Proceedings of the International Conference on
Formal Methods and Foundations of Artificial Intelligence
Eszterházy Károly Catholic University
Eger, Hungary, June 5–7, 2025
pp. 161–173
DOI: 10.17048/fmfai.2025.161

# Fundamental limitations of online supervised learning in dynamic control loops*

**István Pintye**[ab], **József Kovács**[bc], **Róbert Lovas**[ab]

[a]Institute for Computer Science and Control, Hungarian Research Network
istvan.pintye@sztaki.hun-ren.hu

[b]Institute for Cyber-Physical Systems, John von Neumann Faculty of Informatics, Óbuda University
robert.lovas@sztaki.hun-ren.hu

[c]School of Computer Science and Engineering, University of Westminster
jozsef.kovacs@sztaki.hun-ren.hu

**Abstract.** In conventional supervised learning of neural networks, training samples are selected either randomly or in a predefined order, assuming independence across samples. This paper diverges from that setting by embedding the learning process within a dynamic control system. Specifically, we consider a discrete-time control system where the output is given by a nonlinear mapping, that dynamically adjusts the number of virtual machines (VMs) based on workload characteristics such as CPU, memory, and network usage. The system's output is determined by a neural network that estimates the deviation from a target utilization profile.

In online supervised learning embedded in feedback control, data generation is shaped by model performance, leading to a narrowing of the observed input distribution over time. This self-induced sampling bias may reduce model robustness, stability and adaptability. We demonstrate that simple periodic perturbations to the VM allocation process act as an effective form of regularization, improving learning robustness without relying on external reward or replay mechanisms. Unlike traditional approaches using fixed training sets, in our formulation the system operates online where at each time step, multiple candidate control inputs $u[k] \in \mathcal{U}$ are evaluated continuously and each yielding a predicted output $y[k] = f(x[k])$. At each step, the

*The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101131207 (GreenDIGIT). This work is partially funded by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

controller selects the action that minimizes the predicted deviation from the desired reference, which then determines the next state $x[k+1]$ and yields the next training sample for the neural network. As a result the learning trajectory is not predetermined but is dynamically created by the controller's actions, which depend on the network's current predictions. We present how this closed-loop interaction between prediction and sample selection influences learning stability, convergence, and input space coverage in an online setting.

*Keywords:* online learning, closed-loop control, neural networks, cloud resource allocation, distributional shift, adaptive systems

# 1. Introduction

In real-time control systems, the quality of a neural network's predictions depends not only on its internal structure and learning parameters, but also on the data it receives during training - by the trajectory of data it encounters - a trajectory shaped dynamically by its own predictions.

Control decisions are made by evaluating future system states using the current neural network model, thereby creating a prediction-driven data generation process. This feedback loop between learning and decision-making creates a unique situation where the data used for learning is not independent from the learning itself. As the model improves and control stabilizes, the diversity of training samples naturally decreases. The learning agent spends more time in well-regulated regimes, causing the network to adapt to a narrow region of the state space. This effect, which we refer to as distributional narrowing, can reduce generalization and lead the model fragile to noise, drift, or unexpected dynamics when conditions change. This phenomenon can also be viewed as a case of closed-loop covariate shift: predictions influence actions, actions influence state transitions, and the resulting data distribution becomes endogenously biased by the controller's current policy. We adopt this term to emphasize that sample selection is not external, but induced by the model-in-the-loop.

To illustrate this, we use a simulation where a system dynamically adjusts the number of active virtual machines (VMs) in response to changing workload conditions. The system receives inputs such as CPU utilization, RAM usage, and network load, and uses a neural network to predict how many VMs are needed to maintain balanced resource usage. Control decisions based on these predictions directly influence future input states, thereby closing the loop between learning and decision-making.

Our key observation is that this learning loop can be both a strength and a weakness. On one hand, the model adapts to the system behavior it helps regulate. On the other hand, it may overfit to narrow workload patterns, leading to unstable decisions under unexpected load changes. To address this, we test a simple idea: periodically apply artificial perturbations to the VM allocation decisions. These forced adjustments expose the system to underrepresented operating

regimes, increasing data diversity and improving learning robustness.

To validate these ideas, we implement a simulated resource management task in which a neural network regulates the number of virtual machines under time-varying load conditions. The task involves tracking target utilization ranges while adapting to workload dynamics. We evaluate how combinations of model complexity and artificial exploration affect learning performance. Even in a fully deterministic environment, the system's behavior varies widely depending on the diversity of training experiences encountered during learning.

In adaptive control systems, the learning process and the controller's decision-making are often interdependent. While classical supervised models assume fixed training data and offline learning, real-time systems operate under evolving conditions that require continuous adaptation. In this study, we integrate online supervised learning into a feedback-controlled system in which the learning model directly influences, and is influenced by, system behavior. Specifically, we consider a discrete-time nonlinear state-space system of the form: $y[k] = f(x[k], u[k])$, where $f(\cdot)$ is a neural network approximator trained online. Unlike classical linear systems expressed as $y[k] = Cx[k] + Du[k]$, the output here is determined by a learned nonlinear function of the current system state $x[k]$. At each time step, a finite set of candidate control actions $u \in \mathcal{U}$ is evaluated. These candidates yield different state transitions and predicted outputs. The controller selects the action that minimizes the deviation from a predefined target $y_{\text{ref}}[k]$:

$$u^*[k] = \arg\min_{u \in \mathcal{U}} \|f(x[k]; u[k]) - y_{\text{ref}}[k]\|.$$

This feedback mechanism works as the model's current performance affects which training data will be observed next, closing the loop between prediction and future training input. The system is thus self-organizing in terms of data generation and the network continuously adapts to the evolving input-output trajectory.

Through extensive simulations, we show that periodically forcing the agent into previously unseen and suboptimal regions of the state space-rather than allowing it to self-regulate exclusively based on its current policy can improve the learning process. This artificial perturbation mechanism cause greater sample diversity which leads to more robust and generalizable neural network predictions in VM allocation. We demonstrate that the same system, exposed to the same environment, may yield lower cumulative tracking errors depending on the external perturbation and exhibit more stable learning behavior compared to purely autonomous learning mechanism. As we later demonstrate, these results point to the importance of actively managing the learning process state space exploration during learning. In contrast to traditional approaches that focus on dynamic or adaptive hyperparameter optimization, we emphasize the role of sample diversity and input space coverage as one of the key determinant of control performance.

## 2. Related work

Much prior work in supervised learning has investigated the effect of sample selection and ordering on convergence. Curriculum learning, hard negative mining, and

stratified sampling methods are some well-known examples for these. But these typically rely on predefined sampling strategies to improve convergence speed or model robustness [1, 15]. Bouchard et al. introduce an online approach to dynamically select training examples based on the current model state, improving learning efficiency in evolving data contexts [2], which is conceptually related to our closed-loop sample selection.

General surveys such as Soviany et al. [11] provide a taxonomy of curriculum learning methods, including self-paced and RL-guided curricula, while Narvekar et al. [10] frame curriculum generation in reinforcement learning domains as a formal, adaptive process suitable for sequential decision-making settings. The curriculum learning paradigm itself has been more deeply explored in recent work. Hacohen and Weinshall (2019) systematically examine how scoring and pacing strategies affect deep network convergence, showing substantial gains in both accuracy and training speed via curriculum schedules [6]. Graves et al. (2017) propose an automated curriculum learning framework where a multi-armed bandit selects training samples to maximize learning progress, significantly accelerating learning in sequence modeling tasks [5]. Parallels our perturbation-based approach to diversify training trajectories Liu et al. actively controls the sample order to improve convergence of learning [8].

In online learning and streaming data settings, concept drift adaptation has been extensively studied. Gama et al. presented a comprehensive survey on concept drift adaptation methodologies, including ensemble and drift detection techniques [4]. Elwell and Polikar proposed incremental ensemble methods for nonstationary environments that dynamically adapt to drift [3]. Wang et al. explored mining ensemble classifiers for evolving data streams in KDD-2003 [12]. Losing et al. introduced a dual-memory architecture to manage diverse drift types in real-time systems [9].

While reinforcement learning naturally addresses closed-loop feedback dynamics and curriculum generation in RL domains has been formally framed as an adaptive process suitable for sequential decision-making [10], fewer studies [11] consider purely supervised settings where the training data distribution is influenced by the model's own predictions. Our work focuses on these feedback-coupled systems without relying on reward signals or repeated training episodes. It aligns with adaptive control and online learning under concept drift, but prioritizes one-pass supervised updates in continuous feedback systems. These works support the view that careful ordering of training examples can act as a powerful regularizer in dynamic systems.

# 3. Motivation

Traditional supervised learning often assumes that samples are drawn i.i.d. from a static distribution. However, in real-time systems where a model is embedded within a controller, this assumption no longer holds. Here, the learning process affects the system's state evolution, and in turn, the system determines which

samples the model sees next. In contrast, when a model is used within a control loop, the input data becomes dependent on the model's predictions. Each decision influences the system's next state, which in turn determines the next input to the model.

This feedback-induced data generation process creates both a challenge and an opportunity: model predictions directly influence control actions and resource allocation, which in turn shapes future input observations such as CPU, RAM, and network utilization. In other words, the model does not only passively receive and learns from data but also actively contributes to generating it.

This feedback structure creates a coupling between the model's estimation and the trajectory of training samples. A sequence of high-error predictions can move the system into rarely visited regions of the state space, where the model has limited prior exposure. On the other hand, consistently low-error predictions may constrain the system's behavior to a limited subset of states. In both cases, the data distribution becomes non-representative of the broader task space.

Over time, this interaction may cause the learning process to become biased toward a narrow region of the state space, making the model fragile to sensor drift, unexpected conditions, or shifts in workload dynamics. Since online learning systems typically process each sample only once and offer no opportunity to reset or rebalance the training sequence, such distributional biases cannot be corrected retrospectively.

We propose to investigate whether artificially introducing diversity during training improves performance. By occasionally overriding the model's predicted control decision with a random alternative, the system is exposed to unfamiliar states that would otherwise remain unvisited. This forced control input changes the system's trajectory and results in exposure to input regions that would otherwise be excluded. This forced exploration is expected to increase input coverage, potentially improving stability and generalization, particularly in higher-capacity models. Our goal is to evaluate how such a modification affects the convergence, generalization, and stability of the learning process under different model capacities.

The central motivation of this work is to understand how this coupling influences: (1) the speed and stability of convergence, (2) the diversity of encountered workload regimes, and (3) the robustness of learning under dynamically shifting data distributions. Unlike curriculum learning or sample-weighting strategies that assume external control over data order, our method embeds sampling decisions within the system dynamics and controller behavior. This fully endogenous sample selection process also presented by [14] illustrating the value of dynamic sampling policies under evolving data distributions.

# 4. System model

To explore the learning dynamics discussed above, we employ a simplified simulation model which captures the essential features of real-time online learning in control systems. We consider a discrete-time control system (in which) where

an agent (e.g., a simplified autoscaler) dynamically adjusts the number of virtual machines (VMs) in response to synthetic workload signals generated along a time-varying pattern with increasing frequency and amplitude. The system dynamics are modeled in two parts: a linear state transition model and a nonlinear output approximation via a neural network. This workload stresses the spectral resolution of the estimator. The smaller MLP (5, 3) tends to underfit higher-frequency components, while the larger MLP (10, 5) has sufficient capacity to fit them but, in autonomous mode, may over-specialize to frequently visited regimes. Periodic perturbation exposes rarer, high-frequency regions, improving coverage and stabilizing learning, consistent with our capacity proxy $101^2$ vs. $42^2$.

Let the state vector at time $k$ be denoted as $x[k] \in \mathbb{R}^n$, which consists of the current sensor readings (e.g., CPU usage, RAM usage, Network usage). The control input $u[k] \in \{-10, -9, \ldots, 10\}$ represents the agent's discrete VM number adjustment at each time step. The state transition is modeled by the linear system:

$$x[k+1] = Ax[k] + Bu[k], \tag{4.1}$$

where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^n$ are system matrices that approximate how the state evolves based on previous measurements and applied input.

The output variable $y[k] \in \mathbb{R}$ represents the deviation of the current system load state from the desired utilization target (e.g., balanced CPU and network usage), it is estimated through a neural network mapping the predicted state to a scalar value:

$$y[k] = f(x[k]; \theta), \tag{4.2}$$

where $f(\cdot; \theta)$ is a fully connected feedforward neural network with parameters $\theta$, trained online using incoming sensor data and revealed ground truth offsets. The primary goal of the control system is to maintain resource usage close to a predefined reference profile, minimizing deviations from the target CPU and network utilization levels. Both the neural network parameters $\theta$ as well as the linear transition matrices $A$ and $B$ are adapted continuously during execution. The matrices $A$ and $B$ are estimated online using least-squares regression over observed pairs of consecutive system states and applied control inputs.

At each time step, all possible control inputs $u \in \mathcal{U} = \{-10, \ldots, 10\}$ are evaluated by simulating the next state and passed through the neural network to predict the resulting deviation. At each step, all possible actions are simulated using the previously defined dynamics in equations (4.1) and (4.2), to evaluate the predicted next state and output deviation. The control input $u^*[k]$ is selected to minimize the distance from the reference:

$$u^*[k] = \arg\min_{u \in \mathcal{U}} \|\tilde{y}[k; u] - y_{\text{ref}}[k]\|. \tag{4.3}$$

This makes the data-generating process dependent on the model's own predictions, creating a self-reinforcing learning trajectory. After each control decision, the system also transitions to a new state $x[k+1]$, determined by the internal dynamics (which is approximated), and a new data pair $(x[k], y[k])$ is added to the

training set. After executing $u^*[k]$, the true output error is revealed as the actual deviation of the system from the target utilization, computed based on workload indicators such as CPU or network saturation, denoted $y_{\text{true}}[k]$. This value is used to update the neural network via online stochastic gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(f(x[k]; \theta), y_{\text{true}}[k]),$$

where $\theta$ are the neural network parameters, $\eta$ is the learning rate set to 0.01, and $\mathcal{L}$ is the loss function (in this case mean squared error).

As we can see this procedure sequence creates a unique learning loop. Model predictions influence control actions, which influence state transitions, which then determine future training data. The interplay of learning and control thus forms an implicit curriculum, driven not by external design but by internal performance.

In this setup, the learning rate $\eta$ is not tuned during operation and is fixed throughout the entire run. Its value strongly affects both the convergence speed and stability of the prediction model and indirectly, the control behavior of the agent. Since no replay buffer or offline tuning is available, the system's learning trajectory is entirely shaped by the chosen hyperparameters at initialization. But this study does not aim to optimize learning rate dynamics. Once a suitable learning rate was empirically selected, it remained fixed throughout all training runs, allowing us to isolate the effect of perturbations and model capacity on learning behavior.

We investigate: 1. How this learning process converges toward an optimal solution over time. 2. Whether it enables more efficient exploration of the state space. 3. The trade-offs between control accuracy and learning robustness under online adaptation. 4. Whether the learning process becomes more stable when the agent is periodically forced into previously unseen states. For example by artificially perturbing the VM scaling decisions at regular intervals to improve generalization

# 5. Methods

## 5.1. Simulation environment

To investigate the online learning behavior of a closed-loop neural controller, we developed a simulation environment in Python using the Pytorch framework. The environment models a simplified cloud infrastructure where a controller dynamically adjusts the number of active virtual machines (VMs) in response to changing resource demands. The simulated system operates under a synthetic workload pattern that varies over time, mimicking real-world fluctuations in service usage. At each discrete time step $k$, the agent observes current resource utilization and selects a control input $u[k] \in \{-10, ..., 10\}$, which corresponds to scaling the VM pool up or down by discrete steps. At every time step, the agent receives three resource utilization metrics: CPU utilization percentage, RAM (memory) utilization percentage and Network bandwidth usage. These readings are normalized and form the input state vector:

$$x[k] = \begin{bmatrix} \text{CPU}[k] & \text{RAM}[k] & \text{NET}[k] \end{bmatrix}^\top,$$

which serves as input both for the control dynamics (see Equation (4.1)) and the neural network output estimation (see Equation (4.2)).

### 5.2. Training procedure

At each time step, the control loop proceeds as follows:

1. The current state vector $x[k]$ is recorded from system metrics.
2. For each candidate scaling action $u \in \mathcal{U}$:
   (a) Simulate the next system state $\tilde{x}[k+1;u] = Ax[k] + Bu$, reflecting how scaling the VM pool affects resource usage.
   (b) Estimate the deviation from the target operational balance via $y = f(\tilde{x}[k+1;u])$.
3. Choose $u^*[k]$ that minimizes the deviation from the desired output $y_{\text{ref}} = 0$.
4. Apply $u^*[k]$, observe the resulting true deviation $y_{\text{true}}[k]$, and transition to the next state $x[k+1]$.
5. Update the neural network via online stochastic gradient descent. The linear dynamics matrices $A$ and $B$ are also updated using least-squares regression to approximate the effect of control actions on future state transitions.

The neural network function $f(\cdot)$ is modeled as a fully connected feedforward neural network with hyperbolic tangent activations. It maps the current three-dimensional system state $x[k]$ to a scalar estimate of the deviation from balanced resource usage $y$. For a two-hidden-layer MLP with hidden sizes $(h_1, h_2)$, input dimension $d = 3$, and scalar output $m = 1$, the number of trainable parameters is $W = dh_1 + h_1 + h_1h_2 + h_2 + h_2m + m$. In our two configurations this gives: (10, 5) $\rightarrow W = 101$; (5, 3) $\rightarrow W = 42$. For smooth activations such as tanh, classical VC/pseudodimension bounds scale polynomially in $W$; in practice we report $W^2$ as a coarse capacity proxy: $101^2 = 10{,}201$ vs. $42^2 = 1{,}764$. We use these values only to compare relative capacity between settings and capacity alone does not guarantee online generalization in closed-loop learning. The loss is defined as the squared error between predicted and true deviation: $\mathcal{L}(f(x[k]), y_{\text{true}}[k]) = (f(x[k]) - y_{\text{true}}[k])^2$.

Learning occurs online – one sample at a time – without replay buffers or mini-batching. The system operates under a one-pass constraint, reflecting real-world adaptive control environments where replay is infeasible.

## 6. Experiments

As defined in Equation (4.3) in our simulated environment, the system adjusts the number of active virtual machines (VMs) based on current workload indicators: CPU utilization, RAM usage, and network traffic. At each time step, several candidate VM counts $u[k] \in \mathcal{U}$ are evaluated using the neural network model $f(x[k], u[k])$, which predicts the deviation from a target utilization profile. The system selects the VM count that minimizes this predicted deviation (see Equation (4.3)).

To investigate the role of extra perturbation, we conduct multiple simulation runs with different random seeds across identical environments. The primary control objective is to maintain a balanced utilization profile across CPU and network resources. The true deviation from this balance is denoted by $y_{\text{true}}[k]$, which reflects the difference between observed and target utilization ratios. To evaluate learning performance, we conducted multiple simulation runs under each experimental condition and measured cumulative tracking error, the sum of absolute deviations from the target resource utilization profile over time. At each timestep $k$, the agent's distance from the optimal trajectory is measured and added to the total error. We report both the mean and standard deviation of this metric across runs. Formally, this is defined as $E^{(i)}_{\text{track}} = sum^{T}_{k=1}\left|y^{(i)}_{\text{true}}[k] - y_{\text{ref}}[k]\right|$, and the summary statistics are computed as

$$\mu_{\text{track}} = \frac{1}{N}\sum_{i=1}^{N} E^{(i)}_{\text{track}} \quad \text{and} \quad \sigma_{\text{track}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(E^{(i)}_{\text{track}} - \mu_{\text{track}}\right)^2}.$$

These metrics provide insight into both the control accuracy (via tracking error) and the learning performance of the model (via prediction error), enabling a robust comparison between the perturbed and non-perturbed training regimes.

# 7. Results

To further investigate the interplay between model complexity and input space diversity in online neural control, we conducted a set of four experiments combining two architectural settings and two training modes. The neural network used to estimate the system's deviation from target resource balance was configured with either a smaller MLP (5, 3) or a larger MLP (10, 5) structure. Each configuration was trained under one of two regimes. (A.) Autonomous control, where the agent always selected the action predicted to be optimal. (B.) Artificially disturbed control, where every third decision step (from time 0 to 1500) was randomly overridden with a uniformly sampled control value from [-10, +10], while all other steps followed the prediction-based policy. We study pre/during/post phases to assess degradation after perturbation removal. Beyond "every 3 steps (0–1500)", we evaluate schedules with periods $\{2, 5\}$ and truncated windows $[0, T_0]$ with $T_0 \in \{800, 1200\}$. Denser schedules increase coverage but may introduce short-term bias; sparser schedules yield smaller bias but less variance reduction. We report median/mean cumulative error and failure rate across seeds to locate a practical trade-off.

This 2×2 setup allowed us to test the following hypotheses: (1.) The smaller network (5,3) will underperform in tracking accuracy regardless of disturbance, due to limited expressive power. (2.) The larger model (10,5) will achieve better accuracy but may exhibit instability and divergence during training. (3.) Artificial perturbation will not help the smaller network due to its limited capacity, and might degrade convergence. (4.) Larger models would benefit from exploration,

improving stability and generalization. Each experimental condition was evaluated over 100 independent simulation runs, with only the random initialization of network weights varied. The results summarize the cumulative tracking error during the test sequence. For better understanding first we present the comparision of Configuration 3 and Configuration 4 results. Instead of visualizing each trajectory individually of each run, we show the mean resource adjustment trajectory along with one standard deviation over time, providing an aggregated view of tracking performance and learning stability.

Figure 1 shows the agent's deviation from the desired operational target across multiple independent simulation runs. The results demonstrate that artificial exploration reduces variance and improves stability, especially for larger models. In contrast, autonomous learning without perturbation leads to greater performance divergence. Smaller networks show capacity limitations in all scenarios.

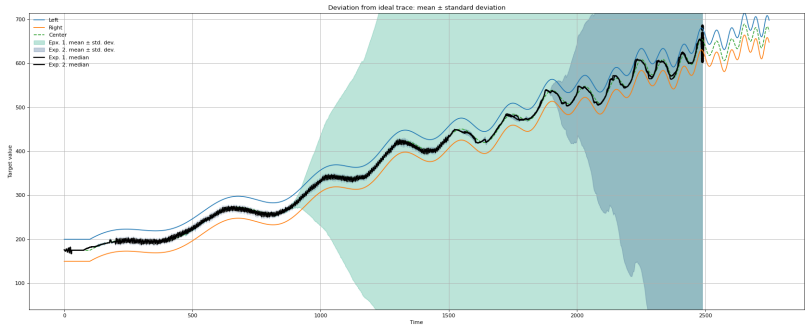


**Figure 1.** Error Deviation over Time for Configurations 3 and 4.

Figure 2 shows the cumulative tracking error over time on a log-scale for all four configurations. The curves are plotted on a log scale due to the large variance between successful and failed runs. In each subplot, individual runs are represented as light blue lines, with bold lines indicating the median and average trajectories. The left subplots show the cases where the agent learned and operated fully autonomously, always selecting the action deemed optimal by the model at that moment. The right subplots show the outcomes when the agent was artificially perturbed every 3 steps by forcing it into a less familiar region of the state space.

We can interpret our results as follows: Without perturbation in Configuration 1. (5,3 no perturbation) high error variance, frequent failure cases, unstable convergence occured. The network quickly overfits to a narrow input regime and fails to recover from errors. The same has happened in Configuration 3. when perturbation was not applied. Configuration 2. (5,3 + perturbation): Slight improvement in median performance, but lower failure rate even during the training. Configuration 4. (10,5 + perturbation): The larger network benefited the most from the perturbation showed better performance right after the training phase, but the performace graudally degrade again if perturbation is not applied furthermore as it can see on the bottom right plot after the 400th timestep. Cumulative

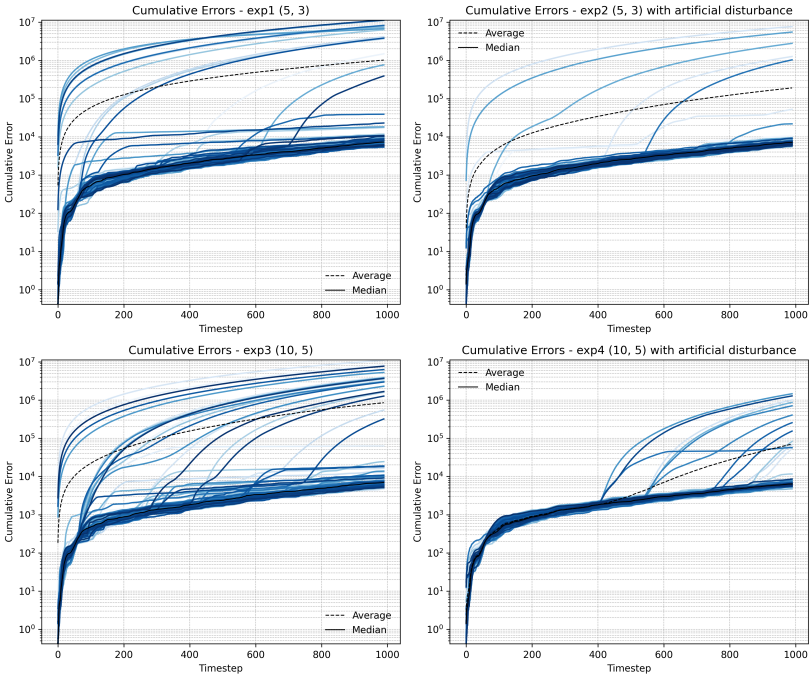


**Figure 2.** Cumulative error over test sequence under two type of training. Left: autonomous control with no intervention. Right: periodic perturbation to increase state-space diversity.

tracking error was recorded over time for all runs. Additionally, we defined a simple failure condition: a run was considered failed if, by the final timestep, the agent's deviation from the operational target exceeded a critical threshold. A run was classified as a failure if its final cumulative tracking error exceeded 20 000. This threshold was chosen empirically by observing that successful runs typically remained below 10 000 at the final timestep, while unstable runs accumulated errors an order of magnitude larger. Thus, the threshold reliably separates stable from divergent trajectories. This indicated that the agent entered an unstable resource state from which it could not recover. This indicated that the agent entered an unstable resource state from which it could not recover. The following failure rates were observed: Configuration 1 (5, 3, no perturbation): 21 / 100, Configuration 2 (5, 3, with perturbation): 8 / 100, Configuration 3 (10, 5, no perturbation): 26 / 100, Configuration 4 (10, 5, with perturbation): 13 / 100. The comparison shows that the perturbation strategy led to more consistent behavior and reduced the variance in deviations across different runs. These results indicate that the perturbed learning process achieved more robust generalization and improved stability over time.

# 8. Discussion and conclusion

This paper examines the effect of this self-induced sampling bias on learning robustness and looks into a simple but useful countermeasure, which is periodic artificial perturbation of the control decisions. By occasionally forcing the system into suboptimal or rarely visited states, we aim to reintroduce input diversity and stabilize the learning trajectory. Our hypothesis just as suggested by [7] is that such artificial exploration acts as an implicit regularizer, particularly beneficial in high-capacity models prone to overfitting and instability. Intuitively, the perturbations enlarge the effective sample size in the relevant regions of the state space, counteracting endogenous sampling bias and delaying premature consolidation to a narrow operating regime. Based on our experiments, models trained with this perturbation approach reach more stable convergence and show lower long-term variance in prediction and control error. This supports the idea that periodic and focused exploration, even in supervised control systems, can work as a regularizer that helps with generalization and stability.

In this study, we have identified a structural limitation of online supervised learning when embedded in closed-loop control systems. As the model gets better at keeping the system near optimal state, the diversity of its training data inherently shrinks. Over time, this feedback loop causes the input data to become more uniform, leading to self-induced overfitting to a narrow and stable operating regime. This distributional narrowing reduces the model's ability to generalize, and makes it more vulnerable to unexpected changes or unmodeled noise, a challenge also observed in semi-supervised one-pass learning under distribution shift [13].

We showed through extensive simulations that periodic artificial perturbation, without reward, curriculum design, or replay, improves the stability of the learning trajectory. This intervention can act as a form of implicit regularization, maintaining diversity in the input space and reducing run-to-run variance in control performance, including improved stability in VM allocation tasks. These findings suggest that in feedback-coupled supervised systems, where the learner influences its own data stream, exploration could be valuable and advantageous.

As a next step, we consider that such a control mechanism would be worth studying and applying not only in simulation but also in real-world cloud infrastructure, where incoming workload and task profiles change continuously, requiring adaptive strategies that preserve robustness and optimize energy efficiency under nonstationary conditions.

# References

[1] G. Alain, Y. Bengio: *Understanding intermediate layers using linear classifier probes*, arXiv preprint arXiv:1610.01644 (2016).

[2] G. Bouchard, T. Trouillon, J. Perez, A. Gaidon: *Online Learning to Sample*, in: Advances in Neural Information Processing Systems, 2015.

[3] R. Elwell, R. Polikar: *Incremental learning of concept drift in nonstationary environments*, in: IEEE Transactions on Neural Networks, vol. 22, 10, 2011, pp. 1517–1531.

[4] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia: *A survey on concept drift adaptation*, ACM Computing Surveys (CSUR) 46.4 (2014), pp. 1–37.

[5] A. Graves, M. G. Bellemare, J. Menick, R. Munos, K. Kavukcuoglu: *Automated Curriculum Learning for Neural Networks*, in: Proceedings of the 34th International Conference on Machine Learning (ICML), 2017.

[6] G. Hacohen, D. Weinshall: *On the Power of Curriculum Learning in Training Deep Networks*, arXiv preprint arXiv:1904.03626 (2019), doi: 10.48550/arXiv.1904.03626.

[7] *Handling Out-of-Distribution Data: A Survey*, IEEE Transactions on Knowledge and Data Engineering (2025), doi: 10.1109/tkde.2025.3592614.

[8] Y. Liu, C. Zhou, P. Zhang, Z. Li, S. Zhang, X. Lin, X. Wu: *CL4CO: A Curriculum Training Framework for Graph-Based Neural Combinatorial Optimization*, in: 2024, pp. 779–784, doi: 10.1109/ICDM59182.2024.00092.

[9] V. Losing, B. Hammer, H. Wersing: *Self-Adjusting Memory: How to Deal with Diverse Drift Types*, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), 2017, pp. 4899–4903, doi: 10.24963/ijcai.2017/690.

[10] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, P. Stone: *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey*, ArXiv abs/2003.04960 (2020).

[11] P. Soviany, R. T. Ionescu, P. Rota, N. Sebe: *Curriculum Learning: A Survey*, International Journal of Computer Vision 130 (2021), doi: 10.1007/s11263-022-01611-x.

[12] H. Wang, W. Fan, P. S. Yu, J. Han: *Mining concept-drifting data streams using ensemble classifiers*, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, 2003, pp. 226–235.

[13] H. Xie, X. Liu, L. Guo: *Semi-supervised One-pass Learning under Distribution Shift*, Proceedings of the 2023 6th International Conference on Big Data Technologies (2023), doi: 10.1145/3627377.3627446.

[14] B. Zhao, L. Wang, Z. Liu, Z. Zhang, J. Zhou, C. Chen, M. Kolar: *Adaptive Client Sampling in Federated Learning via Online Learning with Bandit Feedback*, Journal of Machine Learning Research 26.8 (2025), pp. 1–67.

[15] P. Zhao, T. Zhang: *Accelerating Minibatch Stochastic Gradient Descent using Stratified Sampling*, arXiv preprint arXiv:1405.3080 (2014), arXiv:1405.3080, doi: 10.48550/arXiv.1405.3080.