined rollouts. Right: Normalized number of test instances solved by each method, averaged over 7 seeds.

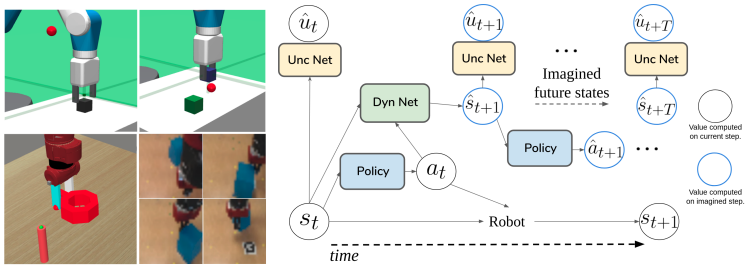

Figure 3: Left: The simulated and real environments used in this work. Right: This diagram depicts the method we designed to estimate the agent's uncertainty described in Section 3.3. At each time step, the agent predicts the future visited states by running its policy network using its internal world model. It then computes the epistemic uncertainty of these imagined state using its uncertainty model, averaging the results over all the imagined steps.

describing Cartesian velocities and gripper position. The 2 Cubes scenario has an observation space of 49 and same action space. The Nut and Peg scenario has an observation space of 9, describing positions of end-effector, object and goal, and an action space of 4, as with the previous scenarios.

For our **real world Sawyer experiments**, we designed a cube pushing task. The inputs are the vision stream given by a fixed camera, which is resized to 32x32, and the end-effector position given by the kinematics of the robot. We use an autoencoder to reduce the dimensionality of the input. We gathered around 10 minutes of random movements and interaction with the environment to quickly collect an image dataset. We then trained the autoencoder on this dataset and use it, frozen, in all our experiments. The encoder transforms the input images to vectors of size 16. Figure 5 (Appendix) describes the process of encoding the input images, concatenating the embedded vector with the state of the robot, and giving it as input to our networks.

5.1 Unsupervised Failure Prediction

In this section, we experimentally demonstrate the ability of our method to predict failures, while also minimizing the steps needed. We first compared our uncertainty network, trained in an unsupervised way only on expert demonstrations, against a **supervised failure predictor baseline**, trained on an ad-hoc dataset of successful and unsuccessful trajectories, to test the prediction abilities of our method. Then, we measured the number of steps needed to predict a failure, and the accuracy of these predictions, when changing the number of steps predicted with the imagination rollout. We demonstrate how our unsupervised method is able to surpass the supervised method, while never seeing an actual failure trajectory in the dataset. Furthermore, we show how our method is able to reduce considerably the number of steps needed to predict a failure by predicting the future, hence improving safety in robotics scenarios.

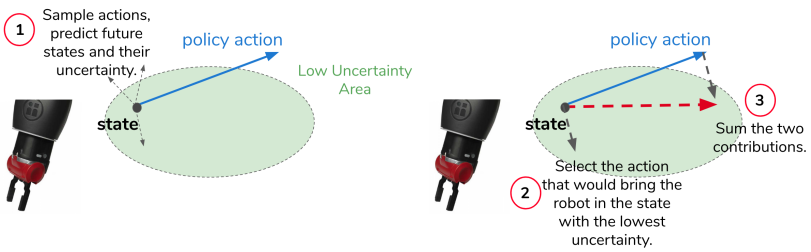

Figure 4: In this figure we visually describe the hybrid behavior of the robot that we propose. After computing the output of the policy network, the agent plans for a series of actions that will minimize state distribution shift as described in Section 3.4.

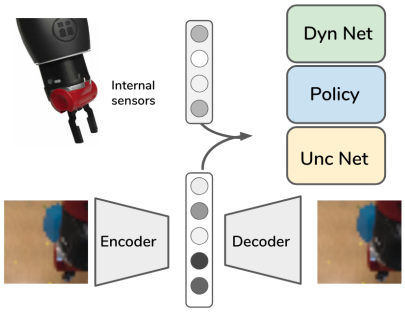

Figure 5: Embedding of the robot's visual input and concatenation with internal states, as described in Section 4.1.2.

For the first experiments, we used the same scenarios described in section 4.1.1. For the first experiments, we trained our networks on a set of 70 expert demonstrations. We then gathered a set of 70 trajectories by running the policy, labelling them as "successes" or "failures". We trained a binary classification neural network using this dataset, the failure predictor, to classify states s_t as success or failures, predicting the outcome of the current trajectory. We then gathered a test set of M test trajectories, run the robot's policy on those, and used both our uncertainty model and the failure predictor to measure the probability of failure from the current state at each step. When one of these outputs surpasses a threshold, the trajectory is labelled as a failure according to the respective model. We then let the agent complete the trajectory and labelled it as actual successes or failures. Finally, we measured the precision, recall and F1 scores of the two methods in predicting actual failures. Our method achieves **0.71 F1 score** averaged on 5 runs, while the supervised failure predictor achieves **0.66 F1 score** on the same test set. Hence, **our unsupervised method actually beats the supervised baseline**, while being trained only on expert's demonstrations to predict epistemic uncertainty.

For the second experiments, we trained our networks on a set of expert demonstrations, and we used the technique described in Section 3.3 to predict future states and anticipate possible failures. We measured the F1 score at predicting failures of our method, along with the number of steps needed, when changing the number of future states predicted. We empirically **demonstrate how the future prediction allows our method to predict failure in fewer steps without degrading performance**. As show in Figure 2 (center, appendix), the F1 score is roughly unchanged from 0 to 10 steps of future states prediction, while the steps needed to actually predict failures are much fewer, hence strongly improving safety.

In our experiments, we also compared the Denoising Autoencoder with Dropout-based Bayesian approximation Gal and Ghahramani (2016) and Random Network Distillation Burda et al. (2018), and the Denoising Autoencoders gave the best results. We plan to further explore additional architectures in future work.

Table 4: Architectures of Policy Network (PN), Denoising Autoencoder (DAE) and Dynamics Network (DN). We describe the hidden nodes' sizes (HS), number of hidden layers (HL).

Task name	**PN HS**	**PN HL**	**DAE HS**	**DAE HL**	**DN HS**	**DN HL**
1 Cube Pick-and-Place	128	2	8	2	128	4
2 Cubes Stacking	128	2	32	2	128	4
Nut and Peg	128	2	8	2	128	4

Table 5: Hyperparameters of Algorithm 1 on the various experiments reported in Table 1.

Experiment name	N **demos**	γ	μ	u_{thr}
1 Cube Aut. AL-PL	60	0.5	5	1.5 error on train set
2 Cubes Aut. AL-PL	300	0.5	25	1.1 error on train set
Nut and Peg AL-PL	40	0.75	5	3 error on train set

6 Robustness to changing the uncertainty threshold

In this section, we show the results of changing the uncertainty threshold u_{thr} that defines when the agent decides to stop to either ask for a demonstration or predict a failure. A lower value means that a lower uncertainty given by the uncertainty network is enough to stop the robot, and vice versa. We show that our method is robust to changes in this value. We show in Table 5 how the results are consistently well above the Passive Learning baselines and do not change much when modifying the u_{thr} hyperparameters, hence showing the robustness of our method.

Test performance with different values of u_{thr} during training.	Passive Learning	u_{thr} =4	u_{thr} =5	u_{thr} =6	u_{thr} =7	u_{thr} =8
Number of test cases solved	670	826	811	865	828	802

Table 6: Experimental results of different values of u_{thr} in the 1 Cube scenario, to test the robustness of our method to changing the threshold for epistemic uncertainty. As we show, results consistently surpass the Passive Learning baselines, while not changing much when modifying the hyperparameter. Results summed over 4 random seeds.