IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Life-long reinforcement learning in robotics for non-stationary environments

Author:
Tilman Hisarli

Supervisor:
Dr Antoine Cully

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Artificial Intelligence of Imperial College London

September 2022

Abstract

In order for a robot to act truly autonomously, it needs the capability to adapt to changes in its environment. Ideally, adaptation occurs fast, and without the need for the researcher to anticipate the future environments the robot may encounter. Within the field of Quality Diversity, a type of evolutionary algorithm that generates a diverse repertoire of high performing solutions for a given problem, research efforts for online adaptation have so far been focused on intelligently choosing solutions from pre-computed archives, in part due to the high number of evaluations that QD requires. However, with continuously changing environments, it may be that none of the pre-computed solutions are effective in the new environments. A recent QD innovation, Dynamics-Aware Quality Diversity (DAQD), provides the sample efficiency required for online repertoire learning through the use of dynamics models. While not specialised for changing environments, DAQD can be used here as it makes no assumption about the distribution of future environments and learns new skills continuously online. As such, it provides a strong baseline to outperform for any new alternative lifelong learning QD algorithm.

In this project we introduce LLQD, a sample efficient model-based QD implementation that produces specialised repertoires of behaviours online for each distinct environment the robot encounters. To do so, LLQD uses probabilistic dynamics models to detect previously seen or newly encountered environments based on the robot's recent state and action trajectories, without the need for anticipating the distribution of environments that may be encountered in the future. We show that LLQD outperforms DAQD with statistical significance with respect to the total number of solutions generated, the quality of those solutions, and their reliability for planning. As the correct identification of environments is of great importance for LLQD, we perform a number of ablation studies that examine the effect of the key hyperparameters on LLQD's ability to correctly classify environments. Using three distinct environments, we find that with the correctly tuned parameters, LLQD can correctly identify 90% of environments at the median in a random sequence of five environment changes. We also find no statistically significant evidence of a drop in performance in the stretch case of constantly changing environments. Finally, we demonstrate that LLQD shows some statistical evidence of improved sample efficiency over learning new environments from scratch, through the use of transfer learning when a new environmment is detected.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Antoine Cully for his invaluable insight, encouragement, commitment and kindness throughout this project. His extraordinary support and guidance have been crucial, went far beyond what I could have expected, and made this project an extremely rewarding experience.

I would also like to thank Dr. Aldo Faisal for his advice on the project, and equipping me with a fantastic toolkit of reinforcement learning techniques that benefited me greatly during my research.

In addition, I am very thankful to Bryan Lim for his expertise, his excellent work on DAQD, and availability to answer my questions.

I am indebted to Shikha Surana, for teaching me how to use containers, taking the time to provide genuine, in-depth help, and making the research period a lot more entertaining.

Finally, I am very grateful to my partner and family for supporting me throughout the degree, and allowing me the freedom to explore a new discipline in the pursuit of happiness.

Contents

Chapter 1

Introduction

In order to exhibit true autonomy, robots need to be able to learn continuously and adapt their behaviour when they encounter substantially new environments. For a walking robot, for instance, this would include adjusting its gait to variations in surface, slope, or to the robot itself (e.g. damage, change in payload, battery level).

Such adaptation, however, is a difficult proposition. This is because adjustment should occur fast, while at the same time be possible for any reasonable new environment, even if the latter features unexpected changes, or is very different to what has been encountered before. There appears to be a trade-off between speed of adaptation and the ability to adapt universally, as algorithms that achieve the former typically need to make restrictive assumptions about the distribution of environments that may be encountered in the future, while those with the theoretical capability to adapt universally, like traditional evolutionary algorithms, require a vast number of computations [1].

Quality Diversity optimization, a form of evolutionary algorithm that produces a repertoire of diverse and high-performing solutions, has so far been used to only partially fulfill these objectives. By intelligently switching to alternative solutions that are pre-computed in simulation, a robot can adapt its behaviour online to damage in minutes without the need for human interference or damage analysis [2] [3]. However, when environments drastically change, there is no guarantee that any previously computed solution is effective in a new environment.

Additionally, when humans encounter an unknown environment, we do not just choose from previously learnt skills. Instead, we use what we have learnt from previous environments as a baseline and innovate, by learning new skills that are specialised to the new environment. To this end, instead of having a single pre-computed repertoire available, it may be beneficial for the robot to learn online a variety of repertoires, each building on one another, but specialised to a substantially different environment. This is the approach to life-long learning that we will explore.

In this paper, building on Lim et al's Dynamics-Aware Quality-Diversity algorithm [4], a remarkably sample efficient model-based QD implementation that learns the environment's dynamics-interactions with the robot, we present Lifelong Quality Diversity (LLQD). LLQD allows a robot to continuously adapt to changes in the environment, by sequentially building a separate repertoire of behaviours for each substantially different environment the robot encounters. In essence, LLQD learns a probabilistic model per environment, that allows it to evaluate how likely a series of real-life observations are according to each model, and thus environment. Using this information, LLQD then decides whether the robot is currently in a previously encountered or substantially new environment. When a new environment is detected, it uses the best fitting existing model as a baseline for training the new model, to speed up the generation of a new archive of solutions specialised for the new environment.

As such, our key contributions to the QD literature are:

1. **Flexible online adaptation:** A sample-efficient continuous learning QD algorithm able to effectively adapt to changing environments, that does not presuppose similarity between environments, or require training on any environments in simulation first. The latter is key in lifelong learning, as in many applications we cannot anticipate the full distribution of environments the robot may encounter ahead of time.

2. **Transition likelihood estimation:** The use of probabilistic models to obtain the likelihood of a number of executed robot transitions coming from a given environment.

3. **Environment detection capability:** A mechanism to detect sufficiently significant changes in the environment dynamics, and assign the relevant interactions to a dynamics model for a previously encountered, or entirely new environment.

4. **Autonomous transfer learning:** A transfer learning approach that automatically uses the most relevant learnt model to adapt to the dynamics of a new environment faster and with fewer real-world evaluations. This improved sample efficiency is important in online learning, as the robot should be able to adapt to environment changes and acquire new skills fast.

In chapter 2, we first provide the reader with some background information on the evolution of Quality Diversity algorithms, featuring the rise of model-based QD in the search for improved sample efficiency, as well as the current key approaches to environment adaptation.

In chapter 3, we deep-dive into a recent key innovation in the QD literature, Dynamics-Aware Quality Diversity [4], which enabled the development of LLQD.

We discuss LLQD in detail in chapter 4, guiding the reader through its exact workings, and providing a thorough insight into its most innovative components, like environment detection and LLQD's approach to transfer learning.

In chapter 5, we discuss the results of our experiments that evaluate LLQD's performance relative to DAQD for changing environments, as well as the effect of the key hyperparameters on LLQD's ability to detect environments correctly.

Finally, we debate LLQD's relevant ethical considerations in chapter 6, and conclude with a look onto future work in chapter 7.

Chapter 2

Background

2.1 Quality diversity

Quality-Diversity (QD) algorithms are a type of evolutionary algorithm that seek to generate a diverse set of high-quality solutions [5]. This stands in contrast to traditional optimization, where we typically search for a global single best solution to a given problem. Having a variety of high quality solutions to hand is particularly useful in the field of robotics, where optimization is often non-convex and done in simulation. Simulator dynamics can often marginally differ from the dynamics of the real environment. In this case, a solution obtained in simulation may not effectively transfer to reality, and having a diverse set of ”locally optimised repertoires of behaviours” [6] at our disposal can provide us with effective alternative solutions [7].

2.1.1 Measuring quality

The quality of a solution is typically evaluated through some fitness function, that scores how good a solution is at performing a set task (e.g. covering distance fast, or the final orientation of the robot). For two solutions that are ”similar”, QD optimizes by keeping the one with higher quality. The similarity between solutions is measured using their behavioural descriptor, which is the subject of the following section.

2.1.2 Measuring diversity

In QD, every potential solution has an associated behavioural descriptor (BD), which can be thought of as a low-dimensional description of the solution’s key features (e.g. for a uni-directional walking robot how long each leg is in contact with the floor [2]). By comparing their BDs, we can determine how novel or diverse a solution is relative to others. Note, that the BD space can be either hand-crafted as in [2], or learnt autonomously [8] [9] using some form of dimensionality reduction (e.g. PCA or deep auto-encoders).

Much emphasis is placed on correctly defining the BD space, as it significantly affects the solutions the QD algorithm ultimately finds [10]. An elegant way to approach this problem is to learn the BD-space, as in AURORA [8], where the sensory behavioural data (e.g. trajectories) associated with the stored solutions in the archive is used to find the low-dimensional latent space that defines the BD at each iteration. This enables the robot to ”autonomously discover the range of their capabilities” [8] and result in highly diverse solutions. However, when prior knowledge about the robot and specific task at hand exists, hand-crafting the BD space can be very effective. The subject of this paper, for instance is a hexapod robot that we desire to walk in every direction in the xy-space, and as such we define the BD associated with each solution as the robot's final position in the xy-space after executing the solution.

2.1.3 MAP-Elites

One of the seminal and yet simplest algorithms in the QD literature is the MAP-Elites algorithm [11], some of whose concepts are applied in this research project. In MAP-Elites, after defining some fitness function that measures how well a given solution performs a set task (i.e. measuring quality), the researcher defines ”N dimensions of variation of interest”, that describe the key features of a solution (i.e. the behavioural descriptor used to measure diversity). Each of the N dimensions of this low-dimensional space is then discretized, either by hand or automatically, to form an N dimensional grid, in which solutions are stored according to their key feature values.

After randomly generating a set of solutions and evaluating their behavioural descriptor and quality, MAP-Elites initialises the grid by storing in each of its cells the solution with the highest quality among those that have the qualifying behavioural descriptor. The algorithm then enters an iterative loop, where (1) a cell in the grid is randomly selected, (2) its solution is mutated into an ”offspring”, (3) the offspring is evaluated according to the fitness function and its low-dimensional behavioural descriptor determined, and (4) the offspring is added to the grid if the relevant cell is empty, or if the offspring's fitness is greater than the fitness of the solution currently occupying that cell. Once some stopping condition is reached (e.g. based on the total number of evaluations), MAP-Elites can present a grid-based archive of diverse yet high-quality solutions, and may even find a better global solution than traditional search algorithms as it tends to perform wider explorations of the search space [11].

There exists a great variety of QD implementations, that in part developed out of MAP-Elites. Despite differences in how solutions are stored, selected and evolved, they tend to share a common core that allowed Cully and Demiris to unify QD algorithms into one modular framework [7].

2.1.4 Generalized QD framework

According to this framework, the first main feature of QD algorithms is a container, which stores the highest performing and most diverse solutions in an ordered collection. This may be in the form of a discretized n-dimensional grid-based container as in MAP-Elites [12], or a more flexible archive whose structure emerges autonomously, as in this project or Novelty Search with Local Competition, which features an ”unstructured archive based on the Euclidean distance between solution descriptors” [12]. Some addition condition, which can be container-specific and is typically based on the quality and novelty of a solution relative to others already in the container, determines which solutions are added to the container.

The second key feature is a selection operator, which chooses the solutions that will be changed and evaluated in the following iteration of the algorithm. For instance, we may apply a uniform random selection rule to existing solutions in the collection, as in MAP-Elites or this project, or use score-proportionate selection (e.g. according to fitness, or novelty) to keep selection pressure high [7]. The choice of selection operator has a significant impact on the final set of solutions, in part by implicitly calibrating the degree of exploitation versus exploration in the BD space.

Regardless of the exact specification of the container and selection operator, QD algorithms repeatedly execute a process of (1) using the selection operator to choose parent solutions that will be mutated into candidate solutions (either from the archive of stored solutions or a separate population), (2) recording the BD of the candidate solutions and evaluating their performance, and (3) attempting to add the relevant candidate solutions to the container if they satisfy certain addition conditions of performance and novelty versus existing solutions in the container [7]. As discussed in the previous section, in MAP-Elites [11], for example, a solution x is stored in the relevant cell of the n-dimensional grid-based container if either the cell is currently empty (i.e. no previous solution with the same BD exists) or the fitness of x is higher than for the existing solution stored in the cell.

Unfortunately, in order to find an archive of diverse and high-fitness solutions, standard QD algorithms typically require tens of millions of evaluations [7]. The latter, however, can be computationally expensive to perform [13], time-intensive, and dangerous, for instance if the robot is damaged while evaluating a novel candidate solution. This is prohibitive for on-the-go online learning, where robots need to adapt to a change in environmental circumstances (e.g. damage to the robot or a change in floor friction) in minutes or less.

2.2 Archives as repertoires

One powerful approach to allow QD-based robots to adapt fast during deployment is to pre-compute the solution container in simulation (where resets are easy to implement and evaluations non-dangerous) and then use this archive as a behavioural repertoire, from which to intelligently find a solution online [2] [3]. While the real environment or state of the robot may not exactly match the simulation, the diversity of solutions available to the robot means we can often find a solution that works.

The seminal paper in such repertoire-based learning is Cully et al's *"Robots that can adapt like animals"* [2]. Here the authors use MAP-Elites to discover in simulation a diverse set of solutions that allow a multi-legged robot to move in a straight line as fast as possible. The robot is then damaged and tasked to find in a timely manner an effective solution from the archive that holds information about each solution's BD and predicted (simulated) performance, using an intelligent trial-and-error algorithm (IT&E). Intuitively, IT&E first assigns high uncertainty to all solutions in the archive, as they have not yet been tested in real-life. It then performs an informed selection of a solution to evaluate in reality, records its true performance, lowers its uncertainty, and updates the expected performance and uncertainty of nearby solutions (close proximity in the BD-space). Technically, this is performed via Bayesian optimization using a Gaussian Process that models the difference between the real-life performance of a solution versus the predicted performance from the behaviour-performance map. Gaussian processes are useful here, as they model the distributions across the domain of a function, thus yielding information about each points expected value and standard deviation, i.e. the expected performance and uncertainty, which we require in IT&E. In order to select the next data point to be evaluated, the authors use an Upper Confidence Bound (UCB) [14] acquisition function, which returns the solution with the highest weighted sum of expected performance (mean) and uncertainty (standard deviation) according to some weighting parameter. By varying the weighting parameter, the authors can elegantly adapt the degree of exploration and exploitation when selecting the next solution. When the evaluated performance of a solution is greater than 90% of the best expected performance of any other solution in the archive, IT&E stops. Using this approach, a robot can effectively adapt to damage in less than two minutes, and with typically less than ten trials.

Building upon IT&E, the Reset-free Trial and Error (RTE) algorithm [3] extends selection from a repertoire of skills to navigation tasks. After creation of a solutions archive in a reset-based simulation environment using MAP-Elites, RTE learns the differences between the predicted and realised outcome of solutions from the archive using Gaussian Processes (GPs). The authors' key innovation is to then employ Monte Carlo Tree Search (MCTS) that takes into account the learnt uncertainties to plan and execute the next best action. The observed outcome in turn updates the GPs. Note, that during the QD process, the MAP-Elites BD is defined as coordinates in the xy-space. The robot thus learns an omni-directional repertoire, which in turn allows MCTS to be used to navigate around obstacles.

While repertoire-based learning with online exploitation allows robots to successfully adapt to changes such as mechanical damage in minutes [2] and perform navigation tasks online by intelligently choosing alternative solutions, this approach has one limitation when it comes to continuous learning: the entire behavioural repertoire is learnt once, offline, in a pre-determined simulation environment. This means that if the test-time environment changes very significantly to what was modelled in simulation, the behavioural repertoire may contain no effective solution that is appropriate for the substantially different environment. We suggest, that for continuous learning, it may thus be beneficial to learn a new solutions archive when encountering a substantially new environment (i.e. acquire new skills). To do so, we will rely on model-based QD.

2.3 Models in QD

2.3.1 Modelling fitness and BD

Originally, models in QD were used to reduce the computational cost of learning, by replacing some computationally expensive evaluation function with its surrogate model. As laid out in [15] and [16], two algorithms that apply surrogate modelling to QD are M-QD [16] and SAIL [13].

M-QD [16] features a model that directly predicts both the behavior and the quality of candidate solutions. It thus reduces the number of candidate solutions that have to be evaluated on the real system, by eliminating those candidates that are ”unlikely to find a novel or to improve a known behaviour” [16] beyond some user-specified minimum threshold of improvement. As such, the authors show that M-QD indeed achieves coverage of the BD-space with fewer evaluations than vanilla QD.

SAIL [13], on the other hand, uses the model to solely learn a less computationally intensive approximation of the true fitness evaluation function. To do so, SAIL uses MAP-Elites to find solutions for each bin that maximise a Gaussian Process based UCB function that models the expensive objective function. The result is an ”acquisition-map”, that for each bin contains the solution with the highest weighted sum of mean expected performance and uncertainty around it. Through iterative uniform drawing from the acquisition-map, evaluation on the true objective function and updating of the GP, SAIL thus produces a cheaper ”model of the objective function in high-fitness regions” [13] of the BD space.

By mapping from the parameter space directly to the reward, in SAIL and M-QD the models implicitly learn the dynamics of how the robot interacts with the environment. However, as the reward is task-specific, the learnt models also capture task-specific relationships. It is thus more difficult to use these models for other purposes, like detecting changes to the environment dynamics, as the model contains error with regards to both the environment and task.

2.3.2 Modelling environment dynamics

More recently, in Dynamics-Aware Quality-Diversity (DA-QD), Lim et al. [4] employed the use of models to learn the dynamics of the environment, and then perform QD’s archive generation in imagination. Here, instead of mapping the parameter space to the BD and performance, the authors learn an ensemble of probabilistic dynamics models that map the current state of the robot and a given action to the change between the current state and the next state, using a trajectories dataset collected during real evaluations of imagined solutions.

These models, specified by neural networks, are then called continuously within each QD iteration to perform *imagined* roll-outs of the candidate solutions and record their BD and performance. The best and most diverse solutions are then stored in an *imagined archive*. Finally, all solutions (or the least uncertain according to the dynamics-ensemble) in the imagined archive, which can be thought of as informed predictions of high-performing solutions, are evaluated on the real-system, and their transitions added to the transitions dataset. According to the addition conditions, imagined solutions are then added to the real archive, and the imagined archive synced.

Learning the environment dynamics in this way has been shown to outperform ordinary QD with respect to both diversity and performance with remarkable sample-efficiency at 20x fewer evaluations on the real system. This is important, as real evaluations are a bottleneck both in terms of safety (e.g. potentially damaging the robot) and computation, as they can rarely be parallelized in real life. Interestingly, as the model learns the environment dynamics, it can also be used for zero-shot and few-shot learning with as little as 100,000 imagined roll-outs, by using the dynamics model in imagination with altered reward functions, adapted to some new task [4].

Unlike any of its model-based predecessors, this approach is particularly well-suited to deal with changing environments online, as a change in environment dynamics will be reflected in the transition datapoints, on which the model is trained. For example, when quadrupling the payload of a robot, it stands to reason that the change in the position of its joints is significantly different on average when executing the same action, compared to its lighter variant. In theory, this should allow the robot to adapt by performing QD and rolling out solutions in imagination using the updated dynamics model.

However, when a robot encounters a new environment, it may be disadvantageous to mix the transitions from two distinct environments, as it likely results in finding solutions that are not optimal for either environment. This poses the question of how to better adapt to changing environments to avoid overwriting and forgetting solutions optimised for previous environments.

2.4 Environment adaptation

A significant portion of the existing literature on RL based walking robots adapting to changes in their environments can be categorised into two distinct approaches.

The first category of algorithms centers around the idea of adapting by intelligently choosing the most relevant solution from a pre-computed archive generated in a different environment, as previously discussed in more detail in section 2.2. IT&E [2] and RTE [3] are two of the seminal algorithms, and consequently gave rise to further evolutions, like Swarm Map-based Bayesian Optimisation (SMBO) [17], which extends IT&E to swarms of robots, or APROL [18], which extends RTE to choosing from not only one but multiple repertoires, pre-trained ”for many different situations”. Alternatively, in ”Environment Adaptation of Robot Morphology and Control” [19], rather than exclusively evolving the controls of the robot to derive an archive of solutions to choose from, the authors also allow the morphology of the robot to change in the offline training stage, which may be beneficial in different environments. The core idea, however, of choosing the most relevant solution from a number of previously evolved solutions is shared across all of these adaptations. As such, they are all exposed to the same risk: while it may be possible to learn how differently the pre-computed solutions will behave in the new environment, there is no guarantee that any of them will work effectively in a distinctly new environment that is far removed from the original environments, in which the pre-computed solutions were generated.

The second category of algorithms adapt to changes in the robot's environment by altering their walking policy directly. This is done by previously learning how to adapt the policy for different environments. One such example is the RMA algorithm [20], which allows a legged robot to adapt to environment changes within fractions of a second. It does so by learning a base policy, that given the robot's current state, previous action, and a low dimensional representation of the current environment z_t outputs the next action (desired joint position) to take. Alongside this, RMA trains an online adaptation module, that given the robot's recent state and action trajectories from interactions in the environment produces an estimate of the current low-level environment representation $\tilde{z}_t$. At deployment, the latter is then fed into the base policy. While highly effective when the encountered environment is within the training distribution of environments, this approach can encounter difficulties when the new environment falls outside the training distribution. In this case, the adaptation module is unlikely to produce a good estimate of the true low-dimensional environment representation. In addition, the base policy will be challenged when $\tilde{z}_t$ falls outside the distribution of z_t that the base policy has been trained on, as is likely the case for a very different environment. As such, the researcher would need to anticipate a wide range of possibly encountered environments, and RMA be pre-trained on these in simulation. Of course, this does not mean that all possible environments need to be included in simulation as the neural networks used for the base policy and adaptation module may generalise well for environment configurations inside of the training distribution. However, universal adaptation to all future environments cannot be guaranteed. Unfortunately, learning to adapt to such out-of-distribution environments during deployment is also not possible, as the true descriptor of the environment is only available in simulation; thus the adaptation module, that maps trajectories to the low level environment estimate $\tilde{z}_t$ cannot be trained online.

Another example of the second category are the model-based meta-RL approaches explored in [21]. Here, a dynamics model parameterized by θ predicts the next state, given the robot's current state and action. However, the model parameters themselves are updated (to θ') using a learnt update rule, that takes as inputs the most recent M steps (i.e. segments of the robot trajectories) taken by the robot in the environment. To learn the update rule, a meta-training step is required to learn the optimised weights that minimise the average negative log likelihood for the next K steps, given the information provided by the recent M steps. As a meta-learning algorithm, it assumes "that the previous meta-training tasks and the new meta-test tasks are drawn from the same task distribution" [21]. By extension, this means that for very different types of environment changes to those seen in meta-training, the parameter update rule may not generalise well.

As such, existing approaches to environment adaptation tend to rely on some degree of transferability between the original environment and the environments encountered in the future, or require the researcher to anticipate the distribution of potentially encountered environments. Our goal instead is to devise a lifelong online learning algorithm that does not rely on these conditions, but instead can adapt to any unanticipated change in environment by truly learning and acquiring new behaviours, rather than just adapting what has previously been learnt.

Chapter 3

Related work: DAQD

In an effort to allow robots to adapt to new environments, LLQD builds on Lim et al's Dynamics-Aware Quality-Diversity algorithm [4], which due to its improved sample-efficiency is well-suited for online learning. In addition, due to its continuous evolutionary and exploratory nature, it can learn new skills continuously, which is crucial for lifelong learning, rather than just learning to adapt existing skills.

Therefore we first provide a detailed explanation of DAQD in this chapter (as described in [4]), followed by an in-depth discussion of LLQD in the next chapter.

3.1 Overview

DAQD is a model-based QD algorithm that learns a dynamics model and skill repertoire for a stationary environment. The key innovations over vanilla QD implementations like MAP-Elites are the introduction of the dynamics model $\tilde{p}_\theta$, and the imagined archive $\tilde{\mathcal{A}}$ that allow for better sample efficiency [4].

3.2 Dynamics model $\tilde{p}_\theta$

The dynamics model $\tilde{p}_\theta(s_{t+1} - s_t | s_t, a_t)$ is a probabilistic ensemble, as proposed in [22], that learns a mapping from the current state and action of the robot, to the consequent change in state. More precisely, in order to capture the aleatoric uncertainty of the system, it learns to parameterize a normal distribution for each dimension of the change of next state vector $s_{t+1} - s_t$, given some current state s_t and action a_t, i.e. $\tilde{p}_\theta(s_{t+1} - s_t | s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \Sigma_\theta(s_t, a_t))$. As such, rather than predicting a point estimate for each dimension of the change in next state, we learn a mean and standard deviation and assume that the dimensions of the change in state vector are independent. In order to capture epistemic uncertainty, DAQD learns an ensemble of such probabilistic models [4].

DAQD uses the dynamics model to perform *imagined roll-outs* of candidate solutions (or controls — we shall use the two terms interchangeably), which define the actions taken at each time-step. By recursively calling the dynamics model at each time-step, we obtain a predicted sequence of states of the robot, from which we can derive the predicted behavioural descriptor and predicted fitness [4]. This allows us to make a prediction (according to the model) whether a given solution will be added to the archive (i.e. it is either novel, or fitter than an existing solution of a sufficiently similar behavioural descriptor).

To train the model, DAQD uses a replay buffer $\mathcal{B}$, which stores the state, action, and next state trajectories of all the solutions that were previously evaluated in the real environment. The latter occurs either at initialization, when randomly generated solutions are evaluated in the real environment, or when evaluating solutions that the model predicted will likely be added to the archive.

3.3 Imagined archive $\tilde{\mathcal{A}}$

In QD, the archive $\mathcal{A}$ is a store of solutions that are both diverse and high performing. When determining whether a candidate solution should be added to the archive, we thus need to assess whether it is sufficiently different from the existing solutions (based on its *real* behavioural descriptor), or better than a nearby solution (based on its *real* fitness). In DAQD, a candidate solution passes the addition condition to the archive if its Euclidean distance in the behavioural descriptor space to its nearest neighbor in the archive is greater than some exogenously determined threshold value. Alternatively, if the distance to its nearest neighbor is too close, a candidate solution can still be added if its distance to its second nearest neighbor in the archive is greater than the threshold value, and its fitness or novelty score exceeds its nearest neighbor [4]. The novelty score of candidate solution s is the mean Euclidean distance of its behavioural descriptor BD_s to its k nearest neighbors in the archive:

$$novelty(s) = \frac{1}{K}\sum_{k=1}^{K} ||BD_s - BD_k||_2$$

The imagined archive $\tilde{\mathcal{A}}$ in DAQD, on the other hand, is used to check whether a candidate solution should be added based on its *predicted* behavioural descriptor and *predicted* fitness (both derived using the dynamics model), when compared to solutions already in $\tilde{\mathcal{A}}$. Note, that $\tilde{\mathcal{A}}$ is synced to the real archive $\mathcal{A}$ at every iteration (see line 4, in Algorithm 1), so that a candidate solution will be evaluated both against the solutions existent in the real archive, and previously added model-predicted solutions. Solutions that are added to $\tilde{\mathcal{A}}$ are candidates with a ”good chance” of being successfully added to the real archive $\mathcal{A}$ at a later step in the algorithm (see lines 8-9 in Algorithm 1), assuming the dynamics model is an accurate representation of the real dynamics. Only these good candidates will later be evaluated on the real environment, to be checked for addition to $\mathcal{A}$. This is where DAQD's 20x improved sample efficiency over vanilla QD stems from [4].

3.4 Algorithm and pseudocode

The DAQD algorithm follows the following steps:

Algorithm 1 DAQD

```
1: Init:
       archive $\mathcal{A} = \emptyset$, dynamics model $\tilde{p}_\theta$, replay buffer $\mathcal{B} = \emptyset$, num_real_evals = 0

2: $\mathcal{A}, \mathcal{B}, \tilde{p}_\theta \leftarrow$ rand_init()    ▷ Init archive w/ rand. ctrls, eval. in real env., add to buffer, train model

3: while num_real_evals $<$ max_num_real_evals do
4:     $\tilde{\mathcal{A}}$ = copy($\mathcal{A}$)    ▷ sync imagined archive
5:     candidate_ctrls $\leftarrow$ select_and_mutate($\mathcal{A}$)    ▷ create candidates
6:     model_eval_ctrls $\leftarrow$ model_eval(candidate_ctrls, $\tilde{p}_\theta$)    ▷ model evaluate candidates
7:     succ_model_ctrls, $\tilde{\mathcal{A}} \leftarrow$ add_test(model_eval_ctrls, $\tilde{\mathcal{A}}$)    ▷ try to add cand. to $\tilde{\mathcal{A}}$, get succ. cand.
8:     real_eval_ctrls $\leftarrow$ real_eval(succ_model_ctrls)    ▷ evaluate cand. added to $\tilde{\mathcal{A}}$ in real env.
9:     succ_real_ctrls, $\mathcal{A} \leftarrow$ add_test(real_eval_ctrls, $\mathcal{A}$)    ▷ try to add cand. to $\mathcal{A}$, get succ. cand.
10:    $\mathcal{B} \leftarrow$ add_to_buffer(real_eval_ctrls.traj, $\mathcal{B}$)    ▷ add traj. of real eval. cand. to buffer
11:    $\tilde{p}_\theta \leftarrow$ train_from_buffer($\tilde{p}_\theta$, $\mathcal{B}$)    ▷ train model with updated buffer
12:    num_real_evals $+=$ len(real_eval_ctrls)    ▷ update number of real evaluations performed
13: end while
```

After instantiating the empty real archive $\mathcal{A}$, replay buffer $\mathcal{B}$ and randomly initialised probabilitic ensemble $\tilde{p}_\theta$, DAQD creates a number of random genotypes (controls), evaluates them in the real environment, and retrieves their state and action trajectories, behavioural descriptor, and fitness. As the transitions (state, action, next state) of all evaluated genotypes carry information about the dynamics, it adds all transitions to the replay buffer $\mathcal{B}$. DAQD assesses each genotype in order for addition to the real archive $\mathcal{A}$, and sequentially adds those that qualify under the addition conditions set out in section 3.3.

As previously discussed, DAQD then instantiates the model archive $\tilde{\mathcal{A}}$ by syncing it with the real archive $\mathcal{A}$. It generates new candidate genotypes, by uniformly randomly selecting pairs of genotypes s_1 and s_2 from the real archive, and applying directional variation to each pair according to: $s_{new} = s_1 + \sigma_1 \mathcal{N}(0, \mathbf{I}) + \sigma_2 \mathcal{N}(0, \mathbf{I})(s_2 - s_1)$ [4]. Here, the new genotype is obtained by adding scaled Gaussian noise $\sigma_1 \mathcal{N}(0, \mathbf{I})$ to s_1 and shifting the resulting vector along the line from s_1 to s_2 [4]. The candidate genotypes are then evaluated by the dynamics model, and their attributes of interest, like predicted behavioural descriptor, fitness, and state-action trajectories stored. Rather than evaluating all candidate genotypes on the real system, DAQD now filters the most promising candidate genotypes according to the model by checking if they adhere to the addition conditions for the imaginary archive $\tilde{\mathcal{A}}$. These most promising candidates are then evaluated in the real environment, updating their behavioural descriptor, fitness and state-action trajectories to the real values (as opposed to model predictions), and are finally checked for addition to the real archive $\mathcal{A}$. The state-action trajectories of all candidates evaluated on the real environment are then added to the replay buffer $\mathcal{B}$, and the model trained.

With a clear understanding of DAQD, we can now discuss LLQD, which leverages some of DAQD's key innovations for improved sample efficiency, but specializes on non-stationary environments.

Chapter 4

Method: LLQD

4.1 Overview

The goal of this research paper is to devise an algorithm for robot locomotion that can generate both diverse and high quality gaits online in changing environments that a robot may encounter throughout its life, without the need to first train the robot in simulation on some distribution of possible environments. To do so, we operate within the sphere of *Quality Diversity* algorithms, and as such name our algorithm *"Lifelong Quality Diversity"* (LLQD).

As explained in section 2.1, QD algorithms require the evaluation of solutions with regards to their fitness ("quality") and behavioural descriptor ("diversity") on the real environment. Online evaluations on the real environment, however, are expensive, as they must typically be performed sequentially, and in the process may cause the robot to fail (e.g. damage). In order to reduce the number of real evaluations required, LLQD is a model-based QD implementation. The latter typically allow us to benefit from better sample-efficiency over their model-free alternatives (see section 2.3), and can thus discover solution archives of similar coverage and QD score with up to 20x fewer real evaluations, as in the case of DAQD [4]. As the dynamics of the physical interactions between a robot and its environment have previously been used to successfully adapt to changing environments [20], and Lim et al's DAQD [4] learns such a dynamics model, LLQD leverages the dynamics model training component and interaction with the imaginary archive from DAQD.

A visual representation of LLQD is provided in Figure 4.1. More detail on the exact workings of the algorithm is provided in Algorithm 2 and sections 4.3 to 4.6.

4.2 Algorithm and pseudocode

The main idea of LLQD is to detect the environment the robot is currently in from the robot trajectories, sequentially create new dynamics models and archives as new environments are visited, and assign the trajectories to the relevant model's replay

Algorithm 2 LLQD

```
 1: Define LLQD class $\mathcal{L}$, which contains
 2:     self.archive $\mathcal{A} = \emptyset$
 3:     self.imagined_archive $\tilde{\mathcal{A}} = \emptyset$
 4:     self.dynamics_model $\tilde{p}_\theta$
 5:     self.replay buffer $\mathcal{B} = \emptyset$
 6:     self.num_real_evals = 0

 7:     self.compute_real_eval_ctrls():
 8:         self.$\tilde{\mathcal{A}}$ = copy($\mathcal{A}$)    ▷ sync imagined archive
 9:         candidate_ctrls $\leftarrow$ select_and_mutate($\mathcal{A}$)    ▷ create candidates
10:         model_eval_ctrls $\leftarrow$ model_eval(candidate_ctrls, $\tilde{p}_\theta$)    ▷ model evaluate candidates
11:         succ_model_ctrls, $\tilde{\mathcal{A}} \leftarrow$ add_test(model_eval_ctrls, $\tilde{\mathcal{A}}$)    ▷ try add cand. to $\tilde{\mathcal{A}}$, get succ. cand.
12:         self.real_eval_ctrls $\leftarrow$ real_eval(succ_model_ctrls)    ▷ evaluate cand. added to $\tilde{\mathcal{A}}$ in real env.
13:         return self.real_eval_ctrls    ▷ return real eval. candidates

14:     self.update_buffer_and_train(real_eval_ctrls):
15:         succ_real_ctrls, $\mathcal{A} \leftarrow$ add_test(real_eval_ctrls, self.$\mathcal{A}$)    ▷ try add cand. to $\mathcal{A}$, get succ. cand.
16:         self.$\mathcal{B} \leftarrow$ add_to_buffer(real_eval_ctrls.traj, self.$\mathcal{B}$)    ▷ add traj. of real eval. cand. to $\mathcal{B}$
17:         self.$\tilde{p}_\theta$, $\leftarrow$ train_from_buffer(self.$\tilde{p}_\theta$, self.$\mathcal{B}$)    ▷ train model
18:         self.test_lhood_mean, self.test_lhood_std $\leftarrow$ get_test_set_likelihood_stats(self.$\tilde{p}_\theta$, self.$\mathcal{B}$)
19:         self.num_real_evals += len(real_eval_ctrls)    ▷ update number of real evaluations performed

20: env_list = get_rand_env_sequence()    ▷ generate random sequence of environments to visit
21: llqd_list = []    ▷ list to store $\mathcal{L}$ objects

22: switch_env(env_list)    ▷ set real env to next env from random sequence; updates simulator
23: llqd = init_new_llqd_instance()    ▷ instantiate first $\mathcal{L}$ object
24: real_eval_ctrls = llqd.compute_real_eval_ctrls()    ▷ real eval. selection of candidate ctrls
25: llqd.update_buffer_and_train(real_eval_ctrls)    ▷ first batch of ctrls always added to first $\mathcal{B}$
26: likeliest_llqd = llqd    ▷ first $\mathcal{L}$ object is always the likeliest object for first env
27: llqd_list.append(likeliest_llqd)    ▷ add first $\mathcal{L}$ object to storage list
28: **while** not reached end of env_list **do**
29:     switch_env(env_list)    ▷ set real env to next env from random sequence; updates simulator
30:     real_eval_ctrls = likeliest_llqd.compute_real_eval_ctrls() ▷ as bef.; next, get lhood of ctrls traj.
31:     ctrls_lhoods = [evaluate_likelihood(real_eval_ctrls.traj, llqd.$\tilde{p}_\theta$) for llqd in llqd_list]
32:     test_lhood_means = [llqd.test_lhood_mean for llqd in llqd_list] ▷ get means of test set lhoods
33:     test_lhood_stds = [llqd.test_lhood_std for llqd in llqd_list]    ▷ get stds of test set lhoods
34:     num_llqds = len(llqd_list)    ▷ get number of existing $\mathcal{L}$ objects
35:     transfer_candidate_idx = argmax(ctrls_lhoods)    ▷ get index of most likely $\mathcal{L}$ object
36:     **for** i in range num_llqds **do**
37:         mx_id = argmax(ctrls_lhoods)    ▷ index for most likely $\mathcal{L}$ object
38:         **if** ctrls_lhoods[mx_id] > (test_lhood_means[mx_id] - EPS * test_lhood_stds[mx_id]): **then**
39:             likeliest_llqd = llqd_list[mx_id]
40:             **break**    ▷ if lhood of ctrls traj. close to mean lhood of $\mathcal{L}$'s test set, $\mathcal{L}$ ID-ed as relevant
41:         **else**    ▷ lhoods of ctrls traj. too low rel. to test set lhoods for likeliest $\mathcal{L}$, try next $\mathcal{L}$
42:             del ctrls_lhoods[mx_id]
43:             del test_lhood_means[mx_id]
44:             del test_lhood_stds[mx_id]
45:         **end if**
46:     **end for**
47:     **if** len(ctrls_lhoods) == 0 **then**    ▷ if no existing $\mathcal{L}$ suitable, make new $\mathcal{L}$ for observed traj.
48:         likeliest_llqd = init_new_llqd_instance()
49:         likeliest_llqd.$\tilde{p}_\theta \leftarrow$ copy_params(llqd_list[transfer_candidate_idx].$\tilde{p}_\theta$)    ▷ transfer learning
```

```
50:         llqd_list.append(likeliest_llqd)                  ▷ add new $\mathcal{L}$ to list of $\mathcal{L}$ objects
51:     end if
52:     likeliest_llqd.update_buffer_and_train(real_eval_ctrls)  ▷ add ctrls traj. to likeliest $\mathcal{L}$'s $\mathcal{B}$, train
53: end while
```

buffer for training.

As outlined in Algorithm 2, the model, buffer, and archive for a newly detected environment is stored in an $LLQD$ class object $\mathcal{L}$ (lines 1-6; future line references in this section are with respect to Algorithm 2). Ideally, LLQD should sequentially create a one-to-one mapping from each distinct environment, to its associated $\mathcal{L}$ object.

The $\mathcal{L}$ class has two main methods. The `compute_real_eval_ctrls()` method (see lines 7-13) performs the selection and mutation of controls (i.e. solutions) in the archive, model evaluates them, and attempts to add them to the model archive to obtain a shortlist of candidate controls that have a higher change of being added to the real archive. It repeats this process until it has generated some exogenously determined number of model evaluated candidates, that is proportional to the number of steps the robot intends to make in the current environment. Finally, it performs the real evaluations of this shortlist of candidate controls.

The `update_buffer_and_train()` method (lines 14-19) takes in a set of candidate controls that have been evaluated in the real environment, and attempts to add them to $\mathcal{L}$'s real archive. This is performed in a separate method, as prior to archive addition, we need to ensure that the correct $\mathcal{L}$ object has been identified for the current environment. Next, the state and action trajectories from the real evaluations are added to $\mathcal{L}$'s buffer, and the dynamics model is trained. After training convergence, we record the mean and standard deviation of the $log\_likelihoods$ of the transitions in the holdout set (for details, see section 4.4), which are later used in the environment detection mechanism.

After generating a random sequence of environments that the simulator will visit (line 20), and loading the first (line 22), we create the first $\mathcal{L}$ object (line 23), that will store the relevant information from the first environment. Naturally, no environment detection needs to occur here, as only one environment has so far been visited, and as such this will always map to the first $\mathcal{L}$ object. As such, we call both of $\mathcal{L}$'s class methods sequentially (lines 24-25), thus creating and evaluating strong candidate controls in the real environment, adding their trajectories to the first $\mathcal{L}$'s buffer, training its model and generating the holdout set likelihood statistics. As no other $LLQD$ object exists yet, the first $\mathcal{L}$ is the "most likely" object for the given trajectories (see line 26). The object is then appended to the list of $\mathcal{L}$ objects (line 21, 27), that keeps track of all previously created $LLQD$ objects.

The algorithm then enters a loop, that sequentially visits all the environments that were randomly sequenced previously. After loading the new environment in the simulator (line 29), we try to create new candidate solutions to add to the current most likely $LLQD$ object's archive. After evaluating these controls on the new real environment, we then compute the mean $log\_likelihood$ of their transitions for all $\mathcal{L}$ objects that were previously created (line 31; see section 4.4 for more detail). We also retrieve each $\mathcal{L}$'s holdout set likelihood statistics (lines 32-33). We then check, if the transitions are sufficiently likely (lines 36-46; see detailed explanation in section 4.5) under any of the previous $\mathcal{L}$ objects, and if so, assign this $\mathcal{L}$ as the "most likely" for the current environment. If no existing $\mathcal{L}$ object qualifies, we create a new $\mathcal{L}$ instance, and assign this the "most likely" object label (line 48). We instantiate the new $\mathcal{L}$'s dynamics model with the weights and biases of the model from the other $LLQD$ objects that produced the highest mean $log\_likelihood$ for the transitions (lines 35, 49; more detail in section 4.6), and add the new $\mathcal{L}$ to the $LLQD$ list. Finally, we add the transitions to the "most likely" $LLQD$ object's buffer, update its model, and generate its holdout likelihood statistics.

It is worth noting, that in Algorithm 2, the pseudocode includes both the inherent workings of LLQD, as well as the handling of the environment switches in the simulator, as performed in our experiments. For instance, to simulate changes in the environment, we create a list containing a randomized order of environments to visit (Algorithm 2, line 20), and our stopping condition for the algorithm is met when all environments in the list have been visited (Algorithm 2, line 28). The exact implementation of the environment switches is, of course, not integral to LLQD.

4.3 Environment information

A key component of LLQD is the correct identification of the environment the robot is currently in. As we desire our robot to be able to adapt to any environment it encounters online, including environments that are significantly different to one other, we choose to not take an approach to environment detection that requires us to pre-train on some distribution of potentially encountered environments. Moreover, we do not provide any extrinsic information about these environments (e.g. friction, gravity) to the LLQD algorithm. Instead, LLQD utilizes one of the key insights from [20] that "when we command a certain movement of the robot joints, the actual movement differs from [the one commanded] in a way that depends on the extrinsics" of the environment. For us, this means that given some state of the robot and action, the robot's next state should depend on and reflect the current environment's dynamics. Note that state, action and next state are stored in the robot's walking trajectories.

Without access to extrinsic information about the environment, environment detection in LLQD is limited to either identifying the robot to be in a previously encountered environment, or in an environment that is sufficiently different to all previously explored environments such that it is deemed "new". We therefore define the term "environment detection" broadly, as the correct handling of the environment encountered, which includes both detecting a previously seen environment or identifying an environment as new. As such, LLQD sequentially builds a memory of environments and their associated dynamics models, replay buffers and archives. As previously explained, for each detected environment, an associated $LLQD$ object stores this information (see lines 1-6, Algorithm 2).

For our hexapod, the state vector has 48 dimensions. This vector consists of

- the position of the robot's centre of mass in the xyz-space (3),
- the rotation of the centre of mass in the xyz-space (3),
- the velocity of the centre of mass in the xyz-space (3),
- the rotational velocity of the centre of mass in the xyz-space (3),
- the joint positions (18), and
- the joint velocities (18)

Similarly, the action vector consists of the joint position commands sent to the robot (18).

As such, the state, action, next state tuple, which is used for detecting the environment, is 114-dimensional. This can be problematic, as reliable patterns in such high-dimensional data are less easily identified. We therefore make use of the probabilistic dynamics model used in DAQD specified in section 3.2, that predicts the distribution for the change in state (i.e. the difference between the next state and the current state), given a current state and action. This allows us to use only the change in state, which is 48-dimensional, for environment detection, reducing the dimensionality of the space by 58%.

4.4 Transition likelihood evaluation

When we perform an evaluation of a candidate control (which defines the gait) on the real environment, we obtain the associated action (i.e. sequence of actions) and state trajectories (i.e. sequence of states), from which we can derive the trajectory of state changes. Every real evaluation runs for three seconds, providing us with 300 state and action observations at the set control frequency. Per real evaluation, we therefore obtain 299 state, action, next state tuples.

In LLQD, we make use of the fact that the dynamics model $\tilde{p}_\theta(s_{t+1} - s_t | s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \Sigma_\theta(s_t, a_t))$ provides us with an estimate of the mean and variance of each dimension of the change in state $s_{t+1} - s_t$ given a current state s_t and action a_t, assuming independence of the state dimensions.

As the predicted distributions are normal, and the dimensions of the state are assumed independent, we evaluate the log-likelihood of each observed state, action, and change in state tuple (also called a transition), obtained from the robot trajectories, as

$$log\_likelihood = \sum_{i=1}^{48} \ln\left(\frac{1}{\sqrt{2\pi\sigma_i^2}} e^{\left(-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}\right)}\right),$$

where x_i denotes the observed change in state for the i-th dimension, μ_i the predicted mean change for the i-th dimension, and σ_i^2 the predicted variance for the i-th dimension change. This yields the likelihood of one robot step resulting in the observed change in state, according to the given dynamics model that specifies the mean and variance of the change in state.

By definition, the $log\_likelihood$ is negative; the closer to zero, the more likely a given observation is under the dynamics model in question. For a given number of observed transitions, and a number of dynamics models, LLQD then detects the most likely environment by computing under which model the observations are most likely, as measured by the mean $log\_likelihood$ across transitions.

4.5 Environment detection and assignment

Intuitively, LLQD's classification mechanism, based on a number of observed robot transitions in the environment, flags (a) if the robot has entered an environment with substantially different dynamics, or (b) a previously encountered environment. In the case of (a) the robot will learn a new dynamics model and behavioural repertoire in imagination, while in the case of (b) it will default to a previously learnt archive associated with the corresponding dynamics model.

To increase efficiency in relation to real evaluations, LLQD combines the real evaluation of newly mutated solution candidates (controls) used for archive generation with the robot's environment detection capacity. As such, we use the state and action trajectories of the real evaluations of candidate solutions as the transition data to determine the robot's current environment.

As previously mentioned, for environment detection we use the $log\_likelihoods$ of the robot's transitions to determine under which model (and thus $LLQD$ object) the observed state and action trajectories are most likely. Figures 4.2 to 4.4 show the mean $log\_likelihoods$ per model (pre-trained on environments 0, 1, and 2) across the 299 transitions per control for 200 randomly created controls, evaluated in environment 0, 1, and 2, respectively. We observe that while the mean $log\_likelihoods$ on average centre around a higher mean when the model matches the environment in which the controls have been evaluated (e.g. computing likelihoods of controls evaluated in environment 0 on the model pre-trained in environment 0), the mean $log\_likelihoods$ can take a wide range of values even in this matching scenario. This is likely due to the fact that when we randomly create controls (or mutate controls with a degree of randomness), the resulting candidate controls may explore parts of the dynamics model outside of the training distribution, and thus the model performs poorly. For example, Figure 4.4 shows the mean $log\_likelihoods$ for 200 random controls evaluated in environment 2 for all pre-trained models. We see that there is an overlap in the left tail of the distribution for model 2, and the right tails for models 0 and 1. As such, if we based our environment detection on only one candidate solution (i.e. one control), we would likely misidentify the environment relatively frequently (this is even more pronounced in Figure 4.2, where the overlap of likelihood distributions is larger). Instead, in LLQD we base environment detection on the mean of the $log\_likelihoods$ across all transitions from all newly evaluated candidate solutions, which provides for a more robust approach to environment detection.

In LLQD, after the evaluation of all candidate solutions on the robot's real environment (see line 12, Algorithm 2), we effectively rank the available models and their associated environments by their mean $log\_likelihoods$ for the executed candidate solutions (although note, that in the code we do not rank the likelihoods, but iterate through the models using the index of the $LLQD$ object that generates the highest mean $log\_likelihood$, see lines 36-37, Algorithm 2). As there will always be a model with highest likelihood, we do not naively identify the environment associated with the most likely $LLQD$ object as the robot's current environment. For instance, it may be the case that neither model fits the environment well, generating relatively low mean $log\_likelihoods$ across all available models. Instead, in order of highest likelihood, we compare the computed mean $log\_likelihood$ of the transitions of the newly evaluated controls (1) for the most likely model to the mean $log\_likelihood$ (2) and standard deviation of that model's holdout set obtained during training from previously assigned datapoints. If (1) is $epsilon$ standard deviations lower than (2) (see line 38, Algorithm 2), we deem the observed transitions from the evaluated controls to be too unlikely for the probed model, and evaluate the next most likely $LLQD$ object for identification. If (1) is within $epsilon$ standard deviations of (2), we detect the robot's current environment as the environment previously associated with this model (see line 39, Algorithm 2). If no model produces a mean $log\_likelihood$ within $epsilon$ standard deviations of (2), we detect a substantially new environment, create a new $LLQD$ object with a new model and replay buffer, and thus associate the current environment with this newly created $LLQD$-object (lines 47-52, Algorithm 2).

It is therefore clear, that the choice of parameter $epsilon$ plays an important role in environment detection, as a small value for $epsilon$ may run the risk of creating and associating multiple models with the same environment, while too large a value of $epsilon$ may wrongly assign datapoints from different environments to the same $LLQD$ object.

We also note the importance of comparing the mean $log\_likelihood$ of the newly evaluated controls against the mean holdout $log\_likelihood$ of the model, as opposed to some constant threshold. This is because some environments with less homogenous dynamics (e.g. walking on gravel as opposed to an even concrete floor) may inherently produce flatter distributions for the change in state, with larger estimated standard deviations per state dimension, which tends to result in a lower mean $log\_likelihood$.

4.6 Transfer learning

Finally, LLQD features a transfer learning mechanism that allows the robot to use information stored in previous environments' dynamics models as a basis for the newly created dynamics model, when a new environment is detected. We do this in an effort to improve the sample efficiency of LLQD. This is important, as LLQD is to be deployed online and we thus desire the robot to learn the new dynamics model and behavioural repertoire as quickly as possible.

In order to speed up the generation of the archive for a newly identified environment (and $LLQD$ object), we initialise the weights of the new dynamics model with those of the next most likely model (see lines 35, 49, Algorithm 2). This is because we expect the next most likely model to have learnt some environment dynamics that may be relevant in the new environment, as well. In other words, we expect some dynamics to be shared across environments, like the effect of gravity on the robot, and even more so, when the environments are relatively similar.

This should improve the sample efficiency of LLQD over DAQD, as we make use of the dynamics-related information of transition samples from previous environments, for the new environment. The relevant experimental evaluation is provided in the next chapter (section 5.3.5).

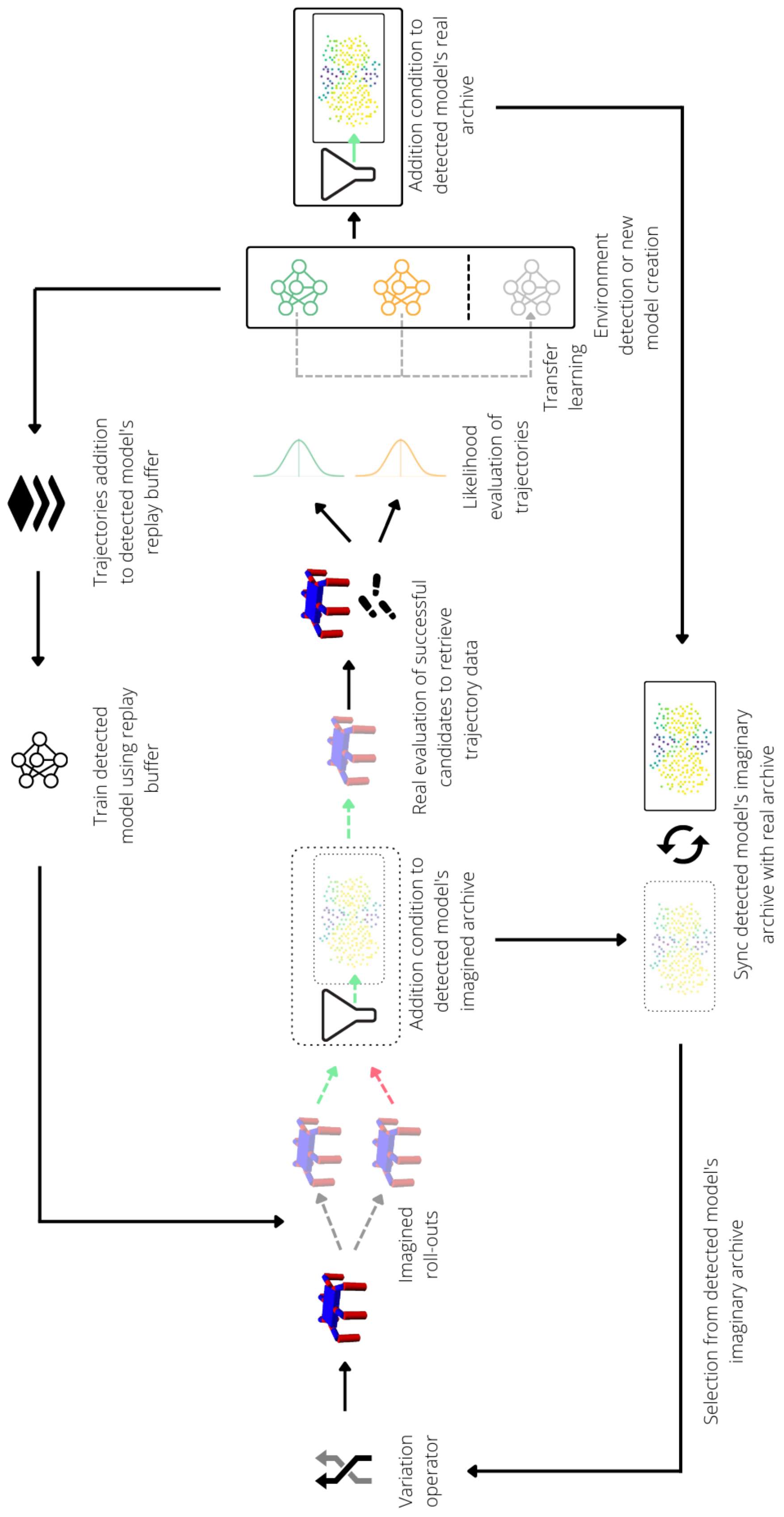
Variation operator
Imagined roll-outs
Addition condition to detected model's imagined archive
Real evaluation of successful candidates to retrieve trajectory data
Likelihood evaluation of trajectories
Transfer learning
Environment detection or new model creation
Addition condition to detected model's real archive
Trajectories addition to detected model's replay buffer
Train detected model using replay buffer
Sync detected model's imaginary archive with real archive
Selection from detected model's imaginary archive

Figure 4.1: Visual representation of LLQD algorithm. DAQD components from [4].

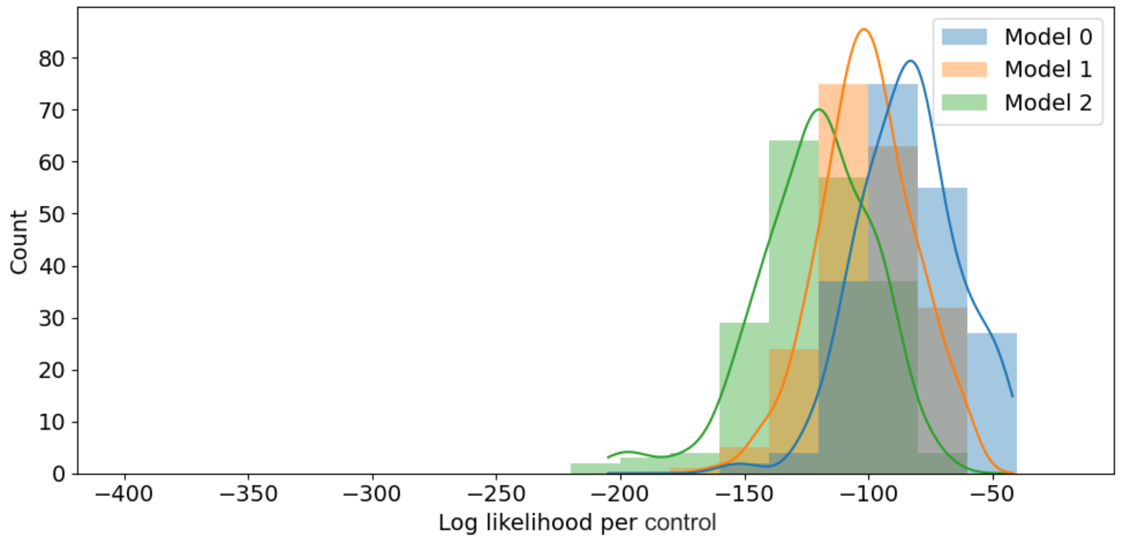

Figure 4.2: Mean sum of log likelihoods across state dimensions per model for 200 random controls executed in environment 0. Model 0 is pre-trained on environment 0, model 1 on environment 1, and model 2 on environment 2.

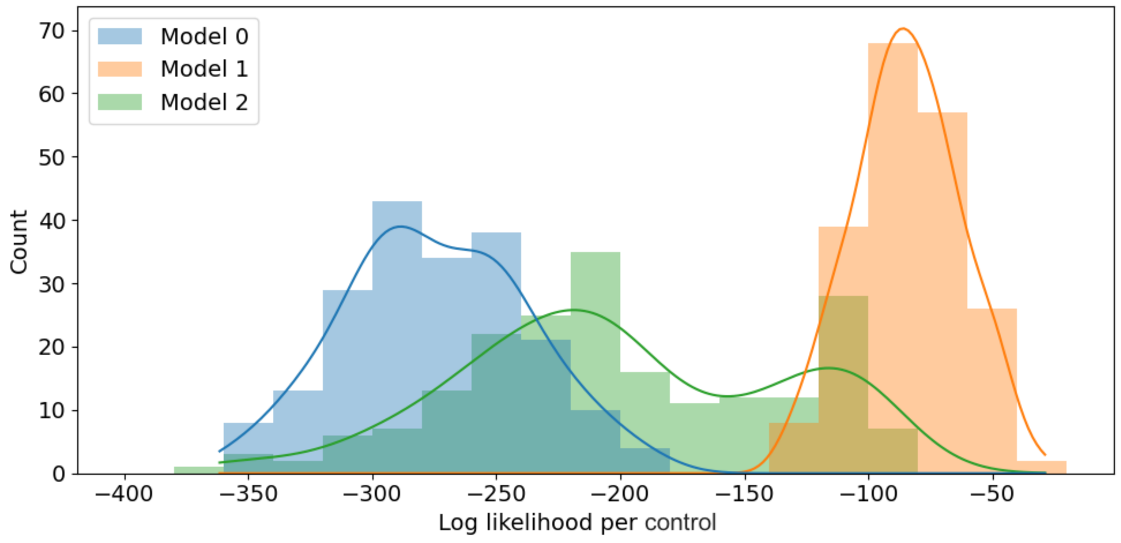

Figure 4.3: Mean sum of log likelihoods across state dimensions per model for 200 random controls executed in environment 1. Model 0 is pre-trained on environment 0, model 1 on environment 1, and model 2 on environment 2.

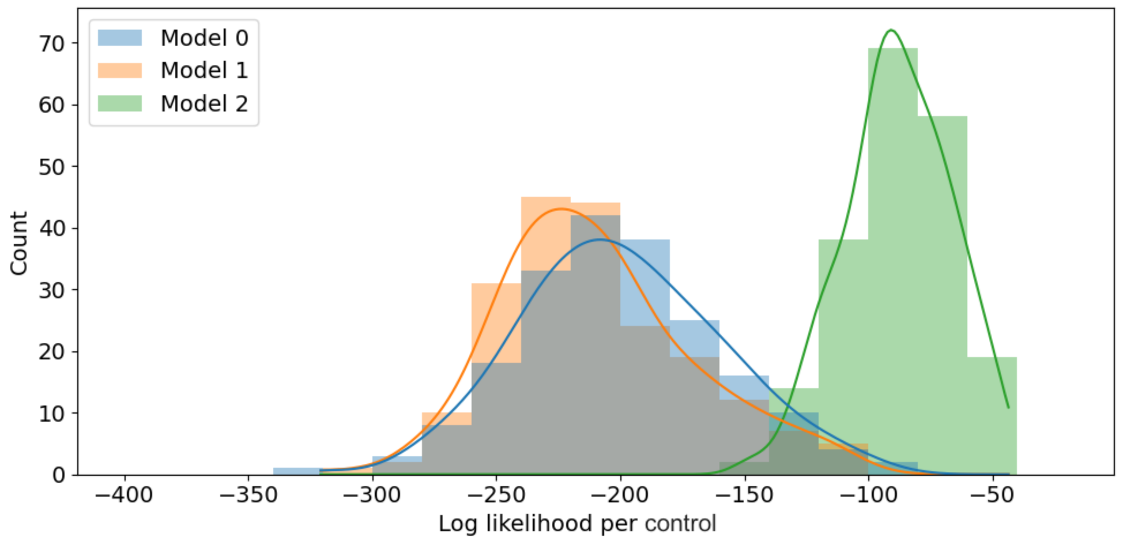

Figure 4.4: Mean sum of log likelihoods across state dimensions per model for 200 random controls executed in environment 2. Model 0 is pre-trained on environment 0, model 1 on environment 1, and model 2 on environment 2.

Chapter 5

Experiments & Evaluation

5.1 Overview

To evaluate LLQD for continuous learning we compare its performance to the current alternative of using DAQD with changing environments. We choose DAQD as the relevant comparison, as, like LLQD, it does not require any pre-training step on some distribution of environments, and can acquire new skills continuously with strong sample efficiency, important for online adaptation. As the correct detection of environments is key to LLQD's efficacy, our main experiment will be followed by a number of ablation studies that explore the effect of the $epsilon$ parameter and the number of steps taken in each environment on the percentage of environments that are correctly identified as either a specific one of the previously encountered environments, or a new unseen environment. We then examine LLQD's environment detection capability under the stretch scenario of constantly changing environments. Finally, we will investigate whether there is any evidence of LLQD generating higher quality or more diverse archives with fewer costly real evaluations than DAQD.

5.2 Experiment setup

5.2.1 Robot

For our experiments, we use the same 18 degrees-of-freedom (DOF; three DOFs per leg) simulated hexapod robot as in [4], depicted in Figure 5.1. For simulation we employ the RobotDART simulator, which is a wrapper "built on top of the DART [23] physics engine" [24].

As in [2], each DOF's angular position follows a periodic function defined by its amplitude, phase and duty cycle. There are three servos per leg, however, in order to keep the final segment of each leg vertical, the "control signal of the third servo is the opposite of the second one" [2]. As such, our choice of controller defines six parameters per leg (three each for servo 1 and servo 2), or 36 parameters in total, that produce the various different gaits.

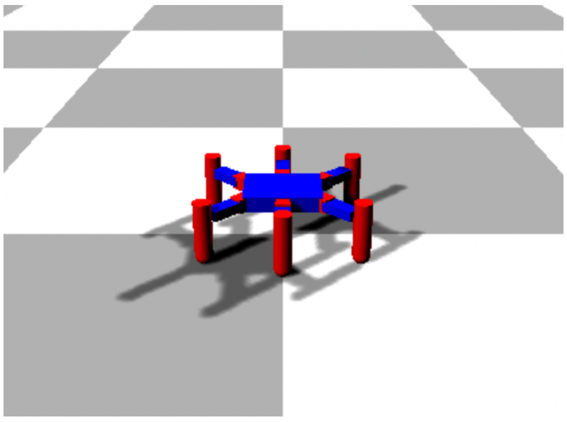

Figure 5.1: Hexapod robot in Robot-Dart [23] simulator, environment 0

5.2.2 Environments

We deploy the robot in three distinct simulated environments:

- **Environment 0**: the default environment, featuring an evenly balanced robot with no extra payload attached, on an even floor (see Figure 5.1).
- **Environment 1**: a robot with a heavy payload attached underneath that lowers and off-centers its center of mass, on a floor slanted in both the x and y dimension (see Figure 5.2).
- **Environment 2**: an evenly balanced robot with no extra payload attached, on a floor slanted in the opposite directions to environment 1 (see Figure 5.3).

In our experiments, we assume that the environment changes abruptly, rather than gradually morphing from one environment into the other.

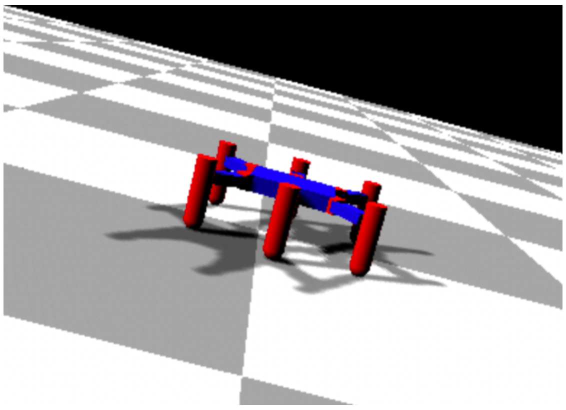

Figure 5.2: Hexapod robot in Robot-Dart [23] simulator, environment 1

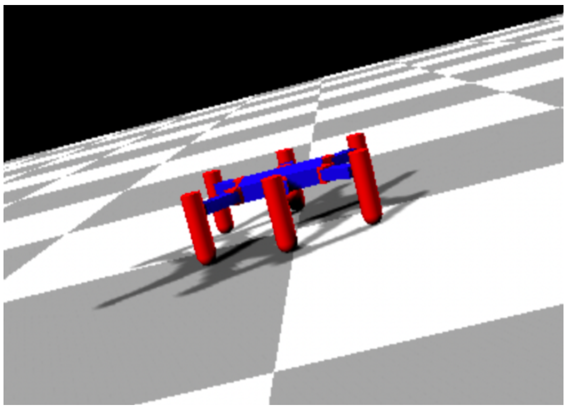

Figure 5.3: Hexapod robot in Robot-Dart [23] simulator, environment 2

5.2.3 Task

Our goal is for the robot to learn a variety of locomotion skills, that allow it to move in every direction in any environment it encounters. The BD records the final position of the robot in the xy-space after three seconds of simulation, and the novelty objective inherent to QD encourages the robot to explore unseen areas.

To accommodate the periodic nature of the controller and make sequential planning with the resulting repertoire easier, as in [25] we encourage the robot to learn behaviours along a circular trajectory, where the final orientation of the robot is aligned with its trajectory. This means that a higher fitness is associated with those solutions, that minimize the distance between the robot's orientation at its final position, and the tangent at the robot's final position to the desired circular trajectory [7].

5.3 Results and evaluation

5.3.1 LLQD vs DAQD with changing environments

In the following we show that LLQD can handle changing environments more effectively than DAQD by assessing the final archives generated under each algorithm for a given number of real evaluations. We choose the latter as the limiting factor, as real evaluations are costly (i.e. can result in damage, cannot be parallelized when deployed in real life, etc.).

Metrics and benchmark

We evaluate the archives generated with regard to three metrics that we define as follows:

1. **Coverage:** The number of solutions in the archive, which is an indicator of the diversity of solutions.
2. **QD score:** The sum of the normalized fitness values of the solutions stored in the archive, which is an indicator of the quality of the solutions.
3. **Reliability score:** The percentage of solutions in the archive that when recomputed in the true environment have near-identical behavioural descriptors and fitness. High reliability is key for planning tasks like long horizon navigation for omnidirectional robots, where we pick solutions according to their BD. Low reliability of an archive means that most of its solutions have a significantly different real BD to the expected BD, and are thus less suited for planning.

We choose DAQD as the appropriate benchmark, as it provides the sample efficiency improvement over vanilla QD required for online learning, can acquire new skills continuously, is applicable in any future environment, and has been shown [4] to significantly outperform vanilla QD and M-QD. As such, DAQD is a tough, relevant benchmark to outperform. In the following section we investigate LLQD's relative performance versus DAQD.

Evaluation vs naive DAQD approach

A naive approach to deal with changing environments using DAQD is to simply learn the dynamics and archive of the first environment the robot encounters, and select policies from this archive in every future environment that the robot may encounter.

In order to illustrate the effects of this approach, we recompute the archive generated under DAQD after 3,000 real evaluations in environment 0 (see Figure 5.4) onto environments 1 (Figure 5.6) and 2 (Figure 5.8). This shows how successfully the solutions that are effective in environment 0 translate to environments 1 and 2, in terms of normalized fitness (ranging from 0 to 1) and BD (final location in the xy-space reached by the robot). While the archive for environment 0 (Figure 5.4) has relatively dense Coverage of 216 solutions of high quality (total normalized QD score of 176.9), this archive performs significantly worse when applied to environments 1 and 2. In both of these environments, many solutions result in the robot moving to near identical locations in the xy-space, and as such the diversity of the archives is significantly diminished to 83 solutions (-61.6%) in environment 1 (Figure 5.6) and 99 solutions (-54.2%) in environment 2 (Figure 5.8). In addition, the quality of the archives in environments 1 and 2 is lower, with a QD score of 34.1 (-80.1%) and 62.6 (-64.6%), respectively. With this approach, using the archive of solutions for planning, like in long-horizon navigation tasks, is challenging, as the BDs and fitness of the solutions change significantly in the various environments.

In contrast, LLQD initiates a new $LLQD$ object for each environment, and thus learns a separate model and generates a distinct, specialised archive for environments 1 and 2. At 3,000 real evaluations each, this results in more diverse and higher quality archives with Coverage of 230 solutions (+177% vs DAQD) and QD score of 156.5 (+358.9% vs DAQD) for environment 1 (Figure 5.7), and Coverage of 170 solutions (+71.7% vs DAQD) and QD score of 115.3 (+84.2% vs DAQD) for environment 2 (Figure 5.9). This allows the robot to reach more locations in the xy-space, with better aligned orientation.

As the robot requires 9,000 evaluations under LLQD (3,000 evaluations per environment), we also generate the archive of environment 0 for 9,000 real evaluations under DAQD, to make a comparison at the same total number of evaluations. Recomputing the resulting archive onto environments 1 (Figure 5.10) and 2 (Figure 5.11), we find that Coverage improves to 102 solutions (+22.9% vs DAQD at 3,000 real evaluations) and 136 solutions (+37% vs DAQD at 3,000 solutions) for environments 1 and 2, respectively. However, this is still -55.6% and -20.0% lower than Coverage under LLQD. The QD score improves only marginally for environment 1 to 35.6 (+4.4% vs DAQD at 3,000 solutions) and to 88.1 for environment 2 (+40.7% vs DAQD at 3,000 solutions), which is -77.3% and -23.6% lower than under LLQD, respectively.

While we have only performed the comparison of LLQD vs the naive approach of DAQD for one run, the above analysis effectively illustrates the key challenge of the naive approach, that archives do not transfer well between significantly different environments. As such, both behaviour and performance of the solutions become unpredictable prior to real evaluation on the environment. For environments that have more similar dynamics, this naive approach may be sufficient as the solutions generated in one environment will translate better to the other environments, however, as DAQD does not have the capability to detect how closely related the dynamics of any two environments are, navigating a world with changing environments in this way is risky. Instead, Figures 5.4 to 5.11 show that there is likely a benefit to learning a new dynamics model for each environment we encounter here.

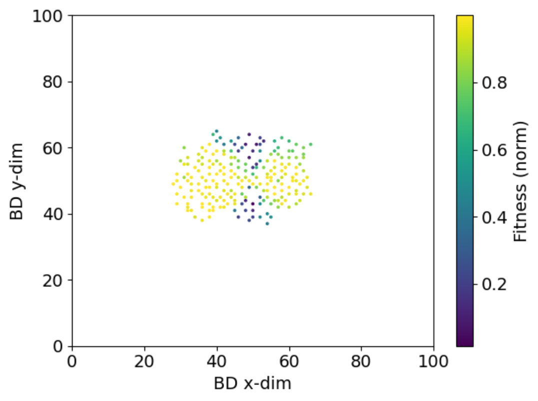

Figure 5.4: DAQD, Env 0, Evals 3k, Cov 216, QD Score 176.9

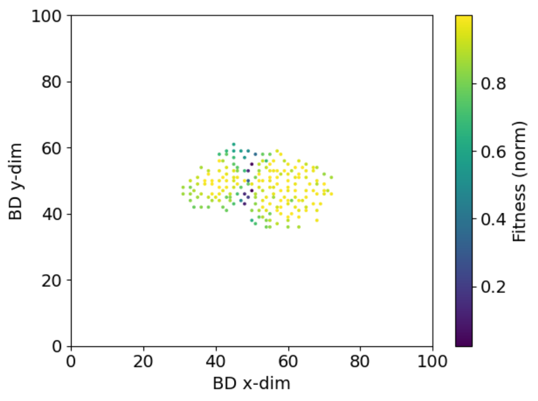

Figure 5.5: LLQD, Env 0, Evals 3k, Cov 190, QD Score 165.6

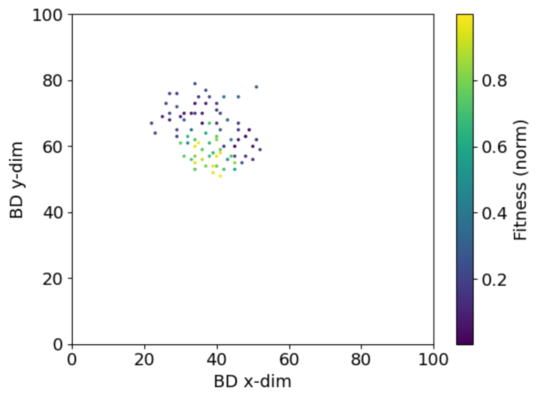

Figure 5.6: DAQD, Env 1, Evals 3k, Cov 83, QD Score 34.1

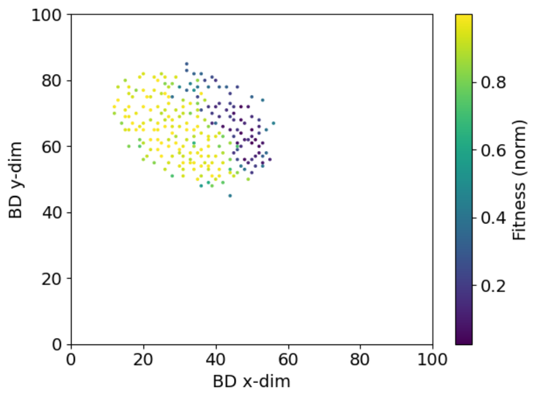

Figure 5.7: LLQD, Env 1, Evals 3k, Cov 230, QD Score 156.5

Evaluation vs sophisticated DAQD approach

A more sophisticated DAQD baseline against which to compare LLQD is to continue learning with DAQD on all environments that are encountered, rather than stopping after the first environment.

To compare the performance of LLQD and DAQD, we execute both algorithms over 20 runs on a random sequence of our three environments. Each sequence changes

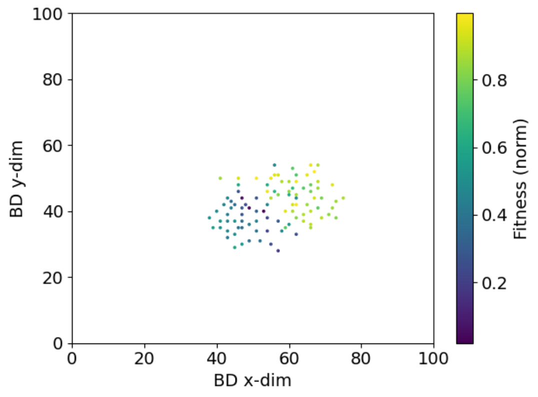

Figure 5.8: DAQD, Env 2, Evals 3k, Cov 99, QD Score 62.6

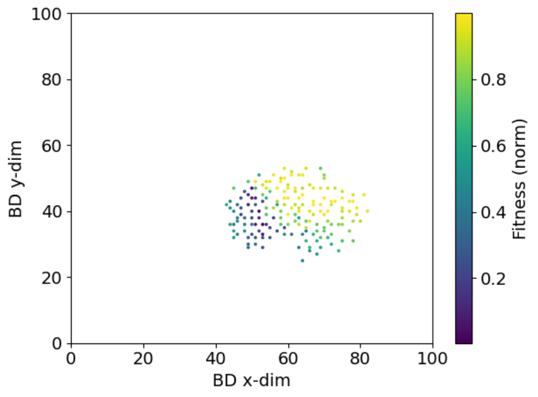

Figure 5.9: LLQD, Env 2, Evals 3k, Cov 170, QD Score 115.3

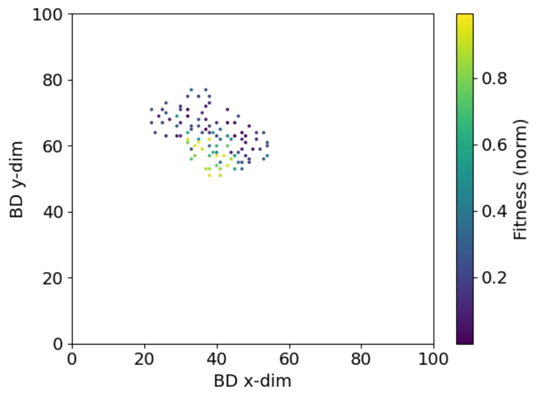

Figure 5.10: DAQD, Env 1, Evals 9k, Cov 102, QD Score 35.6

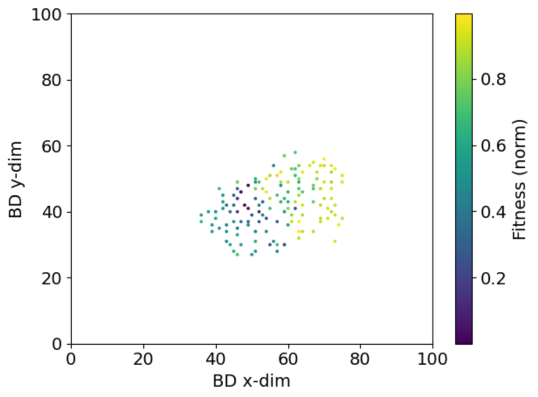

Figure 5.11: DAQD, Env 2, Evals 9k, Cov 136, QD Score 88.1

the robot's environment five times. Per environment, the robot performs about 200 real evaluations, for a total of c. 1,200 real evaluations per run. To distinguish between environments, we set the $epsilon$ parameter in LLQD to 0.5, as set out in section 4.5.

The evaluation of both algorithms is not straightforward, as both may generate archives that contain solutions that are not accurate for the associated environment. In LLQD, for example, the algorithm may fail to correctly identify an environment and thus attempt to add the evaluated candidate solutions to the wrong archive. This can occur in two ways: either the algorithm assigns the evaluated solutions to a wrong pre-existing archive, or it wrongly creates a new archive when it deems the environment to be sufficiently different from any previously seen environment (the latter occurred only once in 20 runs, creating three models when only two environments were encountered). In order to get a holistic view of the Coverage, Quality and Reliability of the archives generated, we recompute each archive generated back onto the environments from which it was created ("origin environments"). This is done by filtering-in those solutions that have near-identical BD and fitness in the generated archive and when the archive is re-computed on each of its origin environments (we call these solutions ”reliable solutions”). For example, after running DAQD on a sequence of three distinct environments, the final archive may feature 100 solutions, of which 40 were evaluated and added when the robot was in environment 0, 35 in environment 1, and 25 in environment 2. After recomputing the final archive on all three environments, we receive three ”recomputed archives” (each specialised on one of the three environments), containing only the reliable solutions for environments 0, 1, and 2, respectively. As it is very unlikely that solutions that have originally been evaluated in environments 1 and 2 have the same BD and fitness as when evaluated in environment 0, we can expect the recomputed archive for environment 0 to contain circa 40 solutions, in this example.

Across all recomputed archives, we then compute the overall Coverage as the total number of reliable solutions, and the overall QD score as the sum of the normalised fitness values of all reliable solutions. Finally, to penalise for the potentially incorrect allocation of solutions to environments, we calculate the total Reliability score as the total number of reliable solutions over the total number of solutions across environments. For example, let us assume that a robot has visited environments 0, 1, and 2. It incorrectly classified solutions from environments 0 and 1 as coming from the same environment (i.e. allocated to model 0, generating archive 0), but correctly classified solutions from environment 2 as coming from a separate environment (i.e. allocated to model 1, generating archive 1). If we assume archive 0 contains 70 solutions from environment 0 and 30 solutions from environment 1, and archive 1 contains 100 solutions from environment 2, the total Reliability score is $\frac{70+30+100}{100+100+100} = 0.67$. Measuring Reliability this way allows us to relatively severely penalise the wrong classification of environments, while at the same time putting more emphasis on environments where more solutions were generated, which tends to increase with the amount of time spent by the robot in the environment.

In DAQD, on the other hand, the algorithm will generate a single final archive that contains solutions that have been evaluated in different environments, as visited by the robot. As in LLQD, their BD and fitness will be accurate only for the environment, in which they have been evaluated, but not for the other environments. We would expect the total Reliability score for runs with three different environments to be close to $\frac{1}{3}$, and for runs with two distinct environments to be close to $\frac{1}{2}$.

The results for both algorithms are summarised in Tables 5.1 and 5.2. We find that LLQD outperforms DAQD across all three metrics, with statistical significance at the 5% level, meaning LLQD produces a greater number of higher performing and more reliable solutions.

Notably, the median outperformance in Reliability is +47.4pp higher under LLQD. On an absolute basis, we find that the median Reliability score for the archives under LLQD is 96.0%, compared to the expected 32.8% under DAQD. This means that under LLQD, for each encountered environment we can typically rely on the solutions with regards to behaviour and fitness from the associated archive when used in planning.

	Absolute performance	
	LLQD	**DAQD**
Mean Coverage	179.7	160.8
Median Coverage	179.5	164.5
Mean QD score	110.5	96.5
Median QD score	111.0	95.5
Mean Reliability score	77.8%	39.4%
Median Reliability score	96.0%	32.8%

Table 5.1: LLQD and DAQD absolute total performance across environments averaged over 20 runs of randomly generated environment sequences. Per run, the same randomly generated sequence is used for both algorithms.

	Outperformance LLQD vs DAQD		
	Coverage	**QD score**	**Reliability score**
Mean	13.8%	16.2%	38.4pp
P-value* **	0.002**	0.001**	0.000**
Median	7.2%	12.4%	47.4pp
25th percentile	1.1%	1.4%	20.0pp
75th percentile	22.4%	27.1%	64.5pp

Table 5.2: Outperformance of LLQD algorithm over DAQD on randomly generated sequence of environments over 20 runs. Minimum 200 simulations per environment, epsilon 0.5. ** indicates statistical significance at the 5% level. *** Wilcoxon signed rank test, one-sided, alternative hypothesis that distribution underlying the difference between LLQD and DAQD is ”stochastically greater than a distribution symmetric about zero” [26]

On a median basis (i.e. median outperformance across all runs) LLQD generates +7.2% greater total Coverage and a +12.4% greater total QD score. This is likely caused by two factors: firstly, the dynamics model in DAQD is trained on transitions from all environments, which have distinct dynamics. As such, when the model is used to generate a shortlist of candidate solutions (that will in turn be evaluated on the real environment when checked for addition to the real archive), the predictions of BD and fitness per solution are less accurate than in LLQD. This error in modelling reduces the sample efficiency of DAQD, and likely means that the shortlist is of lesser quality, i.e. that fewer solutions from the shortlist will be added to the real archive for the same number of real evaluations, as the model is less able to correctly predict which solutions are more novel, or have better fitness than existing solutions in the archive.

Secondly, while LLQD initialises a new archive when it detects a new environment, DAQD attempts to add all solutions to the same archive. As archives tend to become more densely populated with solutions over time, this means it becomes relatively more difficult for DAQD to add solutions. In effect, each candidate solution needs to compete against all other solutions (that have been generated across all environments), and existing solutions that were generated and perform well in one environment may be overwritten by solutions generated in a different environment.

Therefore, it also stands to reason, that as the number of real evaluations rises, the outperformance of LLQD over DAQD with respect to total Coverage and total QD score should increase. This is because as the number of real evaluations increases, DAQD will increasingly wrongly replace existing solutions in the archive with new candidate solutions from a different environment. To illustrate this, we ran LLQD and DAQD on 10,000 real evaluations, equally split between environments 0 and 1. As reported in Table 5.3, with the caveat of only having executed this experiment over one run, the outperformance of LLQD over DAQD increases to +34.3% in total Coverage and +32.6% in total QD score.

	LLQD	DAQD	LLQD Outperformance
Coverage	564	420	+34.3%
QD Score	442.2	333.4	+32.6%

Table 5.3: LLQD vs DAQD performance with respect to total overall Coverage and QD score for 10,000 real evaluations; two environments, over one run.

5.3.2 Ablation study 1: effect of $epsilon$ on environment detection

A key component of LLQD is its ability to correctly identify the environment the robot is currently in. More precisely, LLQD needs to detect whether the most recent steps taken by the robot belong to one of the previously encountered environments (and if so which one), or whether they belong to a new environment. As outlined in section 4.5, to do so, we calculate the mean likelihood across all the recent steps taken by the robot of being in each environment, by comparing the observed changes in state of the robot to the predictions from the trained dynamics models, for each existing $LLQD$ object and associated environment. As previously outlined, LLQD determines the robot to be in the most likely $LLQD$ object's associated environment, unless the mean likelihood of the most recent steps taken is lower than $epsilon$ standard deviations from the mean likelihood of holdout set transitions previously assigned to this $LLQD$ object.

As such, the choice of $epsilon$ (or abbreviated EPS in Algorithm 2) is a key parameter that determines whether LLQD is able to correctly identify the environments. If the value chosen for $epsilon$ is too low, LLQD will tend to create too many new $LLQD$ objects for the same environment (for instance when the model evaluates the likelihood of transitions partially outside of the previous training domain), while too high a value for $epsilon$ means LLQD will fail to detect new environments sufficiently frequently.

To assess LLQD's ability to correctly classify the environments it encounters, for a given run we compare the real sequence of environments to LLQD's detected sequence. As before, for each of the 20 runs, we visit six environments. As LLQD builds up its memory of $LLQD$ objects sequentially, the first environment encountered will always be associated with $LLQD$ object (and model) 0, hence, we ignore the first classification in our scoring. As such, per run, there are 5 environments where LLQD could fail to make a correct classification. A classification is deemed incorrect, when datapoints are wrongly assigned to a previously encountered environment, or when it wrongly assigns datapoints belonging to a previously encountered environment to a new model. For example, if the real sequence of environments encountered is [1, 2, 1, 1, 0, 2], and LLQD's estimate is [0, 1, 2, 2, 3, 1], the associated correctness score is $\frac{3}{5} = 60\%$, as the second, fifth, and sixth environment have been adequately mapped. Here, while the correct mapping would have been [0, 1, 0, 0, 2, 1], labelling the fifth environment as 3 is sequentially correct, as it correctly identifies the environment as previously unseen.

We evaluate the percentage of correctly identified environments for LLQD over 20 runs on random environment sequences. The results for $epsilon$ values of 0.3, 0.5, and 0.7 are reported in Table 5.4.

	Epsilon			
% correct	**0.3**	**0.5**	**0.7**	**Random**
Mean	51.0%	77.0%	67.0%	31.4%
Median	40.0%	100.0%	80.0%	40.0%
25th percentile	35.0%	40.0%	40.0%	20.0%
75th percentile	65.0%	100.0%	100.0%	40.0%

Table 5.4: Percentage of correctly identified environments for various values of epsilon over 20 runs each, compared to a baseline of random classification. Per environment c. 100 simulations are performed, with c. 300 transitions of state, action, next state per simulation.

We find that an $epsilon$ value of 0.5 performs best, with 77% of environments correctly classified at the mean, and a median correct classification of 100%. This compares favourably versus a baseline of random classification (evaluated over $1e^6$ runs) that respects the sequential build-up of the estimated environment list. For the latter, the mean and median correctness score are 31.4% and 40.0%, respectively.

An important caveat is that the optimal value of epsilon with regards to the correct classification of environments likely varies acccording to how closely related the dynamics of the different environments are that the robot encounters. For instance, to correctly distinguish between two very similar environments, the optimal value of epsilon is likely smaller. However, it stands to reason, that when the dynamics of two environments are very similar, the benefit of distinguishing between the two environments also diminishes.

5.3.3 Ablation study 2: effect of steps taken per environment on environment detection

Another key parameter in LLQD's ability to correctly classify the environments is the number of steps the robot takes in each environment (and thus the number of transitions it can collect) before it assesses to which model the transition data from the real evaluations should be assigned. There are two main reasons for this: firstly, the more transitions we collect per environment the more robust and accurate the mean likelihood of our sample of transitions should become, relative to the environment's population mean likelihood. As a result, it should be less affected by transitions from outlier controls. We see this in Figures 5.12 to 5.14, which show the distributions of the mean $log\_likelihoods$ per control becoming more well-behaved as the number of random controls increases from 50 to 200.

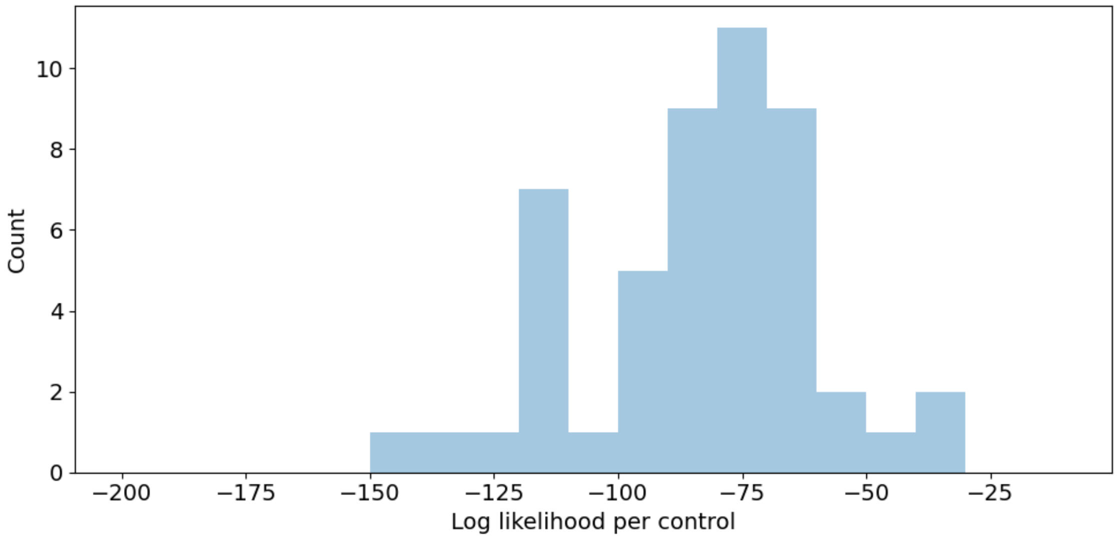

Figure 5.12: Distribution of mean log likelihoods for 50 random controls simulated in environment 1 and evaluated with model pre-trained on environment 1.

Secondly, as we add more transition datapoints to each model, we widen the training distribution of data for each model. This should help reduce overfitting and allow the model to generalise better to unseen transition data, which in turn makes the correct environment classification of transitions more likely. Table 5.5 shows the results of performing c. 50, 100, and 200 simulations per environment visit, with respect to correct environment classification.

As expected, we find that as the number of simulations and thus datapoints collected per visited environment increases, the performance of LLQD in terms of correctly classifying the environments rises. While the median classification performance slightly drops from 100% at 100 simulations to 90% at 200 evaluations per environment, the mean performance increases by +4pp to 81%. More importantly, however, classification seems to become more robust at 200 evaluations, with the lower end of the interquartile range improving from 40% to 60% correctness, which

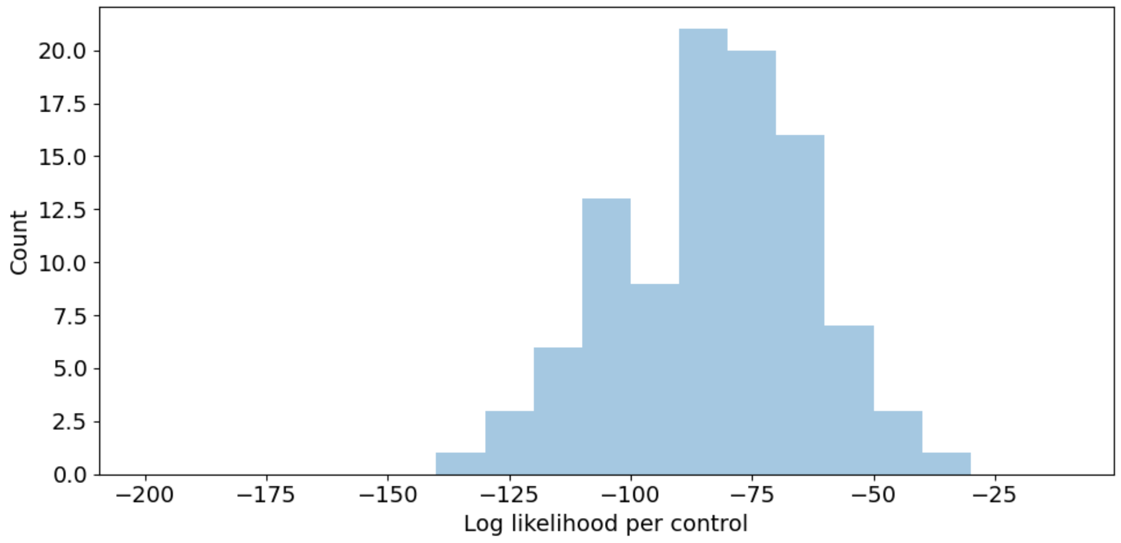

Figure 5.13: Distribution of mean log likelihoods for 100 random controls simulated in environment 1 and evaluated with model pre-trained on environment 1.

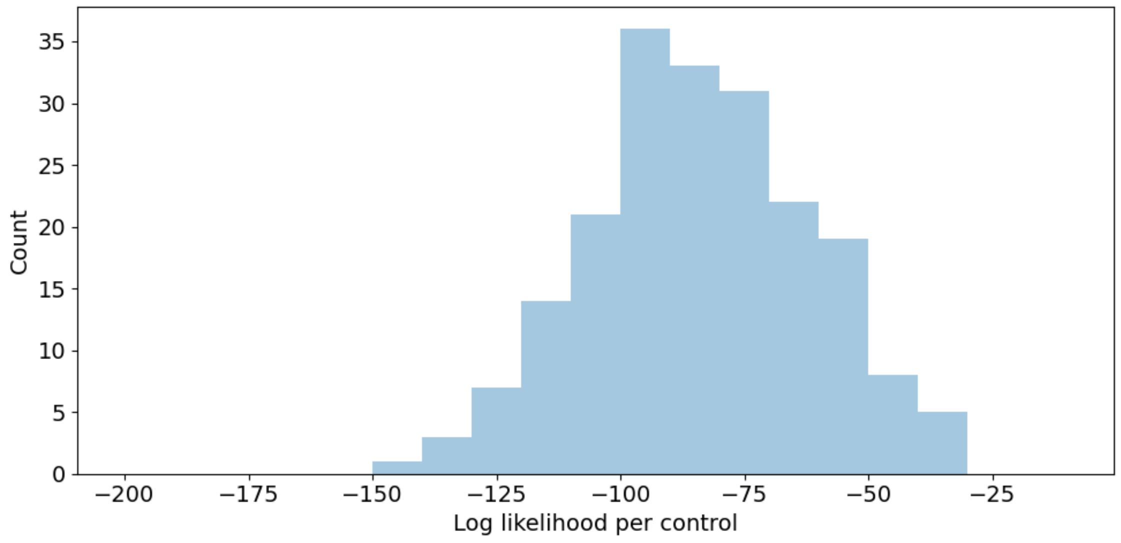

Figure 5.14: Distribution of mean log likelihoods for 200 random controls simulated in environment 1 and evaluated with model pre-trained on environment 1.

is +20pp higher than the 75th percentile of the random baseline previously recorded in Table 5.4.

However, it is important to remember that a trade-off exists between the number of steps taken per environment and the viability of using LLQD in online learning. This is because every real step taken has an associated safety risk (e.g. damage to the robot) and steps are taken sequentially. At 200 simulations of 3 seconds each, for example, the robot would need to spend circa ten minutes in each visited environment, exploring without failure. In real life it may be difficult to ensure such

	Simulations per environment		
% correct	**50**	**100**	**200**
Mean	63.0%	77.0%	81.0%
Median	60.0%	100.0%	90.0%
25th percentile	55.0%	40.0%	60.0%
75th percentile	80.0%	100.0%	100.0%

Table 5.5: Percentage of correctly identified environments for various numbers of simulations per visited environment. Per simulation c. 300 transitions of state, action, next state are recorded, used for environment detection, and ultimately added to the mapped model's replay buffer.

prolonged exploration within the same environment, particularly when we do not have control over the changes in the environment (e.g. rain making the surface more slippery), or change cannot be prevented (e.g. reduction in the robot's battery charge over time).

5.3.4 Ablation study 3: stretch performance for environment detection

Having determined suitable hyperparameter values for $epsilon$ (0.5) and the number of simulations per environment (c. 200), which is proportional to the number of real steps taken per environment, we investigate LLQD's ability to correctly handle and classify the visited environments in a more difficult stretch case. Here, the environment constantly changes randomly to one of the other two alternative environments, after every 200 real evaluations. A representative example for a compliant sequence is [1, 0, 2, 0, 2, 1]. This stands in contrast to the completely random environment sequence generation in the previous experiments, where the environment is not guaranteed to change from one step in the sequence to another (e.g. [1, 2, 2, 2, 0, 0]).

As before, we perform the analysis over 20 runs of sequences with five environment changes. The findings are reported in Table 5.6.

% correct	Base case	Stretch case
Mean	81.0%	77.0%
Median	90.0%	80.0%
25th percentile	60.0%	60.0%
75th percentile	100.0%	100.0%

Table 5.6: Percentage of correctly classified environments for base case of randomized environments and stretch case of constantly changing randomized environments. Performance over 20 runs. $Epsilon$ set to 0.5, number of simulations per visited environment set to 200.

We find that LLQD's detection performance holds up well even in the more difficult stretch scenario, with the mean and median percentage of correctly handled and identified environments dropping only -4pp to 77.0% and -10pp to 80.0%, respectively. Encouragingly, the interquartile range remains steady, from 60.0% to 100.0%. In fact, using the one-sided Wilcoxon rank-sum test, we cannot find with statistical significance at the 5% level that the distribution underlying the correctness scores for the stretch case is "stochastically less than the distribution underlying" [27] the correctness scores of the base case (P-value of 0.32).

5.3.5 Ablation study 4: effect of transfer learning on speed of archive generation and quality

One of the key benefits of DAQD over vanilla QD in online learning is the up to 20x increase in sample efficiency [4].

As LLQD initialises every new model with the weights and biases of the next most likely model given the observed transitions when it detects a sufficiently different environment (see section 4.6), we may expect LLQD to have the potential to require fewer real evaluations than DAQD (applied to each environment individually) to reach the same archive coverage and QD score for all environments visited after the first environment.

To examine this, we compare the median evolution of Coverage and QD score over the number of real evaluations for the archives for environments 0, 1, and 2, when the latter are learnt individually using DAQD versus when they are initialized with the trained parameters from the other (i.e. transfer) models, as in LLQD (e.g. generating the archive for environment 0, when model 0 is instantiated with the trained parameters from the models associated with environment 1 or 2).

For each transfer scenario (e.g. learning model 0 from model 1, learning model 0 from model 2, etc.) the transfer model is trained on 200 simulations. The evolutions of the archives with respect to Coverage are visualised in Figures 5.15 to 5.20, and in Figures 5.21 to 5.26 with respect to QD score, over 10 runs per scenario.

We find that for all transfer scenarios, apart from learning the model associated with environment 2 from the model associated with environment 1, there appears to be some statistically significant evidence of outperformance of the transfer learning approach inherent to LLQD over the individual learning approach inherent to DAQD, either with respect to Coverage, QD score, or both. Even for the most challenging scenario (learning model 2 from model 1), the transfer learning approach appears to perform on par with DAQD. In addition, if a model for environment 0 has been learnt, LLQD would likely default to instantiate the model for environment 2 with model 0's parameters where outperformance is observed, as transitions from environment 2 have a higher likelihood under the model learnt for environment 0 than for environment 1 (see Figure 4.4). As such, LLQD indeed appears to have the potential for building more diverse and higher quality archives with fewer real evaluations than DAQD, and thus exhibits the potential for improved sample efficiency.

Tests for the statistical significance of the outperformance of the transfer learning approach with respect to Coverage and QD score are provided in Tables 5.7 and 5.8, respectively. We provide P-values both at 400 and 800 real evaluations (i.e. simulations). This confirms that for all except one transfer scenario there is at least some evidence (statistically significant at the 10% level) of outperformance in relation to QD score, Coverage or both (statistically significant at the 5% level for three out of the six scenarios).

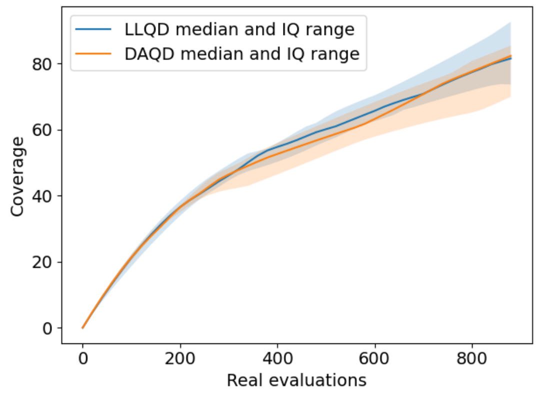

Figure 5.15: LLQD learning model for env 0 from env 1 vs DAQD baseline

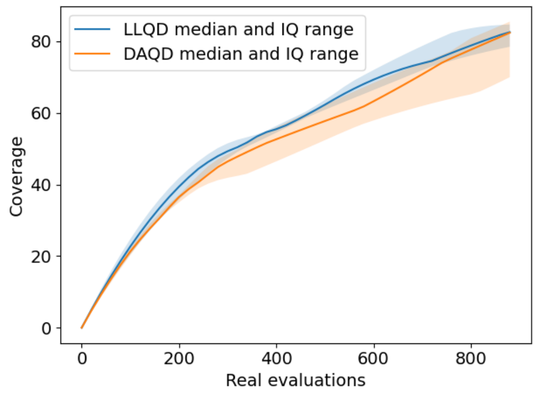

Figure 5.16: LLQD learning model for env 0 from env 2 vs DAQD baseline

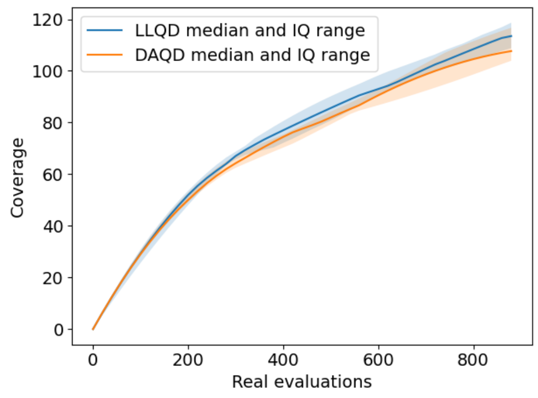

Figure 5.17: LLQD learning model for env 1 from env 0 vs DAQD baseline

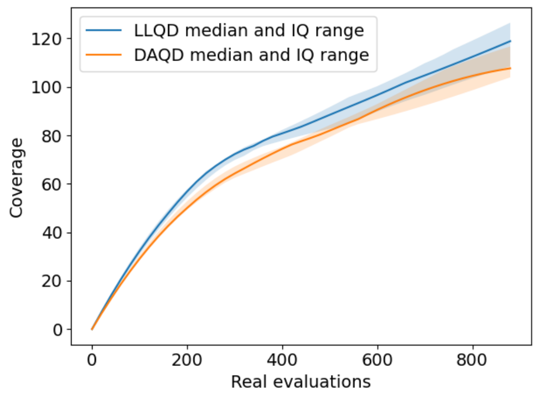

Figure 5.18: LLQD learning model for env 1 from env 2 vs DAQD baseline

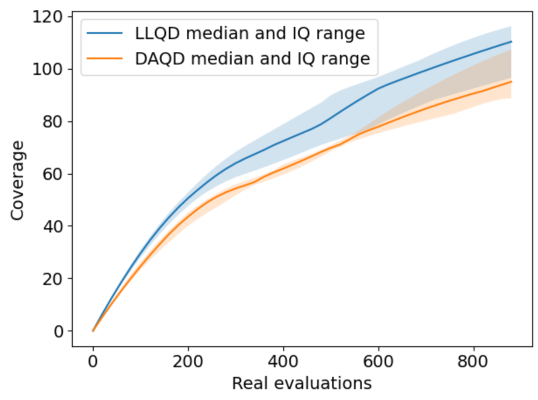

Figure 5.19: LLQD learning model for env 2 from env 0 vs DAQD baseline

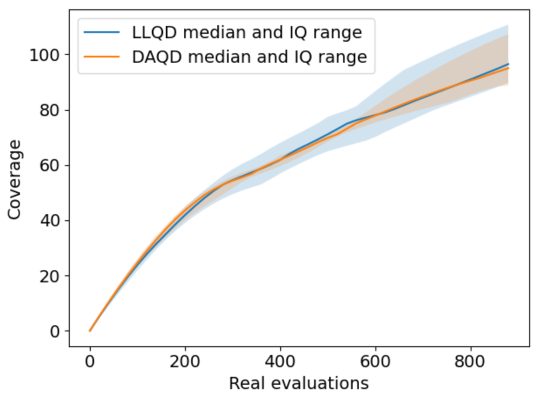

Figure 5.20: LLQD learning model for env 2 from env 1 vs DAQD baseline

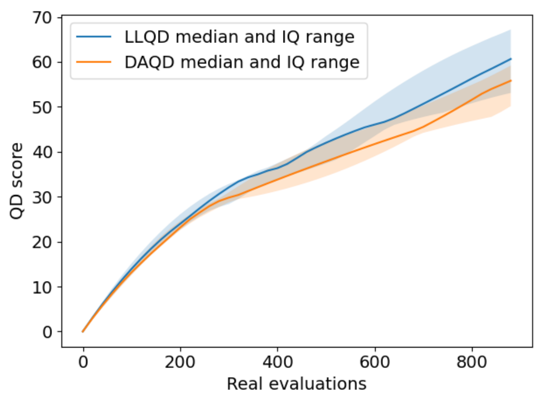

Figure 5.21: LLQD learning model for env 0 from env 1 vs DAQD baseline

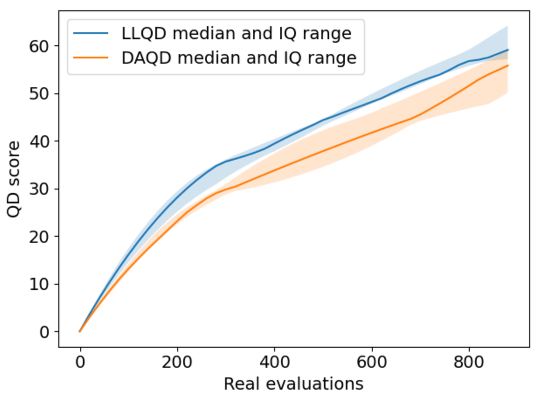

Figure 5.22: LLQD learning model for env 0 from env 2 vs DAQD baseline

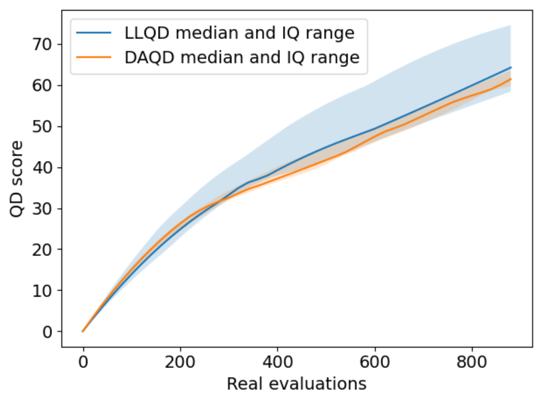

Figure 5.23: LLQD learning model for env 1 from env 0 vs DAQD baseline

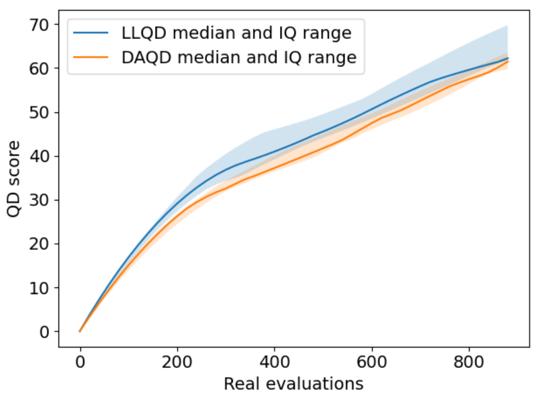

Figure 5.24: LLQD learning model for env 1 from env 2 vs DAQD baseline

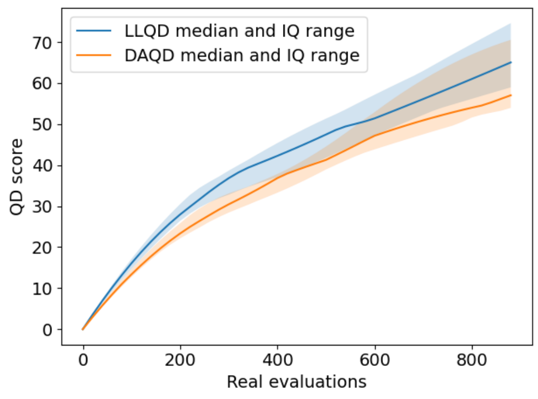

Figure 5.25: LLQD learning model for env 2 from env 0 vs DAQD baseline

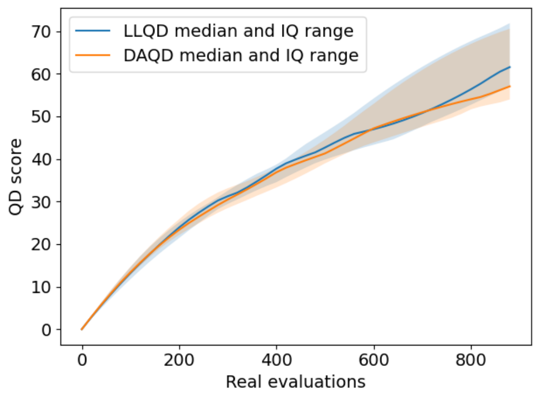

Figure 5.26: LLQD learning model for env 2 from env 1 vs DAQD baseline

		P-value	
Learn env	**From env**	**@ 400 evals**	**@ 800 evals**
0	1	0.225	0.145
0	2	0.141	0.217
1	0	0.056*	0.163
1	2	0.003**	0.087*
2	0	0.008**	0.056*
2	1	0.353	0.500

Table 5.7: P-values for one-sided Wilcoxon rank-sum tests with alternative hypothesis that the distribution of archive Coverage underlying LLQD ”is stochastically greater than the distribution underlying” [27] DAQD. Measured over 10 runs.

		P-value	
Learn env	**From env**	**@ 400 evals**	**@ 800 evals**
0	1	0.248	0.056*
0	2	0.016**	0.010**
1	0	0.128	0.248
1	2	0.041**	0.113
2	0	0.014**	0.128
2	1	0.163	0.440

Table 5.8: P-values for one-sided Wilcoxon rank-sum tests with alternative hypothesis that the distribution of archive QD score underlying LLQD ”is stochastically greater than the distribution underlying” [27] DAQD. Measured over 10 runs.

Chapter 6

Ethical considerations

6.1 Dual use dilemma

As previously discussed, the aim of this research project is to devise an algorithm that allows a walking robot to continuously adapt online to previously unseen changes in its environment, without the need for human intervention.

While our research is solely focused on the application of learning an omni directional repertoire of walking gaits for a hexapod robot, the LLQD algorithm itself is subject to the ”dual-use dilemma” [28], insofar as it could be used, or lead to subsequent technologies that could be used, to intentionally cause harm.

At its core, LLQD allows a robot to detect previously encountered or significantly new environments, and then autonomously builds a diverse repertoire of behaviours specialized to that environment, leveraging prior knowledge from other environments. Applied to military ends this could, for example, lead to more potent combat robots that are more effective at killing humans by adapting to changing terrains autonomously. It may also result in the robot engaging in unethical behaviours that conflict with the rules of war, such as targeting civilians [29], if those behaviours result in more effective combat.

Despite this, our research is fundamental, and we take no steps adapting the algorithm towards exploring or facilitating military use cases.

6.2 Environmental considerations

LLQD is a type of Quality Diversity algorithm, and as such is an evolutionary algorithm [30]. The latter, however, require a ”large number of function evaluations” [1] to produce high performing solutions, and thus tend to require a high number of computations that may consume more energy. The latter may not always be generated from renewable sources. When deployed in real life, this will additionally require a high number of battery charges for the robot. LLQD attempts to alleviate this concern through its improved sample efficiency.

Chapter 7

Conclusion

7.1 Summary

We introduced LLQD as a lifelong model-based reinforcement QD algorithm for non-stationary environments. Within the task of a hexapod robot learning an omni-directional repertoire in changing environments, LLQD enables the robot to detect previously encountered or new environments and to build a specialised repertoire of behaviours for each different environment. It performs the detection and handling of environments without the need for pre-training in simulation on some distribution of environments, but instead uses the robot's state and action trajectories to sequentially learn a probabilistic dynamics model for each sufficiently different environment encountered. In order to determine the robot's current environment, LLQD computes the $log\_likelihoods$ of obtaining the robot's most recent state and action trajectories under all models associated with the previously encountered environments. It then identifies the current environment as the environment associated with the model with the highest $log\_likelihood$, where the latter is not substantially lower than the mean $log\_likelihood$ of the model's hold-out set. If no existing model qualifies, a new environment is detected, and an associated new model is instantiated with the weights of the next most likely model.

Using a set of three environments in randomly changing order, we evaluate the performance of LLQD versus DAQD. We choose DAQD as the comparison, as it can continuously acquire new skills in any reasonable environment, and provides a strong sample efficient baseline that significantly outperforms alternative algorithms, like vanilla QD or M-QD, by requiring up to 20x fewer real evaluations to produce archives of equal coverage and QD score. With six environment changes and 200 real evaluations each, we find that LLQD outperforms DAQD with regards to total Coverage (+7.2% median outperformance), and total QD score (+12.4% median outperformance) across environments, and in terms of the Reliability score of the archives (+47.4pp median outperformance) with statistical significance at the 5% level. The outperformance with regards to total Coverage and QD score likely rises as the number of real evaluations increases.

With the correct identification of environments as the key source of LLQD's outperformance, we next investigate the effect of two important hyperparameters on the percentage of environments correctly detected. Firstly, we examine the $epsilon$ parameter, which determines the sensitivity with which real observations are assigned to existing models and their associated environments. Here, we find that for values of $epsilon$ that are too low or too high, the percentage of correct detection decreases, as either too many models are created for the same environment, or transitions from different environments are assigned to the same model, respectively. With the correctly tuned value for $epsilon$, the three tested environments are on average (mean) correctly detected 77% of the time (100% median), versus a random identification baseline of 31.4% at the mean (40% median). Secondly, we examine how the number of steps taken by the robot in the environment prior to environment detection affects the number of correctly identified environments. We observe that as the number of steps increases, the mean percentage of correctly identified environments improves from 63% at 50 real simulations (299 steps per simulation) to 81% at 200 simulations, and detection seems to become more robust, with the interquartile range tightening from 40%-100% at 100 simulations to 60%-100% at 200 simulations. Still, we note that a trade-off exists between the number of steps taken, and the viability of using LLQD as an online learning algorithm, as each simulation takes three seconds at the set control frequency.

Using the correctly tuned hyperparameters for $epsilon$ and the number of steps taken per environment, we test LLQD's ability to correctly detect environments on the difficult stretch case of constantly randomly changing environments, versus a baseline of randomly changing environments (the latter may encounter the same environment multiple times in a row, while the former does not). While the mean and median percentage of correctly identified environments slightly drops from 81% to 77% and from 90% to 80% respectively, this drop in performance is not statistically significant at the 5% level, with a P value of 0.32.

Finally, we examine the efficacy of LLQD's transfer learning mechanism, that instantiates new models for new environments with the weights and biases of the existing model with the highest $log\_likelihood$ for the observed transition data. Iterating through all pairs of combinations of an environment-specific model inheriting the weights from the model of a previously seen environment (e.g. learning the model for environment 0 using the trained weights from environment 1 after 200 simulations), we find that for all but one scenario there exists at least some statistically significant evidence of LLQD's outperformance relative to the baseline of learning the models from scratch with regards to Coverage, QD score or both at the same number of real evaluations.

In summary, LLQD therefore provides a sample efficient model-based QD implementation that can effectively handle non-stationary environments by creating reliable and high quality environment specific repertoires. Given its sample efficiency, and without the need for pre-training an environment adaptation module or making restrictive assumptions about the distribution of environments that may be encountered in the future, LLQD is well suited for online lifelong learning. As such, unlike existing algorithms in the QD adaptation space, it can continuously acquire new skills in new environments online. Given the strong reliability of the solutions in the generated archives for each environment, LLQD creates archives that may also be effectively used in planning tasks.

7.2 Future work

This research paper explored the viability of LLQD as a lifelong learning algorithm for changing environments. In our implementation, we use resets to bring the hexapod back to its original position after every simulation. As the three representative environments we test LLQD on have continuous dynamics throughout their respective maps, it does not matter where in the map the robot is located when it executes the candidate controls. As such, resets do not affect the accuracy of our findings, but provide a simpler and more convenient way to handle situations where the hexapod flips onto its back, as well as a more direct comparison to DAQD, where resets are also used.

Still, for a complete online learning application we should ideally avoid the use of resets, which requires our robot to have a notion of safety to avoid executing potentially unsafe behaviours that might damage or otherwise incapacitate the robot. As such, future work should explore the integration of RFQD (Reset-Free Quality Diversity) [15] into LLQD, which allows for the reset-free exploration of open-ended environments by using a safety-compliant behaviour selection policy. Interestingly, the latter may also be used to ensure that the hexapod remains within the same environment for the required duration to execute a sufficient number of steps to generate a high quality archive, robust model, and improve model detection (see section 5.3.3). Using open-ended maps will then allow us to more easily assess the performance of LLQD and the necessity for any adaptations to handle gradually changing environments, as opposed to the sudden changes in environments that we explored in this paper. In addition, as LLQD is not immune to assigning datapoints to the wrong environment, future work may explore the integration of RTE applied to each archive individually, for planning tasks.

Further to this, we observe from Figures 5.5, 5.7, and 5.9 that using a random uniform selector for parent genotypes can result in archives that do not evenly explore all directions in our omni-directional task. For example, in slanted environments due to gravity controls exhibit a bias towards the direction of steepest decline, resulting in archives that are less dense in the opposite direction of the BD space. Using a selector that prioritises less dense areas of the BD space should help here, by making the robot evaluate more mutations of controls that move towards more difficult-to-reach spaces.

Finally, our skill descriptor is limited to two dimensions (i.e. position in the xy-space). To traverse changing environments, it may be useful for our hexapod to learn more complex skills, that would benefit from a higher-dimensional skill descriptor (e.g. exploring the xyz-space for stepping or climbing). As such, future work may wish to explore the use of a learnt BD space instead, for instance using deep auto-encoders or PCA.

Bibliography

[1] T. Chugh, K. Sindhya, J. Hakanen, and K. Miettinen, "A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms," *Soft Computing*, vol. 23, pp. 3137–3166, 2019.

[2] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[3] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems*, vol. 100, pp. 236–250, 2018.

[4] B. Lim, L. Grillotti, L. Bernasconi, and A. Cully, *Dynamics-aware quality-diversity for efficient learning of skill repertoires*, 2021. DOI: 10.48550/ARXIV.2109.08522. [Online]. Available: https://arxiv.org/abs/2109.08522.

[5] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[6] K. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret, "Quality-diversity optimization: A novel branch of stochastic optimization," in *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, Springer, 2021, pp. 109–135.

[7] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2018.

[8] A. Cully, "Autonomous skill discovery with quality-diversity and unsupervised descriptors," in *Proceedings of the Genetic and Evolutionary Computation Conference*, Prague, Czech Republic: ACM, 2019, pp. 81–89. DOI: 10.1145/3321707.3321804.

[9] G. Paolo, A. Laflaquiere, A. Coninx, and S. Doncieux, "Unsupervised learning and exploration of reachable outcome space," *algorithms*, vol. 24, p. 25, 2019.

[10] J. K. Pugh, L. Soros, P. A. Szerlip, and K. O. Stanley, "Confronting the challenge of quality diversity," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, ACM, 2015, pp. 967–974.

[11] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[12] J. Lehman and K. O. Stanley, “Evolving a diversity of virtual creatures through novelty search and local competition,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, 2011, pp. 211–218.

[13] A. Gaier, A. Asteroth, and J.-B. Mouret, “Data-efficient design exploration through surrogate-assisted illumination,” *Evolutionary computation*, pp. 1–30, 2018.

[14] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, “Information-theoretic regret bounds for gaussian process optimization in the bandit setting,” *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3250–3265, May 2012. DOI: `10.1109/tit.2011.2182033`. [Online]. Available: `https://doi.org/10.1109%2Ftit.2011.2182033`.

[15] B. Lim, A. Reichenbach, and A. Cully, “Learning to walk autonomously via reset-free quality-diversity,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, Jul. 2022. DOI: `10.1145/3512290.3528715`. [Online]. Available: `https://doi.org/10.1145%2F3512290.3528715`.

[16] L. Keller, D. Tanneberg, S. Stark, and J. Peters, “Model-based quality-diversity search for efficient robot learning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[17] D. M. Bossens and D. Tarapore, “Rapidly adapting robot swarms with swarm map-based bayesian optimisation,” *arXiv preprint arXiv:2012.11444*, 2020.

[18] R. Kaushik, P. Desreumaux, and J.-B. Mouret, “Adaptive prior selection for repertoire-based online learning in robotics,” *arXiV*, vol. abs/1907.07029, 2019. [Online]. Available: `http://arxiv.org/abs/1907.07029`.

[19] T. F. Nygaard, C. P. Martin, D. Howard, J. Torresen, and K. Glette, “Environmental adaptation of robot morphology and control through real-world evolution,” *arXiv preprint arXiv:2003.13254*, 2020.

[20] A. Kumar, Z. Fu, D. Pathak, and J. Malik, *Rma: Rapid motor adaptation for legged robots*, 2021. DOI: `10.48550/ARXIV.2107.04034`. [Online]. Available: `https://arxiv.org/abs/2107.04034`.

[21] A. Nagabandi, I. Clavera, S. Liu, *et al.*, *Learning to adapt in dynamic, real-world environments through meta-reinforcement learning*, 2018. DOI: `10.48550/ARXIV.1803.11347`. [Online]. Available: `https://arxiv.org/abs/1803.11347`.

[22] K. Chua, R. Calandra, R. McAllister, and S. Levine, *Deep reinforcement learning in a handful of trials using probabilistic dynamics models*, 2018. DOI: `10.48550/ARXIV.1805.12114`. [Online]. Available: `https://arxiv.org/abs/1805.12114`.

[23] *Dart physics engine documentation*, `https://dartsim.github.io`, Accessed: 2022-08-21.

[24] *Robotdart simulator documentation*, `http://www.resibots.eu/robot_dart/`, Accessed: 2022-08-21.

[25] A. Cully and J.-B. Mouret, “Behavioral repertoire learning in robotics,” in *Proceedings of the 15th annual conference on Genetic and Evolutionary Computation*, ACM, 2013, pp. 175–182.

[26] *Scipy wilcoxon signed rank test documentation*, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html`, Accessed: 2022-08-21.

[27] *Scipy wilcoxon ranksum test documentation*, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ranksums.html`, Accessed: 2022-08-23.

[28] S. Miller, “Moral responsibility, collective-action problems and the dual-use dilemma in science and technology,” in *On the Dual Uses of Science and Ethics: Principles, Practices, and Prospects*. ANU Press, 2013, pp. 185–206. [Online]. Available: `http://www.jstor.org/stable/j.ctt5hgz15.17` (visited on 08/29/2022).

[29] J.-M. Henckaerts, L. Doswald-Beck, C. Alvermann, K. Dörmann, and B. Rolle, *Customary International Humanitarian Law*. Cambridge University Press, 2005, vol. 1. DOI: `10.1017/CBO9780511804700`.

[30] *Quality-diversity optimisation algorithms*, `https://quality-diversity.github.io`, Accessed: 2022-08-29.

[31] Resibots, *Pymap elites*, `https://github.com/resibots/pymap_elites`, Accessed: 2022-08-29.

[32] K. Lu, *Lifelong rl*, `https://github.com/kzl/lifelong_rl`, Accessed: 2022-08-29.

Appendix A

Codebase

The repository containing the implementation of *LLQD* can be found at:

https://tinyurl.com/tth21llqd

As previously outlined, LLQD developed out of the key innovations from DAQD [4], and as such our code is built on top of DAQD's codebase. DAQD, in turn, leverages the MAP-Elites implementation from [31], and the core architecture of the probabilistic dynamics model ensemble from [32].

The key components of the code of this project are:

- **LLQD script**: the main script to perform LLQD on some randomized sequence of environments, containing the complex logic for the environment detection/handling and transfer learning features of LLQD.
- **LLQD class**: the definition of the LLQD class, which stores all the relevant objects (e.g. dynamics model, buffers, archives) and contains the logic that splits evaluation of candidate controls from the assignment of trajectories to the buffer and training of the relevant model.
- **LLQD utilities**: contains all the utility functions needed to perform LLQD, such as computing log likelihoods of transitions, or creating randomized environment sequences and scoring.
- **Environment utilities**: consists of the alternative environments used in our analysis, as well as the complex implementation logic to switch environments during script execution.
- **Visualisation script**: a script to visualise various aspects of the LLQD algorithm, such as the hexapod gait, or distribution of the log likelihoods of the trajectories for the various models.
- **Experiment scripts**: scripts for all conducted experiments, both for LLQD and its DAQD comparison, ranging from the main evaluation of LLQD vs DAQD, to

various ablation studies including the assessment of LLQD's transfer learning capabilities.

- **Results**: contains the results for all experiments for verification purposes.