Detecting Environment Drift in Reinforcement Learning Using a Gaussian Process

Zhizhou Fang
Research Group Software Architecture
Faculty of Computer Science
Doctoral School Computer Science (DoCS)
University of Vienna, Austria
zhizhou.fang@univie.ac.at

Uwe Zdun
Research Group Software Architecture
Faculty of Computer Science
University of Vienna, Austria
uwe.zdun@univie.ac.at

***Abstract*—This study introduces a novel two-stage method, GPAction, for detecting environment drift in reinforcement learning settings. We first train a Gaussian process predicting the reinforcement learning agents' actions and then detect environment drifts by monitoring the mean squared error between the predicted actions of a Gaussian process action predictor and actual actions. Our proposed method is evaluated against three baselines across four environments with continuous action spaces. Results demonstrate the superior performance of GPAction in detecting environment drift. In an ablation study, by analyzing the plots and AUC values of the MSEs, we show that our method GPAction can provide more distinguishable monitoring metrics than the Gaussian process state predictor.**

***Index Terms*—Reinforcement Learning, Environment Drift, MLOps**

I. Introduction

Reinforcement learning (RL) is a machine learning (ML) paradigm. It learns agents' policies to map states to actions from an environment. The agents are not told which actions to take but, instead, discover which actions yield the most reward by interacting with the environment through trial and error [1]. Machine learning operations (MLOps) is a set of practices aiming to deploy, monitor, and maintain ML models in production environments reliably and efficiently [2]. In the context of RL, MLOps involves continuous monitoring of the agent's performance to detect any degradation that may occur due to environment drift, i.e., changes in the underlying environment dynamics $P(S_{t+1}|S_t, A_t)$ [3]. Traditional monitoring metrics, such as step rewards and their moving averages, are commonly used during training but may not always provide the most sensitive or accurate drift detection. There are several reasons for this limitation: (1) Uninformative rewards in some environments: In some RL environments, step rewards may not be informative about the agent's performance. (2) Insensitivity to subtle drifts: Step rewards are often designed to reflect an action's immediate success or failure, which means they may not be sensitive to slight or gradual changes in the environment's dynamics.

This study aims to establish a better monitoring metric by learning a Gaussian Process (GP) model in the training environment and monitoring its performance in the production environment. We assume that the performance of the GP will drop if environment drift happens. By comparing the effectiveness of this approach with traditional metrics, we seek to improve the detection of environment drift and enhance the robustness of RL policies in production environments. Previous research has demonstrated the utility of GP in various machine learning applications, including model-based RL methods such as PILCO [4], which employs GPR to predict state transitions.

In our study, we address the following research questions:

- **RQ1** How effective is a GPR predicting $P(A_t|S_t, \Delta S_t)$ in detecting environment drift compared to a GPR predicting $P(\Delta S_t|S_t, A_t)$ used in PILCO [4]?
- **RQ2** How effective is a GPR predicting $P(A_t|S_t, \Delta S_t)$ in detecting environment drift compared to traditional metrics, such as step rewards and moving averages of step rewards?

By exploring these questions, we aim to comprehensively evaluate GP-based monitoring metrics and their potential to improve environmental drift detection in RL systems. Our main contributions are:

- We introduce a two-stage method leveraging a Gaussian Process to detect environment drift.
- We evaluate our method across four environments and compare it with three baselines, demonstrating its superior performance.
- We conduct an ablation study to provide insights into the benefits of our approach.

The rest of the paper is organized as follows: Section II formalizes the problem statement. Section III reviews related works. Section IV presents our methodology. Section V details the experimental setup and results. Finally, Section VI concludes the paper and discusses future work.

II. Problem Statement

Environment drift in reinforcement learning (RL) is defined by Wang et al. [3] as changes in the conditional distribution $P(S_{t+1}|S_t, A_t)$, where S_t and S_{t+1} are the states at times t and $t+1$, and A_t is the action at time t. Environment drift detection aims to predict the time step at which the environment drift happens. The scope of this work is limited to RL environments with continuous action spaces.

III. Related Works

In this section, we compare our work to related works on handling concept drift in supervised learning and the more specific concept of environment drift in reinforcement learning.

A. Handling Concept Drift in Supervised Learning

In general, in supervised learning and especially in the context of Machine Learning Operations (MLOps), where production deployment and monitoring are essential, observing and handling concept drift is crucial for maintaining the performance and reliability of machine learning models in production. MLOps is a set of practices that aims to deploy and maintain machine learning models in production reliably and efficiently [2]. In supervised learning, concept drift refers to the changes in the relationship between the input data and the dependent variables [5], which can be denoted by the conditional distribution $P(y|x)$, where x is the input and y is the label. Most Concept Drift Detection works are based on tracking the prediction error rate of the machine learning models [6]. That is, the error rate of the models will increase if concept drift happens. Gamma et al. [7] proposed the Drift Detection Method (DDM), which monitors the online error rate of a model and raises the alarm when statistically significant changes are detected. Baena-García et al. [8] proposed the Early Drift Detection Method (EDDM), focusing on detecting gradual drifts by monitoring the distance between classification errors. Frias-Blanco et al. [9] developed HDDM, which leverages Hoeffding's bounds with a moving weighted average test to detect changes in the distribution of data streams. Bifet et al. [10] introduced ADWIN, which maintains an adaptive sliding window for detecting the changes in the data stream. Raab et al. [11] used the Kolmogorov-Smirnov (KS) statistical test [12] to detect concept changes. All of these error rate-based methods require access to the ground-truth labels. To overcome this limitation, Baier et al. [13] developed Uncertainty Drift Detection (UDD), which firstly estimates the uncertainty of deep neural network with Monte Carlo Dropout, and then ADWIN [10] is used on the uncertainty estimates to detect concept drift. Unlike these works for concept drift detection in supervised learning, our work focuses on detecting environment drift in reinforcement learning.

B. Handling Environment Drift in Reinforcement Learning

Compared with supervised learning tasks, MLOps for reinforcement learning (RL) tasks have not been sufficiently studied. Li et al. [14] proposed the concept of reinforcement learning operations (RLOps), focusing on the principles to deliver RL policies to the industry.

The term corresponding to concept drift in RL is environment drift [3], which refers to the change in the underlying environment dynamics, denoted by $P(S_{t+1}|S_t, A_t)$. Environment drift can impact the RL agent's production performance. In the context of RLOps, detecting and adapting to environment drift is crucial for maintaining the effectiveness of RL agents in dynamic and evolving environments. Despite its significance, only a few works studied environment drift in RL tasks.

Wang et al. [3] used a detector-agent to detect environment drift by exploring the new environment with equal probability, and incremental learning was used to update the Q function to make it adapt to the drifted environment. This work has some obvious limitations: (1) Using a detector agent to explore the environment in a production scenario is unrealistic, and the detector agent can only explore a finite state space, which makes this approach unsuitable for infinite state space. (2) The definition of a drifted environment in this work is for a deterministic environment, not for a stochastic environment. (3) The incremental learning approach is designed for table-based Q-learning. Therefore, it might not be suitable for other RL algorithms.

Greenberg et al. [15] proposed a method for detecting the deterioration of episodic performance, which can serve as an indicator of environment drift. This method monitors the episodic returns of the RL agent and raises alarms when significant declines are detected, suggesting potential changes in the environment. While this approach helps identify environment drift, its application is limited in realistic production environments where episodes are not clearly defined. In many real-world scenarios, interactions are continuous and do not naturally segment into discrete episodes, making it challenging to apply episodic performance-based methods. Therefore, more general approaches that do not rely on episodic structures are needed to effectively detect and manage environment drift in diverse and continuous production environments.

Compared with these works, our approach can detect drifts for RL environments with infinite state space, continuous action space, and stochastic dynamics in the unit of steps, and our approach is independent of the RL policies.

Some works model environment drift detection as out-of-distribution detection problems. Danesh et al. [16] and Haider et al. [17] used neural networks to predict the next states, and the prediction error was used as an indicator for drifts. Compared with these works, our approach applied Gaussian Processes, which are simpler in model structure and interpretability and easier to tune.

IV. Methodology

This study introduces a novel method for online detecting drifts in reinforcement learning environments through a two-stage process. Initially, we employ a Gaussian Process (GP) to predict the current actions based on states, i.e., $P(A_t|S_t, \Delta S_t)$, where A_t denotes the action at time t, S_t is the current state, and ΔS_t represents the state change. This probabilistic model enables us to estimate the likely actions given the observed state changes. We then compute the Mean Squared Error (MSE) between predicted and true actions observed. We assume that the MSE will remain relatively low in a stable environment where no drift occurs and will increase if environment drift occurs.

Subsequently, we leverage drift detection algorithms to monitor this MSE of GP over time. The primary objective of this stage is to identify drifts in the environment that might affect the RL agent's performance. These drifts are detected when the MSE significantly increases, suggesting changes in the underlying environment dynamics, i.e., $P(\Delta S_t|S_t, A_t)$. By continuously monitoring the MSE, our method provides a robust mechanism for detecting environment drift, thereby facilitating timely adjustments to the RL policy to maintain its accuracy and reliability.

Our approach can be regarded as applying a Gaussian Process to convert environment drift detection in reinforcement learning into error rate-based concept drift detection in supervised learning by monitoring the MSE of the Gaussian Process.

A. *Gaussian Process*

Our method for detecting environment drift is inspired by the work of PILCO [4], a model-based RL method that utilizes GP models to predict the next states given the current states and actions, i.e., $P(\Delta S_t|S_t, A_t)$, where $\Delta S_t \equiv S_{t+1} - S_t$. Instead of predicting $P(\Delta S_t|S_t, A_t)$, in our work, a GP predicting $P(A_t|S_t, \Delta S_t)$, which is trained in the training environment and then deployed in the production environment. The proof of $P(A_t|S_t, \Delta S_t) \propto P(\Delta S_t|S_t, A_t)$ when the policy π is fixed is shown in Equation 1.

Since in most reinforcement learning environments, the action space is smaller than the observation space, detecting the changes of $P(A_t|S_t, \Delta S_t)$ can suffer less curse of dimensionality [18] compared with that of $P(\Delta S_t|S_t, A_t)$. As dimensionality increases, the space volume increases exponentially, meaning data points become sparser. In high-dimensional spaces, the amount of data required to maintain the same density (or coverage) increases exponentially with the number of dimensions, and distance metrics such as Euclidean distance become less discriminative [19]. This phenomenon, known as distance distortion, makes distinguishing between different distributions or detecting small changes difficult. The relationship between distance distortion and the use of MSE of GP as an indicator to monitor environment drift is critical. In high-dimensional spaces, distance distortion can influence the calculation of MSE by making it less sensitive to changes. Although applying a GP predicting the action A_t given the current state S_t and the change of the current state ΔS_t could significantly increase the input dimensions when dealing with high-dimensional state space, dimensionality reduction techniques such as Principle Component Analysis (PCA) can be applied to mitigate this problem.

$$\begin{aligned} P(A_t|S_t, \Delta S_t) &= \frac{P(S_t, A_t, \Delta S_t)}{P(S_t, \Delta S_t)} \\ &= \frac{P(S_t)P(A_t|S_t)P(\Delta S_t|S_t, A_t)}{P(S_t, \Delta S_t)} \\ &= \frac{P(S_t)\pi(S_t)}{P(S_t, \Delta S_t)} P(\Delta S_t|S_t, A_t) \\ &\propto P(\Delta S_t|S_t, A_t) \end{aligned} \tag{1}$$

We apply the squared exponential kernel for the Gaussian process regression, the same method as in PILCO [4]. To further enhance the scalability and efficiency of our method, we employ Stochastic Variational Gaussian Processes (SVGPs) [20] instead of Exact GPs. SVGPs offer several benefits that are particularly advantageous in the context of large datasets:

1) **Scalability:** Exact GPs have a computational complexity of $O(n^3)$ due to the inversion of the covariance matrix, where n is the number of data points. SVGPs, on the other hand, use inducing points and variational inference to approximate the GP posterior, significantly reducing the computational burden to $O(m^2n)$, where m is the number of inducing points and $m \ll n$.
2) **Handling Large Datasets:** By leveraging stochastic optimization techniques, SVGPs can handle mini-batches of data during training, making it feasible to train on large datasets that would be prohibitive for Exact GPs. This is particularly beneficial in reinforcement learning, where policy training usually requires many interactions with the environment. Utilizing SVGPs allows us to effectively manage and learn from the substantial transition data generated through these interactions.

B. *Drift Detection Algorithms*

We employ several widely used drift detection algorithms to detect environment drift effectively, each with unique strengths. These algorithms help us monitor changes in the underlying distribution of MSE between the predicted actions $\hat{A}_t$ by the Gaussian Process and the true actions A_t, allowing for online detection of drifts in the RL environments that could affect the performance of RL policies. The drift detection algorithms used in this work include Probability Cumulative Sum (Prob CUSUM) [21], Page-Hinkley [22], Adaptive Windowing (ADWIN) [10], and Kolmogorov-Smirnov Windowing (KSWIN) [11]. These four drift detection algorithms are particularly suitable for detecting drifts in real-valued data, unlike other algorithms such as Drift Detection Method (DDM) [7] and Hoeffding's Drift Detection Method (HDDM) [9], which are primarily designed for categorical data.

V. Experiments

In this section, we evaluate the effectiveness of our proposed method for detecting environment drift in reinforcement learning (RL) environments. Before presenting the results, we first discuss the environments and RL policies used in our experiments, the evaluation metrics employed to access performance, the baseline methods for comparison, and the experimental setup.

A. *Environments and RL Policies*

We employed four distinct RL environments, each with specific RL policies to evaluate the effectiveness of our proposed method for detecting environment drift.

a) Pendulum with SAC: The Pendulum environment involves a single pendulum that swings around, to balance it upright. We utilized the Soft Actor-Critic (SAC) [23] algorithm, a state-of-the-art off-policy method known for its stability and efficiency in continuous action spaces. The trained RL policy can be found at HuggingFace.[1]

b) MountainCarContinuous with SAC: MountainCar is a classic RL problem where an underpowered car must drive up a steep hill. The continuous version of this environment is used in the Experiments with the SAC policy. The trained RL policy can be found at HuggingFace.[2]

c) LunarLanderContinuous with PPO: LunarLander is a simulated environment where the agent controls a lander to touch down on the moon's surface safely. In our experiments, we set the action space to continuous. We used the Proximal Policy Optimization (PPO) algorithm [24] for this environment. The trained RL policy can be found at HuggingFace.[3]

d) Reacher with PPO: The Reacher environment involves controlling a robotic arm to reach a target location. PPO was chosen for this environment. The trained RL policy can be found at HuggingFace.[4]

B. Evaluation Metrics

To evaluate the effectiveness of our proposed method for detecting environment drift, we used two metrics: false alarms (F) and delay (D). False alarms refer to instances where the drift detection method incorrectly predicts a drift when none has occurred. This metric is critical because a high rate of false alarms can lead to unnecessary interventions, such as retraining the model or adjusting policies, which can be costly and time-consuming. Delay measures the time the drift detection method takes to identify a true drift after it has occurred correctly. This metric is crucial because a shorter delay means the RL agent can respond more quickly to environmental changes, maintaining optimal performance. A longer delay, on the other hand, can result in prolonged periods of suboptimal performance and missed opportunities for timely interventions. To combine these two metrics, we use the data from the experiment results first to normalize false alarms and delay, to get the normalized metrics $\tilde{F}^e_{i,j}$ and $\tilde{D}^e_{i,j}$, shown in Equation 2 and Equation 3, respectively. This step gives the two metrics the same scales. e is the index of the experiment; i denotes the method i and j denotes the jth result of experiment e. μ_{F^e} and σ_{F^e} denote the mean and standard deviation of false alarms of the experiment e, respectively, and μ_{D^e} and σ_{D^e} denote the mean and standard deviation of delay of the experiment e, respectively.

$$\tilde{F}^e_{i,j} = \frac{F^e_{i,j} - \mu_{F^e}}{\sigma_{F^e}} \tag{2}$$

$$\tilde{D}^e_{i,j} = \frac{D^e_{i,j} - \mu_{D^e}}{\sigma_{D^e}} \tag{3}$$

[1]https://huggingface.co/sb3/sac-Pendulum-v1
[2]https://huggingface.co/sb3/sac-MountainCarContinuous-v0
[3]https://huggingface.co/sb3/ppo-LunarLanderContinuous-v2
[4]https://huggingface.co/fatcat22/ppo-reacher-v4

Then we combine the two normalized metrics $\tilde{F}^e_{i,j}$ and $\tilde{D}^e_{i,j}$ by computing the average of them to get a normalized score $S^e_{i,j}$, shown in Equation 4.

$$S^e_{i,j} = \frac{\tilde{F}^e_{i,j} + \tilde{D}^e_{i,j}}{2} \tag{4}$$

By normalizing and combining these metrics, we can evaluate the overall performance of the drift detection methods consistently and comprehensively, balancing the trade-offs between false alarms and detection delay.

C. Baselines

We will compare the results of our method with the following method baselines. All the baseline methods are followed by drift detection algorithms mentioned in Section IV-B, consistent with our method.

- **GP predicting states (GPState)**: A stochastic variational Gaussian Process (SVGP) is used to predict state transitions, i.e., $P(\Delta S_t|S_t, A_t)$, the same as the GP in the work of PILCO [4]. Drift detection algorithms monitor the MSE of the SVGP to detect environment drift. To compare our method with this baseline, we can demonstrate if our method can improve the detection of environment drift by changing predicting $P(\Delta S_t|S_t, A_t)$ to predicting $P(A_t|S_t, \Delta S_t)$.
- **Moving Average Step Rewards Distribution (AveStepRew)**: Many works monitor the performance of agents using the episode rewards, such as Mnih et al. [25] and Greenberg et al. [15]. Unlike training environments, where episodes are typically well-defined with a clear start and end, production environments may involve continuous tasks without explicit episodic boundaries. As a result, monitoring episode rewards becomes impractical, and in our work, we aim to detect environment drift in the unit of steps rather than episodes. To address this challenge, we estimate the moving average of step rewards to monitor overall performance, similar to episode rewards. The moving average could smooth out short-term fluctuations and provide a clearer picture of the agent's performance trends over time. This approach helps understand the long-term behaviour and stability of the RL agent in the production environment. We used two settings of the number of steps, 100 and 200, to compute the moving average step rewards in our experiments.

D. Experiment Settings

To simulate the environment drift, we slightly modify the parameters of each RL environment:

- **Pendulum**: The gravity parameter increased from the default value of 10.0 to 10.5.
- **MountainCarContinuous**: A leftward wind with a power of 0.35 was added.
- **LunarLander**: Wind was enabled, and the wind power was set to 5.

- **Reacher**: The motor gear parameter was decreased from 200 to 185.

The performance of the RL agents in undrifted and drifted environments are listed in Table I.

TABLE I
PERFORMANCE OF AGENTS IN UNDRIFTED AND DRIFTED ENVIRONMENTS

Env (Policy)	Undrifted Env	Drifted Env
Pendulum (SAC)	-143.44±81.23	-147.99±85.73
MountainCarContinuous (SAC)	94.65±1.10	93.76±2.07
LunarLander (PPO)	271.82±26.40	251.82±56.96
Reacher (PPO)	-4.41±1.32	-4.52±1.48

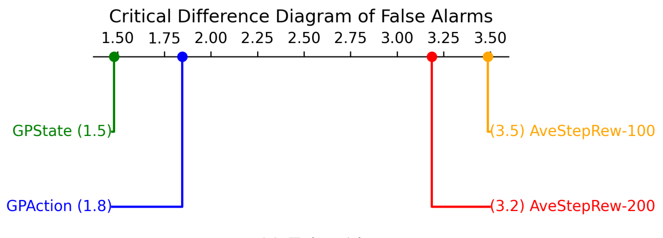
To train the stochastic variational Gaussian Process (SVGP) models, we collected transitions by running each training environment for 20,000 steps. The SVGP models are implemented with GPytorch [26]. To evaluate the effectiveness of our environment drift detection method and the baselines, we ran each production environment for 6000 steps. The environment drift was introduced at the 3000th step, allowing us to calculate the number of false alarms and the delay metrics. If the method fails to detect any drift after the 3000th step in one experiment, the delay will be set to 4000. The mean and standard deviation of the monitoring metrics, i.e., MSEs of GPs and moving average step rewards from the training environment, were used to normalize the monitoring metrics in the production environment. This normalization ensures that all monitoring metrics have the same scale, allowing us to use the same parameters for the drift detection algorithms on different monitoring metrics since these drift detection algorithms are sensitive to the scale of the monitoring metrics. The significance level of the Prob CUSUM is set to 0.05. For the other three drift detection algorithms, we used their default parameters as suggested by river[5]. In each environment, we train three GPs. Each GP will be tested ten times for each drift detection algorithm. Therefore, there will be 30 experiments for each drift detection algorithm in each environment. For the moving average step rewards baselines, we will test them 30 times, consistent with the 30 experiment results of the GP methods. Therefore, there will be 120 experiment results for each method in each environment.

The code of the experiments is available in this Github repository.[6]

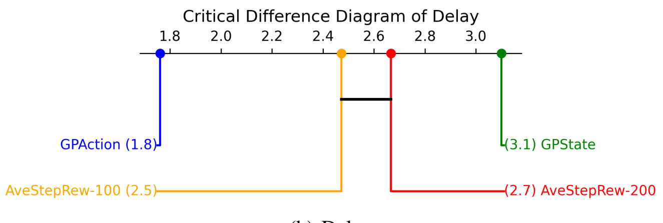
E. Results

We present the results of our experiments comparing the proposed method with the baseline methods for detecting environment drift in RL environments. The results are listed in Table III in Appendix A. According to Table III, in terms of false alarms, the baseline GPState outperforms other methods in most experiments. Both methods based on Gaussian process regression outperform the moving average step rewards-based methods. Regarding delay and normalized score, our method GPAction outperforms all the baselines.

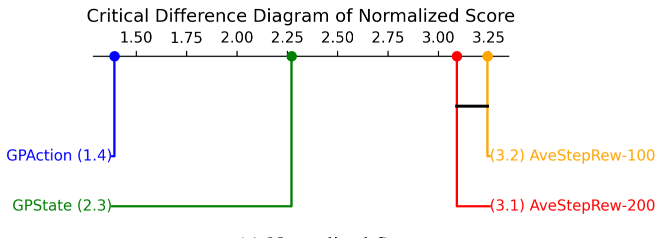
To compare the methods across multiple experiments from different environments, we applied the Friedman chi-square test [27] to check for significant differences between rank means. Subsequently, the Nemenyi post-hoc test [28] was used to identify significant pairwise differences in average ranks. The ranking of the methods is visualized by critical differences (CD) diagrams [28], with a significance level of 0.05, which is shown in Figure 1. The CD-diagram for false alarms in Figure 1a indicates that GPState has the lowest average rank (1.5). GPAction also performs well but slightly worse than GPState. The CD-diagram for delay shown in Figure 1b shows that GPAction achieves the best performance with the lowest average rank (1.8). This indicates that GPAction has the shortest delay in detecting environment drifts compared to other methods. GPState and the moving average step rewards methods (AveStepRew-100 and AveStepRew-200) have higher ranks, indicating longer delays. The CD-diagram for the normalized score shown in Figure 1c, which combines both false alarms and delay, shows that GPAction has the best overall performance with the lowest average rank (1.4). GPState is the second-best, followed by AveStepRew-100 and AveStepRew-200.



(a) False Alarms



(b) Delay



(c) Normalized Score

Fig. 1. CD-Diagrams of False Alarms, Delay and Normalized Score.

F. Ablation Study

This ablation study aims to delve deeper into comparing the effectiveness of GPAction and GPState in detecting environ-

[5]https://riverml.xyz/latest/

[6]https://github.com/fatcatZF/EnvironmentDriftGP

ment drift. By analyzing the Mean Squared Error (MSE) over 6000 steps in four environments—Pendulum, MountainCar-Continuous, LunarLander, and Reacher, we aim to determine if the ActionPredictor provides better distinguishability between undrifted and drifted states than the StatePredictor. We use the Area Under the Curve (AUC) [29] of the MSE plots as a measure of this distinguishability. Higher AUC values suggest greater sensitivity and effectiveness in detecting environment drift, indicating that the predictor can distinguish between undrifted and drifted states more effectively. The AUC is particularly beneficial as it is independent of the specific drift detection algorithms and their settings, providing an objective measure of the methods' sensitivity to environment drift. All MSEs are normalized based on the mean and standard deviation of the undrifted environment (first 3000 steps) to better align the plots, thus providing better visualization. The plots of the MSEs of GPAction and GPState of the four environments are shown in Figure 2. The AUC values are presented in Table II.

TABLE II
AUC VALUES FOR GPACTION AND GPSTATE ACROSS DIFFERENT ENVIRONMENTS

Environment	GPAction AUC	GPState AUC
Pendulum (SAC)	0.994±0.003	0.977±0.005
MountainCarContinuous (SAC)	0.999±0.001	0.737±0.012
LunarLander (PPO)	0.912±0.013	0.643±0.028
Reacher (PPO)	0.719±0.048	0.701±0.026

Based on the MSE plots and the AUC values presented in Table, GPAction consistently demonstrates superior performance in distinguishing between undrifted and drifted environments. The higher AUC values for GPAction, especially in environments like MountainCarContinuous and Pendulum, indicate greater sensitivity in detecting environment drift than GPState. These results are corroborated by the visual analysis of the MSE plots, where GPAction shows more significant changes post-drift. Thus, GPAction is a more reliable method for monitoring and detecting environment drift in reinforcement learning settings.

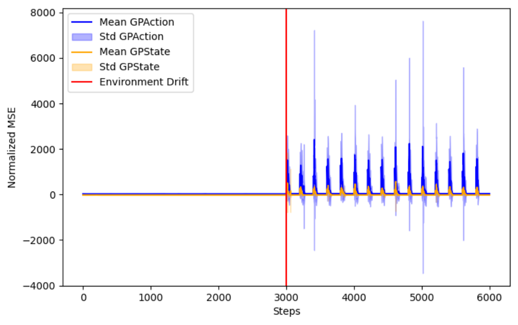
VI. Conclusion and Future Work

This study introduced a novel two-stage method for detecting environment drift in reinforcement learning environments by leveraging Gaussian Process Regression to predict actions. By comparing our method with three baselines across four environments, we demonstrated the effectiveness of our method in detecting environment drift.

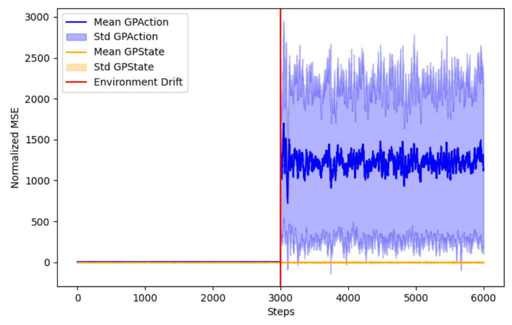
In future work, it is worth exploring suitable kernels in the Gaussian Process for detecting environment drift, how to extend our current method to the environments with discrete action space, and the software architecture aspects of environment drift detection through industrial case studies.

Acknowledgment

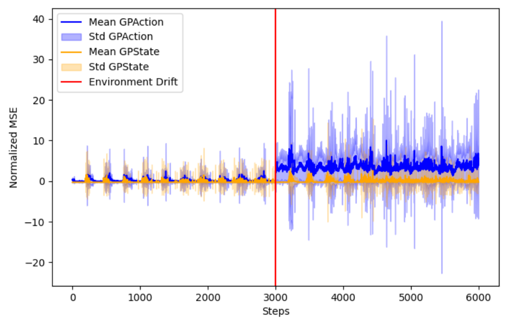
This work was supported by the FFG (Austrian Research Promotion Agency) project MODIS (no. FO999895431).



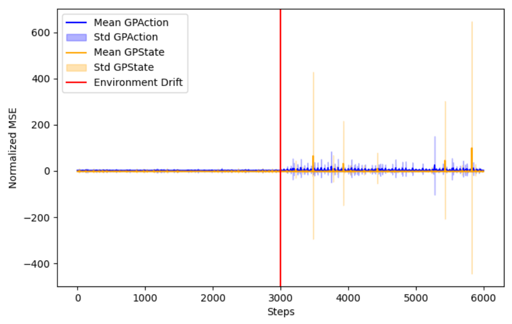
(a) Pendulum (SAC)



(b) MountainCarContinuous (SAC)



(c) LunarLander (PPO)



(d) Reacher (PPO)

Fig. 2. Mean and Standard Deviation of MSE for GPAction and GPState across different environments.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM computing surveys*, vol. 55, no. 6, pp. 1–29, 2022.

[3] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, "Incremental reinforcement learning with prioritized sweeping for dynamic environments," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 621–632, 2019.

[4] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[5] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[6] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[7] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection advances in artificial intelligence," in *Proc. of 17th Brazilian Symp. on Artificial Intelligence*, pp. 286–295.

[8] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, Citeseer, vol. 6, 2006, pp. 77–86.

[9] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2014.

[10] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*, SIAM, 2007, pp. 443–448.

[11] C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive soft prototype computing for concept drift streams," *Neurocomputing*, vol. 416, pp. 340–351, 2020.

[12] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[13] L. Baier, T. Schlör, J. Schöffer, and N. Kühl, "Detecting concept drift with neural network model uncertainty," *arXiv preprint arXiv:2107.01873*, 2021.

[14] P. Li, J. Thomas, X. Wang, *et al.*, "Rlops: Development life-cycle of reinforcement learning aided open ran," *IEEE Access*, vol. 10, pp. 113 808–113 826, 2022.

[15] I. Greenberg and S. Mannor, "Detecting rewards deterioration in episodic reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021, pp. 3842–3853.

[16] M. H. Danesh and A. Fern, "Out-of-distribution dynamics detection: Rl-relevant benchmarks and results," *arXiv preprint arXiv:2107.04982*, 2021.

[17] T. Haider, K. Roscher, F. Schmoeller da Roza, and S. Günnemann, "Out-of-distribution detection for reinforcement learning agents with probabilistic dynamics models," in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023, pp. 851–859.

[18] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[19] A. Zimek, E. Schubert, and H.-P. Kriegel, "A survey on unsupervised outlier detection in high-dimensional numerical data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012.

[20] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," *arXiv preprint arXiv:1309.6835*, 2013.

[21] Seitz, *Probabilistic cusum for change point detection*, Jan. 2024. [Online]. Available: https://sarem-seitz.com/posts/probabilistic-cusum-for-change-point-detection/.

[22] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[26] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, vol. 31, 2018.

[27] P. Sprent and N. C. Smeeton, *Applied nonparametric statistical methods*. CRC press, 2016.

[28] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine learning research*, vol. 7, pp. 1–30, 2006.

[29] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve.," *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

Appendix A
Tables

TABLE III
The tables below present the results of experiments across various environments and drift detection methods. Each Table presents the results from one environment, including the metrics for false alarms (F), delay (D), and the combined normalized score S for different methods. Each row of a table denotes one drift detection algorithm. GPAction and GPState denote the stochastic variational Gaussian Process predicting $P(A_t|S_t, \Delta S_t)$ and $P(DeltaS_t|S_t, A_t)$, respectively. AveStepRew-100 and AveStepRew-200 denote the moving average step rewards methods, which set the number of steps to 100 and 200, respectively. Smaller values indicate better performance. The results with the smallest mean in each row are highlighted in bold text.

	GPAction			GPState			AveStepRew-100			AveStepRew-200		
	F	D	S	F	D	S	F	D	S	F	D	S
ProbCUSUM	2.77±2.42	**1.9±5.35**	**-0.56±0.08**	**2.57±2.94**	2.57±6.74	**-0.56±0.1**	12.57±13.25	2055.77±1765.74	0.43±0.54	21.27±20.62	1927.87±1762.44	0.69±0.68
PageHinkley	4.53±2.56	**0.87±2.15**	-0.43±0.44	**2.9±1.83**	1.63±4.72	**-0.71±0.32**	8.8±1.4	211.4±159.33	0.86±0.47	3.73±1.08	320.3±204.08	0.28±0.52
ADWIN	0.57±0.94	50.73±39.85	-0.58±0.18	**0.4±0.5**	**49.67±48.52**	**-0.6±0.16**	11.97±2	83.8±84.29	0.67±0.27	4.37±2.22	279±242.8	0.51±0.78
KSWIN	**5.73±3.65**	35.03±18.09	**-0.76±0.38**	9.87±2.13	**16.23±10.51**	-0.38±0.23	13.3±1.47	134.03±72.84	0.36±0.3	11.7±6.27	311.17±187.05	0.77±0.75

(a) Pendulum (SAC)

	GPAction			GPState			AveStepRew-100			AveStepRew-200		
	F	D	S	F	D	S	F	D	S	F	D	S
ProbCUSUM	0.67±1.15	**2.83±9.02**	**-0.66±0.05**	**0.57±0.77**	1370.37±1137.19	-0.19±0.39	9.37±8.13	1722.87±1713.3	0.32±0.52	16.07±16.08	1444.23±1596.31	0.53±0.67
PageHinkley	2.43±2.62	**0.53±2.92**	**-0.67±0.37**	**0.67±0.8**	395.67±245.37	-0.01±0.58	7.67±2.02	163.27±177.93	0.45±0.49	7.07±1.53	108.13±114.39	0.23±0.35
ADWIN	0.83±0.99	**28.33±15.34**	**-0.65±0.09**	**0±0**	4000±0	0.45±1.69	9.93±5.55	150.9±206.77	0.2±0.47	7.8±4.77	131.8±138.94	0.01±0.42
KSWIN	3.53±2.37	**40.63±33.78**	**-0.41±0.2**	**1.33±1.24**	874.93±842.16	0.16±0.77	8.67±7.41	146.6±214.93	0.11±0.6	9.3±6.29	123.47±207.81	0.14±0.54

(b) MountainCarContinuous (SAC)

	GPAction			GPState			AveStepRew-100			AveStepRew-200		
	F	D	S	F	D	S	F	D	S	F	D	S
ProbCUSUM	2.37±4.17	**267.8±208.09**	**-0.54±0.14**	**0.77±1.59**	2064.3±1316.3	0.05±0.48	9.17±10.69	1500.27±1625.72	0.16±0.54	21.27±17.8	738.87±1194.48	0.34±0.65
PageHinkley	2.93±1.68	**56.5±53**	**-0.61±0.21**	**0.47±0.68**	836.23±510.93	0.01±0.6	9.2±0.81	174.27±205.84	0.33±0.29	8.77±1.33	174.47±190.11	0.27±0.3
ADWIN	1.67±1.24	145.57±179.38	**-0.64±0.12**	**0.13±0.35**	2844.9±1405.05	0.19±0.5	11.03±1.4	107.27±68.13	0.22±0.13	11.23±2.28	**99.8±73.91**	0.23±0.22
KSWIN	4.47±1.24	86.7±105.74	**-0.416±0.12**	**0.6±0.89**	1003.83±1128.2	-0.09±0.8	17.83±1.76	**42.97±44.46**	0.43±0.12	13.23±3.38	70±59.42	0.12±0.23

(c) LunarLander (PPO)

	GPAction			GPState			AveStepRew-100			AveStepRew-200		
	F	D	S	F	D	S	F	D	S	F	D	S
ProbCUSUM	2±2.51	**405.43±423.92**	**-0.66±0.12**	**1.77±4.47**	2581.37±1709.06	-0.05±0.44	10.6±14.88	2487.83±1676.69	0.23±0.63	15.3±22.32	2839.83±1664.72	0.49±0.8
PageHinkley	1.67±2.01	**214±346.37**	**-0.55±0.4**	**1.2±1.21**	1872.9±1572.34	0.13±0.73	6.33±1.56	199.73±186.81	0.35±0.29	4.5±0.86	341.93±236.12	0.06±0.2
ADWIN	0.03±0.18	2397.33±1638.44	**-0.25±0.45**	**0±0**	3820.2±684.36	0.14±0.19	10.73±2.89	**160.7±183.63**	0.12±0.26	9.2±2.62	222.23±202.95	-0.01±0.25
KSWIN	3.47±3.27	225.8±327.8	-0.42±0.41	**0.97±1.19**	772.53±885.39	**-0.91±0.03**	14.33±2.71	**136.43±146.62**	0.3±0.24	13.17±5.41	164.63±182.72	0.24±0.45

(d) Reacher (PPO)