Dynamics Models for Offline Hyperparameter Selection in Water Treatment

Jordan Coblin, Han Wang, Martha White, Adam White

Keywords: Applied RL, Dynamics Models, Hyperparameter Selection, Water Treatment, Industrial Control

Summary

A key barrier to deploying reinforcement learning algorithms in real-world systems is the challenge of hyperparameter selection, particularly when simulators are unavailable or online interactions are costly. Recent work has proposed using calibration models trained on offline data to simulate environment interactions and support offline hyperparameter selection, but prior applications have been restricted to simple, simulated domains. In this paper, we present the first application of calibration models to a real-world industrial setting: the Drayton Valley water treatment plant in Alberta, Canada. We evaluate several calibration models, including a k-nearest neighbors model with a Laplacian distance metric, on high-dimensional, non-stationary sensor data for nexting prediction tasks. We demonstrate that these models can generate realistic long-horizon rollouts and recover meaningful hyperparameter sensitivity curves. We also explore how calibration models scale to year-long datasets, how they help select fine-tuning learning rates for pre-trained agents, and how they perform under distribution shifts. Our results provide a proof of concept for using offline dynamics models to guide RL deployment in real-world environments, highlighting both its promise and the challenges that remain.

Contribution(s)

1. We present the first application of calibration models for offline hyperparameter tuning in a real-world industrial context, focusing specifically on sensor prediction tasks in a water treatment plant.
   Context: The calibration model framework was previously proposed in Wang et al. (2022), but was only tested in simple, simulated environments. Prediction tasks for a water treatment plant were explored in Janjua et al. (2023), but the authors did not focus on hyperparameter tuning.
2. We extend empirical evaluation methods for calibration models by introducing analyses of rollout quality, hyperparameter sensitivity, and the use of dynamic time warping to assess alignment between model-generated and true trajectories. We use these methods to compare the performance of several calibration model architectures.
   Context: Previous work has focused primarily on comparing performance of the best hyperparameter configuration, but has not examined measures of model accuracy or hyperparameter sensitivity.
3. We bridge the gap toward real-world deployment by scaling calibration models to a year-long offline dataset and investigating their ability to simulate distribution shifts for the fine-tuning setting.
   Context: Prior research has focused on small-scale, simulated domains and has not addressed the critical challenges of scalability and adaptation under realistic distribution shifts, both of which are essential for deploying reinforcement learning systems in practice.

Dynamics Models for Offline Hyperparameter Selection in Water Treatment

Jordan Coblin[1,2], **Han Wang**[1], **Martha White**[1,2,3], **Adam White**[1,2,3]
{coblin,han8,whitem,amw8}@ualberta.ca

[1]**University of Alberta**
[2]**Alberta Machine Intelligence Institute (Amii)**
[3]**Canada CIFAR AI Chair at Amii**

Abstract

A key barrier to deploying reinforcement learning algorithms in real-world systems is the challenge of hyperparameter selection, particularly when simulators are unavailable or online interactions are costly. Recent work has proposed using calibration models trained on offline data to simulate environment interactions and support offline hyperparameter selection, but prior applications have been restricted to simple, simulated domains. In this paper, we present the first application of calibration models to a real-world industrial setting: the Drayton Valley water treatment plant in Alberta, Canada. We evaluate several calibration models, including a k-nearest neighbors model with a Laplacian distance metric, on high-dimensional, non-stationary sensor data for nexting prediction tasks. We demonstrate that these models can generate realistic long-horizon rollouts and recover meaningful hyperparameter sensitivity curves. We also explore how calibration models scale to year-long datasets, how they help select fine-tuning learning rates for pre-trained agents, and how they perform under distribution shifts. Our results provide a proof of concept for using offline dynamics models to guide RL deployment in real-world environments, highlighting both its promise and the challenges that remain.

1 Introduction

Reinforcement learning (RL) offers adaptive, data-driven control across a range of industrial applications, including assembly line automation (Tortorelli et al., 2022), thermal power generation (Zhan et al., 2022), and commercial cooling (Luo et al., 2022). However, deploying RL in real-world systems remains constrained by numerous practical challenges. Among these, the issue of hyperparameter selection has received comparatively little attention, despite RL performance being highly sensitive to hyperparameters such as learning rate, exploration schedule, and model architecture (Andrychowicz et al., 2020; Eimer et al., 2023; Henderson et al., 2018). Standard approaches either rely on default hyperparameters (Degrave et al., 2022), simulator-based tuning (Levine et al., 2016; OpenAI et al., 2019), or direct tuning in the real environment (Azuatalam et al., 2020; Luo et al., 2022), but the latter two are often infeasible when accurate simulators are unavailable or real-world interactions are costly, risky, or time-intensive.

A promising alternative is to leverage large offline datasets, which are commonly available in industrial systems, to support hyperparameter selection for RL agents deployed online — a setting referred to as *Data2Online*. Recent work suggests that calibration models trained on offline logs can approximate environment dynamics well enough for hyperparameter selection (Wang et al., 2022). However, this approach has so far been explored only in simple simulated domains, leaving open questions about its effectiveness in real-world systems.

In this work, we extend the calibration model framework to sensor prediction tasks in a water treatment plant (WTP) in Alberta, Canada. Our contributions are threefold: (1) we present the first application of calibration models for offline hyperparameter selection in a real industrial system; (2) we extend evaluation methods of calibration models, incorporating rollout quality analysis, hyperparameter sensitivity, and dynamic time warping to assess trajectory similarity; and (3) we investigate scalability to large offline datasets and robustness under distribution shifts, addressing challenges critical to real-world RL deployment.

2 Background

This section introduces the core framework and methods underlying our approach to offline hyperparameter selection with calibration models. We begin by formalizing the problem setting and calibration objective, then outline the k-nearest neighbors (KNN) calibration model used to approximate environment dynamics. We describe the nexting prediction framework as the primary learning task in our experiments and conclude with a brief overview of dynamic time warping as a method for evaluating calibration model rollouts.

2.1 Problem Formulation

Let $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{N}$ be a dataset of N transitions sampled from a Markov decision process (MDP) under a behavior policy π_β, where $s_i \in \mathcal{S}$ is a state, $a_i \in \mathcal{A}$ is an action, $r_i \in \mathbb{R}$ is a reward, and $s'_i \in \mathcal{S}$ is a next state. A dynamics model $\hat{p}(s', r|s, a)$ is trained on $\mathcal{D}$ to approximate the environment's transition function $p(s', r|s, a)$. The purpose of the model is not to plan or learn a policy, but to evaluate hyperparameter configurations by simulating environment interactions. Accordingly, we refer to the model as a *calibration model*.

Let $\mathcal{A}$ be a learning algorithm, Λ the hyperparameter space, and $\lambda \in \Lambda$ a hyperparameter configuration. For each λ, we define a sequence of policies $\{\pi_t^\lambda\}_{t=0}^{\infty}$ as the result of running $\mathcal{A}$ interactively in the environment p, i.e. $\{\pi_t^\lambda\} = \mathcal{A}(\lambda; p)$. The expected return in the true environment is defined as:

$$J_{\text{env}}(\lambda) = \mathbb{E}_p\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, a_t \sim \pi_t^\lambda(\cdot|s_t),\, (s_{t+1}, r_t) \sim p(\cdot|s_t, a_t)\right],$$

where $\gamma \in [0, 1]$ is the discount factor. Similarly, the expected return using a calibration model is $J_{\text{model}}(\lambda) = \mathbb{E}_{\hat{p}}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, a_t \sim \pi_t^\lambda(\cdot|s_t),\, (s_{t+1}, r_t) \sim \hat{p}(\cdot|s_t, a_t)\right]$, where the policy sequence $\{\pi_t^\lambda\}$ is now learned via interaction with $\hat{p}$, $\{\pi_t^\lambda\} = \mathcal{A}(\lambda; \hat{p})$.

We define the optimal hyperparameters in each case as $\lambda^\star_{\text{env}} = \arg\max_{\lambda \in \Lambda} J_{\text{env}}(\lambda)$ and $\lambda^\star_{\text{model}} = \arg\max_{\lambda \in \Lambda} J_{\text{model}}(\lambda)$, and aim to learn a calibration model $\hat{p}$ such that $\lambda^\star_{\text{model}} = \lambda^\star_{\text{env}}$.

2.2 KNN Calibration Model

A good calibration model must be stable under long horizon rollouts, since hundreds or thousands of steps are typically necessary to evaluate a hyperparameter configuration. However, typical dynamics models are known to suffer from compounding errors, which can lead to significant divergence from the true environment over long horizons (Talvitie, 2017; Lambert et al., 2022). In order to mitigate this, Wang et al. (2022) propose using a non-parametric KNN model to estimate p by predicting next states and rewards using only transitions within the dataset, avoiding extrapolation into unobserved regions.

The KNN model samples transitions from $\mathcal{D}$ to find the k nearest neighbors of a given state-action pair according to some distance metric. Instead of using distance in the raw state-action space to find neighbors, an approximate Laplacian representation (Wu et al., 2018) can be used to construct a distance metric that is sensitive to the underlying structure of the MDP. We refer to a model that uses this distance metric as a *Laplacian KNN* model, and a model that uses distance in the raw state-action space as a *Euclidean KNN* model.

To address epistemic uncertainty, we use a bootstrapped ensemble of KNN models, where each model is trained on a different subset of $\mathcal{D}$; we use five bootstraps in our experiments. Results across models are aggregated by taking the *worst rank* of a given hyperparameter configuration across all models, providing a conservative estimate of hyperparameter performance.

2.3 Nexting Prediction

The *nexting prediction* problem involves an agent predicting the sum of a future observation signal framed as a general value function (GVF) (Modayil et al., 2012). This formulation enables temporally extended predictions about scalar signals (often called *cumulants*) in an environment, providing a foundation for learning and adaptation. Following Janjua et al. (2023), in this work we focus on the nexting prediction problem for sensors in a water treatment plant. For a signal o_t^i at time t, associated with channel or sensor i, the nexting value function is defined as:

$$v_t^i(s) = \mathbb{E}\left[G_t^i \mid s_t = s\right], \quad G_t^i = \sum_{k=0}^{\infty} \gamma^k o_{t+k+1}^i, \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor controlling the timescale of the prediction. Predictive accuracy is often assessed using the root mean squared error (RMSE) between the predicted value $\hat{v}_t^i$ and the empirical Monte Carlo return G_t^i from data trajectories across all time steps $t \in \{0, 1, \ldots, T\}$.

2.4 Dynamic Time Warping

Dynamic time warping (DTW) is a technique for measuring similarity between time series that may differ in speed, timing, or phase, making it widely useful in tasks like speech recognition and time series analysis (Kruskal & Liberman, 1983). Given two sequences, DTW uses warping functions ϕ_x and ϕ_y to flexibly align elements along a common time axis, allowing sections to "stretch" or "compress" for better matching, even when sequences are out of sync or unevenly sampled. Alignment is guided by constraints collectively defined by a *step pattern*, which shapes allowable warping paths. We apply DTW to compare calibration model rollouts against dataset rollouts, since long-horizon rollouts are rarely perfectly aligned. We also use four common DTW step patterns (Giorgino, 2009) in our analysis, to mitigate sensitivity to a single step pattern choice.

3 Real-World Application: Water Treatment Plant

To move beyond simulation environments, we test our calibration models on a working membrane-filtration pilot at the Drayton Valley water treatment plant (WTP) in Alberta, Canada. Here, we face high-dimensional, non-stationary, and noisy sensor data that are typical of real-world systems. Moreover, the absence of a simulator — combined with safety and sampling constraints — makes calibration models a practical and necessary tool for offline evaluation in this setting.

3.1 Dataset

The offline dataset consists of over two years of sensor logs (collected between 2022 and 2024) from the WTP, with 480 sensor channels sampled at 1 Hz. For our initial experiments, we use a one-week slice of the dataset, which contains $\sim 3.5 \times 10^5$ transitions. We discard sensors with consistent missing data and constant values, leaving us with a 142-dimensional feature vector for our prediction agents and calibration models. In general, we leverage the data processing pipeline outlined in Janjua et al. (2023).

For our experiments, we consider three sensors that are both critical for control of the plant and provide variety in dynamics to predict. These are the membrane pressure (PIT300), influent temperature (TIT101), and influent turbidity (TUIT101) sensors.

3.2 Experimental Setup

We train four calibration models: a bootstrapped Euclidean KNN, a bootstrapped Laplacian KNN, a feedforward neural network (NN), and a gated recurrent unit neural network (GRU). The NN and GRU architectures are trained to minimize the one-step next-state prediction loss $\mathcal{L} = \mathbb{E}_{(s_t,a_t,s_{t+1})\sim D} \left\|\hat{s}_{t+1} - s_{t+1}\right\|^2$, and serve as baselines, illustrating the limitations of models prone to compounding errors when rolled out over many steps. We evaluate each model based on its ability to (1) generate realistic long-horizon rollouts and (2) support hyperparameter selection for a TD(0) (Sutton & Barto, 2018) prediction agent. The evaluation strategies can be summarized as follows:

- **Rollout Quality:** We visualize 30k-step trajectories from the dataset and compare them to model rollouts starting from the same initial state. We opt for a qualitative approach to evaluating rollout quality here, as time series comparisons are made tricky by the fact that there may exist temporal offsets or changes in frequency between two otherwise similar time series. While we explore using DTW in Section 4 to tackle these issues, we find that a single scalar metric is limited in its ability to convey information.
- **Hyperparameter Selection:** We evaluate how well each calibration model supports offline hyperparameter tuning by sweeping Adam learning rates for a TD(0) prediction agent. The agent learns from scratch using either a calibration model or the true environment, which is used as a ground-truth for comparison. We measure performance using RMSE normalized by average return (NRMSE) or, for bootstrapped KNNs, hyperparameter rank. We then compare the resulting hyperparameter curves to those from the true environment to assess how well each model preserves hyperparameter sensitivity.

3.3 Results

In Figure 1, we compare the rollouts of the four calibration models to the true environment (i.e. offline test set) for PIT300. We find that the KNN models produce sensor trajectories that resemble the true environment for PIT300, with the Laplacian KNN showing closer alignment overall. For other sensors, rollouts are less accurate across all models, but the KNN models still outperform NN and GRU baselines, which tend to collapse after $\sim$50–100 timesteps.

Figure 2 shows learning rate sensitivity curves for each model. KNN results are plotted on a separate rank axis in accordance with the bootstrapping strategy described in Section 2 — similarity in *shape* of the sensitive curves is what we use to determine agreement. Both KNN models generally recover the correct hyperparameter rankings, with the Euclidean KNN aligning most closely with the ground truth for TIT101 and TUIT101. The weaker performance of the Laplacian KNN may stem from using a representation tuned for PIT300, highlighting the need for more robust Laplacian selection. As expected, the NN and GRU models show poor hyperparameter sensitivity, consistent with their low-quality rollouts.

4 Towards Real-World Deployment

There is often a significant gap between an algorithm working within the context of a controlled research experiment and in the increased complexity of real-world application. In this section, we hope to bridge the gap towards application by exploring several modifications to the setting that was introduced in Section 3. These modifications are broken down into three categories:

Scaling Up: In industrial settings like water treatment, years of offline sensor logs are available, presenting an opportunity to train models that capture non-stationarity, seasonality, and rare events. We extend the KNN calibration model to a full year of water treatment data, amounting to $\sim$32M samples at 1 Hz, which we sub-sample by a factor of 10 to yield a more manageable $\sim$3.2M samples. The model is built in two phases—KD-tree construction and neighbor table generation—resulting in

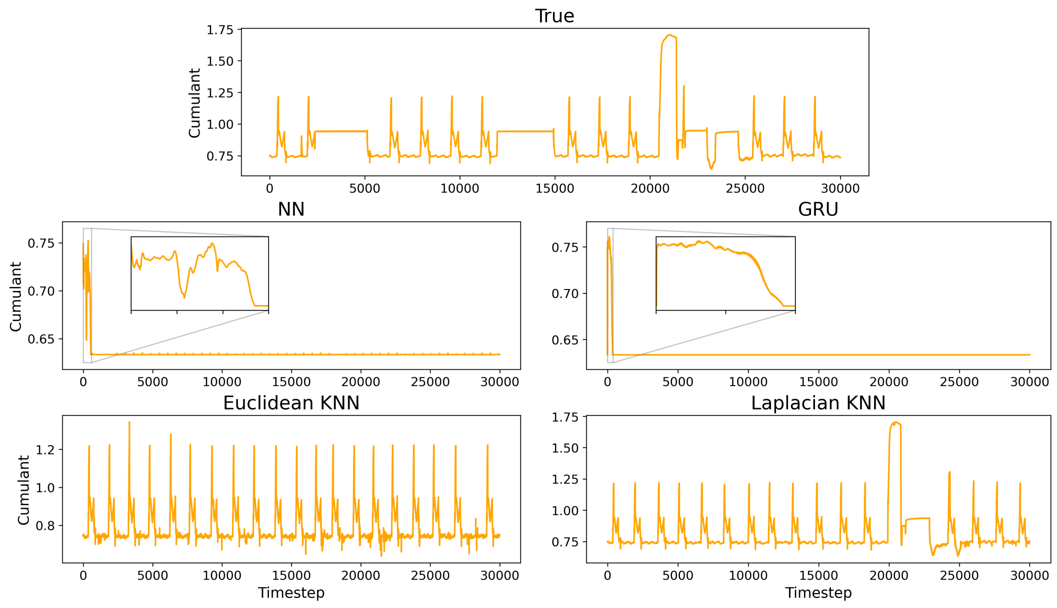

Figure 1: PIT300 sensor (membrane pressure) rollouts in the true environment and calibration models. Each model is rolled out for 30k steps, beginning from the same start state.

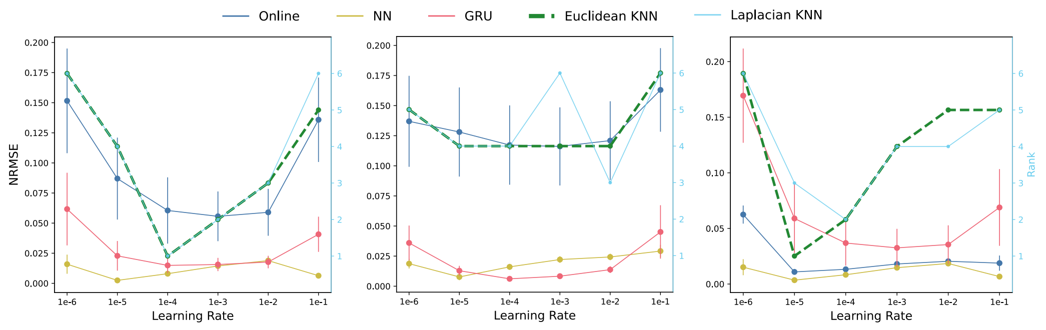

Figure 2: Learning rate sensitivity curves for a TD(0) prediction agent in the true (Online) environment and calibration models using the 1-week WTP dataset. From left to right, prediction targets are the PIT300, TIT101 (influent temperature), and TUIT101 (influent turbidity) sensors. Normalized root mean squared error (NRMSE) with 95% confidence intervals is shown for non-bootstrapped models, while worst rank is used for bootstrapped models (i.e. KNN models), which is plotted against the right rank y-axis. Wider, dotted lines are used for the Euclidean KNN to distinguish it from the Laplacian KNN line, which often overlap.

a total complexity of $\mathcal{O}((d+k)n\log n)$ (Brown, 2015), which takes approximately 10 hours on a 2 GHz Quad-Core Intel Core i5 processor for $d = 142$, $k = 3$, and $n = 3.2$M. While computationally intensive, this process is a one-time cost, and can be further accelerated via dimensionality reduction (e.g., PCA or autoencoders), prototype selection (Wilson & Martinez, 2000), or approximate nearest-neighbor methods (Indyk & Motwani, 1998; Arya et al., 1998).

Agent Pre-training: In real-world settings where no simulator exists, we aim to maximize the utility of offline data. This typically involves pre-training an agent to avoid learning from scratch at deployment, after which the agent can continue to adapt online. We refer to this as the *fine-tuning* setting, and shift our focus to selecting the fine-tuning learning rate. Since the same offline data is also used to construct a calibration model, we adopt a simple partitioning strategy to separate data for pre-training and calibration, reducing overlap and better simulating a realistic transfer scenario.

Distribution Shift: System dynamics in real-world settings like the water treatment plant can vary significantly over time due to factors such as rainfall, temperature, filter condition, and sensor drift. To evaluate how well calibration models handle such changes, we construct test sets that begin one week, one month, and three months after the training period, which correspond to April 2023, May 2023, and July 2023 respectively. These test sets are meant to capture different distribution shifts from the training data, and are used to evaluate the generalization capabilities of the calibration model. As illustrated in Figure 3, sensor patterns such as those from TIT101 change meaningfully across time, posing a challenge for generalization.

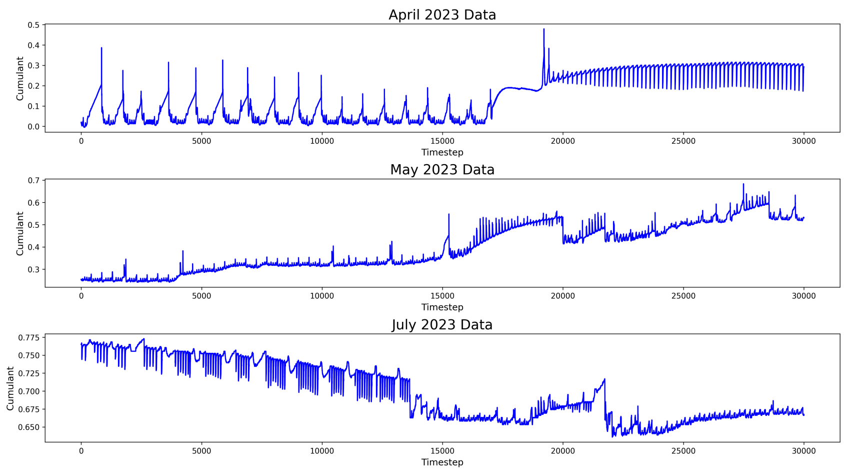

Figure 3: TIT101 sensor values across several time periods in the water treatment plant (WTP) dataset. Each plot shows roughly 3.5 days of data.

In general, it is not guaranteed that the calibration model will be able to simulate an arbitrary deployment period, since the distribution shift may be too large. However, we hypothesize that selecting a rollout start state that is representative of a deployment period can encourage the calibration model to simulate the dynamics of that period. There are a variety of possible strategies for selecting rollout start states $\mathcal{S}_0 = \{s_0^{[0]}, \ldots, s_0^{[r]}\}$, where r is the number of rollouts we perform:

(i) Randomly selecting $\mathcal{S}_0$ from the entire dataset (baseline method).

(ii) Selecting $\mathcal{S}_0$ from the same calendar month in a previous year (e.g., July 2022 for a July 2023 deployment), if available.

(iii) Using samples from the online deployment period and find their most similar neighbors in the offline dataset.

In our experiments, we focus on (iii) with (i) as a baseline, as (ii) assumes seasonal consistency that may not hold due to sensor drift or evolving plant conditions. Our use of bootstrapping also means that each individual start state will not exist in all bootstrap models, further complicating (ii).

4.1 Experiments

To put the preceding ideas into practice, we evaluate our Laplacian KNN calibration model through two key questions: (1) does scaling to a year-long dataset improve generalization, and (2) can the model simulate distribution shifts to support fine-tuning learning rates? We combine qualitative rollout analysis and quantitative metrics to assess the model's utility for offline hyperparameter selection.

Does scaling up the KNN calibration model improve its generalization capabilities? We compare a KNN model trained on one year of data (*12-month KNN*), with one trained on the one-week dataset from Section 3 (*1-week KNN*). For each test period (April, May, and July 2023), we sample 30 random states and use their nearest neighbors under the Laplacian distance metric as rollout start states, following technique (iii) from Paragraph 4. We evaluate rollouts both qualitatively (visual inspection) and quantitatively using DTW, where a lower DTW score indicates closer resemblance to the ground-truth sequence.

Figure 4 shows a subset of PIT300 rollouts using start states from the April 2023 test set. While it is challenging to qualitatively compare rollouts, we note that the 12-month KNN is able to capture a broader range of dynamics than the 1-week KNN, with similar results for TIT101 and TUIT101. Quantitative results using DTW (Table 1) show mostly consistent rankings across step patterns, but mixed results across models: the 12-month KNN performs best on TIT101, the 1-week KNN on TUIT101, and mixed results on PIT300. Overall, these findings suggest that while the 12-month model may offer broader behavioral coverage, its generalization advantage is not conclusive. While DTW was preferable over mean squared or absolute error due to alignment issues, alternate metrics may be needed for clearer assessment (Coblin, 2024).

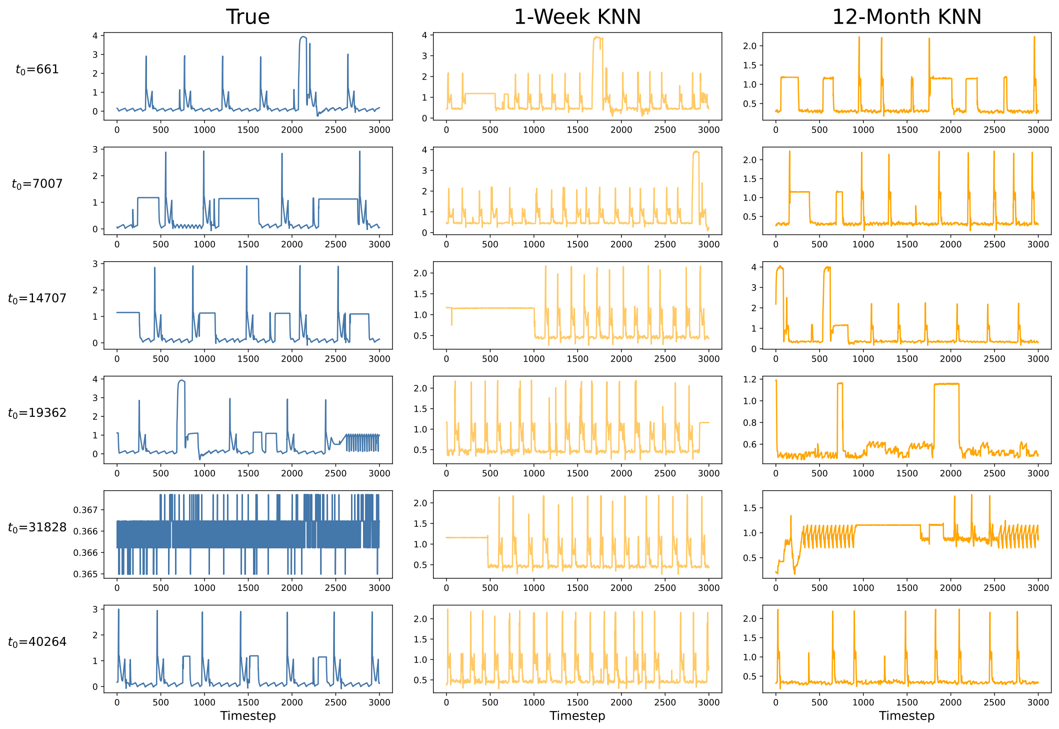

Figure 4: PIT300 rollouts for the 12-month and 1-week WTP KNN calibration models using start states t_0 from the April 2023 dataset. True rollouts from those start states are shown in the leftmost plots.

Can the KNN calibration model simulate distribution shifts to support fine-tuning learning rate selection? To simulate a fine-tuning scenario, we pre-train a TD(0) prediction agent on the first six months of data and use the remaining six months to construct a KNN calibration model. We then use this model to guide selection of the fine-tuning learning rate for the pre-trained agent, with start states selected using technique (iii) from Paragraph 4 (*Month Start States*), and using random start states as described in technique (i) (*Year Start States*) for comparison.

The ground truth (Online) curves in Figure 5 show that small learning rates yield good performance given a small distribution shift in the April 2023 test set. However, as the deployment period gets

	PIT300		TIT101		TUIT101	
	1-Week	12-Month	1-Week	12-Month	1-Week	12-Month
Symmetric2	$\mathbf{414.02 \pm 0.02}$	483.24 ± 0.02	910.20 ± 0.05	$\mathbf{782.05 \pm 0.04}$	$\mathbf{154.78 \pm 0.01}$	194.86 ± 0.01
Asymmetric	$\mathbf{224.0 \pm 0.02}$	283.38 ± 0.03	523.05 ± 0.05	$\mathbf{431.87 \pm 0.04}$	$\mathbf{100.92 \pm 0.01}$	153.81 ± 0.02
SymmetricP1	805.51 ± 0.04	$\mathbf{782.79 \pm 0.04}$	1031.66 ± 0.05	$\mathbf{898.61 \pm 0.04}$	$\mathbf{199.55 \pm 0.01}$	268.24 ± 0.01
RabinerJuang	$\mathbf{339.87 \pm 0.03}$	367.87 ± 0.04	521.66 ± 0.05	$\mathbf{447.82 \pm 0.04}$	$\mathbf{100.43 \pm 0.01}$	154.09 ± 0.02

Table 1: Dynamic time warping distances computed between rollouts from KNN models (1-Week and 12-Month) and true rollouts, averaged over three test sets with 30 rollouts each for each model and sensor combination. Smaller distance is better, and the best model for a specific sensor is presented in bold font. The leftmost column shows the step pattern used for the DTW algorithm.

further away, the performance of the smallest learning rates deteriorates, indicating that the agent requires more adaptation. We find that the 12-month KNN model is able to simulate some kind of distribution shift, as shown by the learning rate curves giving best performance around 1×10^{-4} to 1×10^{-5}. However, it struggles to capture *specific* shifts, as seen by its mismatch with the ground truth sensitivity curves. Additionally, performance between the month start states and full year start states is similar, suggesting limited benefit from our targeted start state rollout strategy. While these experiments offer a first step toward using calibration models under distribution shift, further work is needed to understand how to prompt the model to simulate dynamics from a specific deployment period, or even a period within its training data.

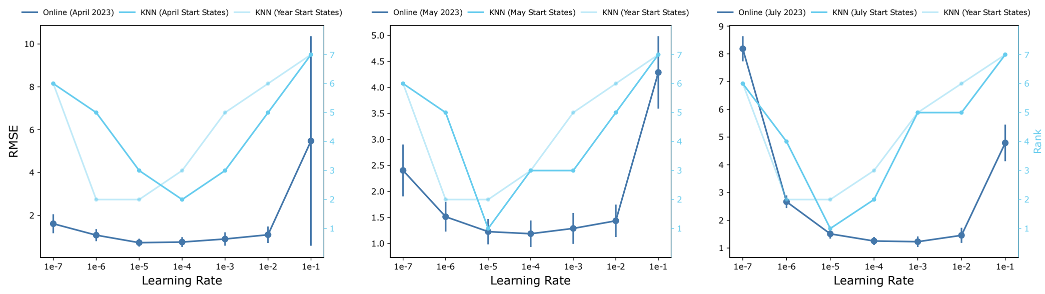

Figure 5: Fine-tuning learning rate sensitivity curves for the true (Online) environment and calibration models for PIT300 using a full year of data. Left to right: April 2023, May 2023, and July 2023 test sets. Root mean squared error (RMSE) with 95% confidence intervals is shown for the Online environment, while worst rank is used for bootstrapped KNN models, which is plotted against the rank axis.

5 Conclusion

This work extends the calibration model framework for offline hyperparameter selection to a real-world industrial setting, focusing on sensor prediction tasks in a water treatment plant. We show that KNN-based calibration models, particularly those using Laplacian distance metrics, can support realistic rollouts and accurate hyperparameter selection. By investigating key extensions such as scaling to large datasets, fine-tuning, and non-stationarity, we provide a proof of concept for leveraging offline data to guide online RL deployment. However, important open questions remain: for what kinds of environments do extrapolating models perform poorly? How should we approach simulating distribution shifts? To what extent is model accuracy necessary for hyperparameter selection? And are there better metrics than DTW for assessing rollout quality? Tackling these questions will help advance our understanding of calibration models and learned dynamics models more generally, particularly in their application to real-world systems.

References

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020.

Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45 (6):891–923, 1998.

Donald Azuatalam, Wee-Lih Lee, Frits de Nijs, and Ariel Liebman. Reinforcement learning for whole-building hvac control and demand response. *Energy and AI*, 2:100020, 2020. ISSN 2666-5468.

Russell A. Brown. Building a balanced k-d tree in o(kn log n) time. *Journal of Computer Graphics Techniques*, 4(1):50–68, 2015.

Jordan Coblin. Calibration models for real-world deployment of reinforcement learning agents. Master's thesis, University of Alberta, 2024.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

Muhammad Janjua, Haseeb Shah, Martha White, Erfan Miahi, Marlos Machado, and Adam White. Gvfs in the real world: making predictions online for water treatment. *Machine Learning*, pp. 1–31, 11 2023.

JB Kruskal and Mark Liberman. The symmetric time-warping problem: From continuous to discrete. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 01 1983.

Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating compounding prediction errors in learned dynamics models. *arXiv preprint arXiv:2203.09637*, 2022.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Jerry Luo, Cosmin Paduraru, Octavian Voicu, Yuri Chervonyi, Scott Munns, Jerry Li, Crystal Qian, Praneet Dutta, Jared Quincy Davis, Ningjia Wu, Xingwei Yang, Chu-Ming Chang, Ted Li, Rob Rose, Mingyan Fan, Hootan Nakhost, Tinglin Liu, Brian Kirkman, Frank Altamura, Lee Cline, Patrick Tonker, Joel Gouker, Dave Uden, Warren Buddy Bryan, Jason Law, Deeni Fatiha, Neil Satra, Juliet Rothenberg, Mandeep Waraich, Molly Carlin, Satish Tallapaka, Sims Witherspoon, David Parish, Peter Dolan, Chenyu Zhao, and Daniel J. Mankowitz. Controlling commercial cooling systems using reinforcement learning, 2022.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pp. 2295–2304. PMLR, 2017.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

Joseph Modayil, Adam White, and Richard S. Sutton. Multi-timescale nexting in a reinforcement learning robot. In Tom Ziemke, Christian Balkenius, and John Hallam (eds.), *From Animals to Animats 12*, pp. 299–309. Springer Berlin Heidelberg, 2012.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 2595–2601. AAAI Press, 2017.

Andrea Tortorelli, Imran Muhammad, Francesco Priscoli, and Francesco Liberati. A parallel deep reinforcement learning framework for controlling industrial assembly lines. *Electronics*, 11:539, 02 2022.

Han Wang, Erfan Miahi, Martha White, Marlos C. Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Investigating the properties of neural network representations in reinforcement learning. 330:104100. ISSN 0004-3702.

Han Wang, Archit Sakhadeo, Adam White, James Bell, Vincent Liu, Xutong Zhao, Puer Liu, Tadashi Kozuno, Alona Fyshe, and Martha White. No more pesky hyperparameters: Offline hyperparameter tuning for rl, 2022.

D Randall Wilson and Tony R Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38:257–286, 2000.

Yifan Wu, George Tucker, and Ofir Nachum. The laplacian in rl: Learning representations with efficient approximations. *arXiv preprint arXiv:1810.04586*, 2018.

Xianyuan Zhan, Haoran Xu, Yue Zhang, Xiangyu Zhu, Honglei Yin, and Yu Zheng. Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning, 2022.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Laplacian Distance Metric

Laplacian representations involve the graph Laplacian matrix L, which is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix of the MDP. Laplacian representations have proved useful in RL for value-function approximation (Mahadevan & Maggioni, 2007), option discovery (Machado et al., 2017), and reward shaping (Wu et al., 2018), among other applications.

Let $\mathbf{u}_1, \ldots, \mathbf{u}_d$ be the first d eigenvectors of the graph Laplacian L. These define the representation

$$\psi : \mathcal{S} \longrightarrow \mathbb{R}^d, \qquad \psi(s) = \left[\mathbf{u}_1(s), \ldots, \mathbf{u}_d(s)\right]^\top,$$

where $\mathbf{u}_i(s)$ is the value of the i-th eigenvector evaluated at state s. The distance metric is then defined as

$$d(s_i, a_i, s_j, a_j) = \|\psi(s_i, a_i) - \psi(s_j, a_j)\|_2^2$$

Analytically computing the eigenvectors of L is typically not feasible for large graphs with an unknown transition function. Hence, we use an approximate method following the work in Wu et al. (2018) and Wang et al. (2022), which leverages spectral graph drawing to stochastically approximate the eigenfunctions of the Laplacian. Given a dataset $\mathcal{D}$, the graph drawing objective can be expressed as

$$\sum_{s_t \sim \mathcal{D}} \left\|\psi_\theta(s_t) - \psi_\theta(s_{t+1})\right\|_2^2 + \sum_{s_i, s_j \sim \mathcal{D}} \Big((\psi_\theta(s_i)^T \psi_\theta(s_j))^2 - \left\|\psi_\theta(s_i)\right\|_2^2 - \left\|\psi_\theta(s_j)\right\|_2^2\Big),$$

where $\psi_\theta : \mathcal{S} \to \mathbb{R}^d$ is the representation learned via a neural network with parameters θ. Intuitively, we can view this objective as being comprised of an *attractive term* and a *repulsive term*. The first term is attractive insofar as it encourages ψ_θ to map states s_t and their successors s_{t+1} closely in the representation space — this roughly captures temporal distance within an MDP. Conversely, the second term encourages independently sampled state pairs from the dataset to have orthogonal representations.

B Calibration Model Training Details

KNN and Laplacian Models The KNN calibration models were trained using a two-step procedure. First, a KD-tree was constructed over all transitions in the dataset, enabling efficient nearest-neighbor searches. For the Laplacian KNN, an additional learning step was applied to produce a learned representation that incorporates graph-structured distances between states, guided by a dynamics awareness score. Candidate Laplacian representations were validated using dynamics awareness (Wang et al.) to match rollout similarity to the true environment. Once the distance metric was selected, a fixed neighbor table was built to allow constant-time next-state predictions during rollouts.

NN Calibration Models The NN calibration models used a two-layer feedforward neural network architecture with a shared set of hyperparameters across sensors. Models were trained on a one-step-ahead prediction objective, mapping the current state (and optionally action) to the next state. A grid search over learning rates, hidden sizes, and batch sizes was conducted, and the best hyperparameters were selected based on performance on a validation set — see Table 3. Although multi-step prediction targets were explored, one-step predictions yielded better rollout performance in practice.

GRU Calibration Models The GRU calibration models employed a two-layer recurrent architecture designed to capture temporal dependencies in the sensor data. Models were trained on one-step-ahead prediction using sequences of historical states, with a burn-in period and sequence length tuned for each dataset — see Table 4. Dropout and other regularization techniques were tested to improve long-horizon stability, but simple GRU models with tuned hidden sizes and sequence lengths produced the best results. As with the NN models, hyperparameters were selected via grid search, focusing on validation performance — see Table 4.

Hyperparameter	**Symbol**	**Water 1-Week (Online)**	**Water 12-Month (Pre-training)**
Optimizer	-	Adam	Adam
Learning Rate	α	-	1×10^{-5}
Discount Factor	γ	0.99	0.9
Batch Size	B	256	256
Hidden Layers	-	2	2
Hidden Units	-	256	512
Replay Buffer Size	-	1M	3M
Train/Validation Split	-	-	0.9/0.1
Epochs	-	-	1000

Table 2: Hyperparameters used for the TD(0) prediction agent in water treatment experiments.

Hyperparameter	**Symbol**	**Water 1-Week**
Optimizer	-	Adam
Learning Rate	α	1×10^{-3}
Batch Size	B	256
Hidden Layers	-	2
State Model Hidden Size	-	512
Epochs	-	100

Table 3: NN calibration model training hyperparameters for water treatment.

Hyperparameter	**Symbol**	**Water 1-Week**
Optimizer	-	Adam
Learning Rate	α	1×10^{-3}
Batch Size	B	256
Hidden Layers	-	2
State Model Hidden Size	-	512
Burn-in Length	-	0
Sequence Length	-	20
State Model Epochs	-	100

Table 4: GRU calibration model training hyperparameters for water treatment.

Hyperparameter	**Symbol**	**Water 1-Week**	**Water 12-Month**
Optimizer	-	Adam	Adam
Learning Rate	α	3×10^{-4}	1×10^{-5}
Batch Size	B	256	256
Hidden Layers	-	2	2
Hidden Units	-	256	256
Output Dimension	-	64	64
Training Steps	-	100,000	200,000
Train/Validation Split	-	0.8/0.2	0.8/0.2
Sequence Length	-	20	20
Kappa	κ	0.95	0.95
Beta	β	5	5
Zeta	ζ	0.05	0.05

Table 5: Laplacian representation training hyperparameters for water treatment.

C Experiment Details for Scaled Up Generalization Capabilities

We compare a KNN model trained on one year of data between March 31, 2022 and March 31, 2023 (*12-month KNN*), with one trained on the one-week dataset from Section 3 (*1-week KNN*).

For each test period (April, May, and July 2023), we sample 30 random states and use their nearest neighbors under the Laplacian distance metric as rollout start states, following technique (iii) from Paragraph 4. Each rollout spans 10k steps. Hyperparameters are provided in Table **??**.

D Experiment Details for Fine-Tuning Learning Rate Selection

To simulate a fine-tuning scenario, we pre-train a TD(0) prediction agent on the first six months of data and use the remaining six months to construct a KNN calibration model (i.e. an equal partitioning strategy). We then use this model to guide selection of the fine-tuning learning rate. Hyperparameters for the TD(0) prediction agent are provided in Table 2.

For each learning rate, we perform 30 runs in the online environment and 10 per bootstrap for the calibration model, resulting in 50 total runs per α in the bootstrapped setting. We report RMSE averaged over the final 25% of each run, so that early training error does not dominate the metric.

E Additional Results

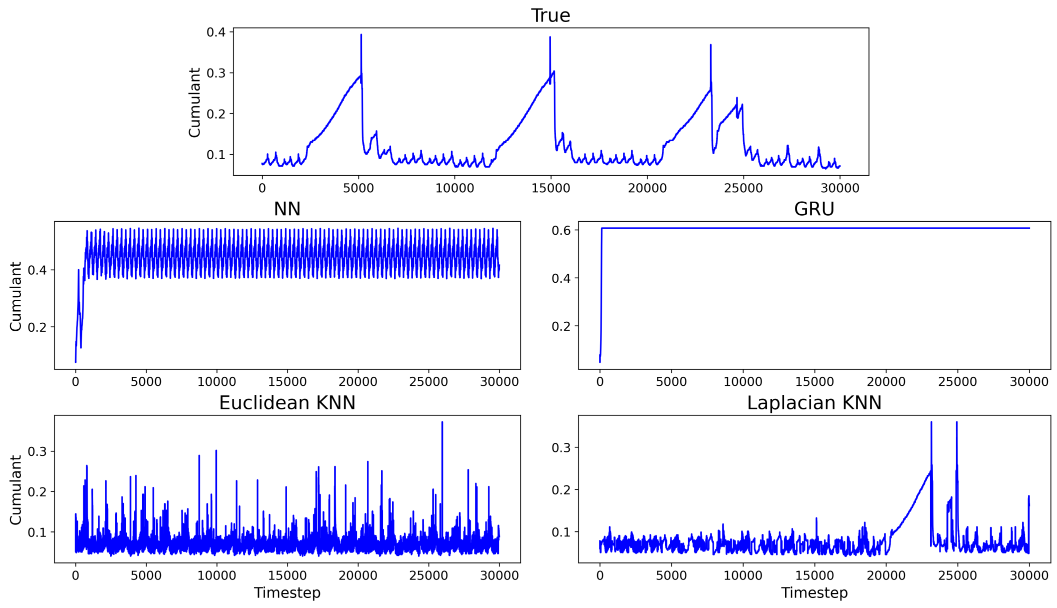

Figure 6: TIT101 sensor (influent temperature) rollouts in the true environment and calibration models. Each model is rolled out for 30k steps, beginning from the same start state.

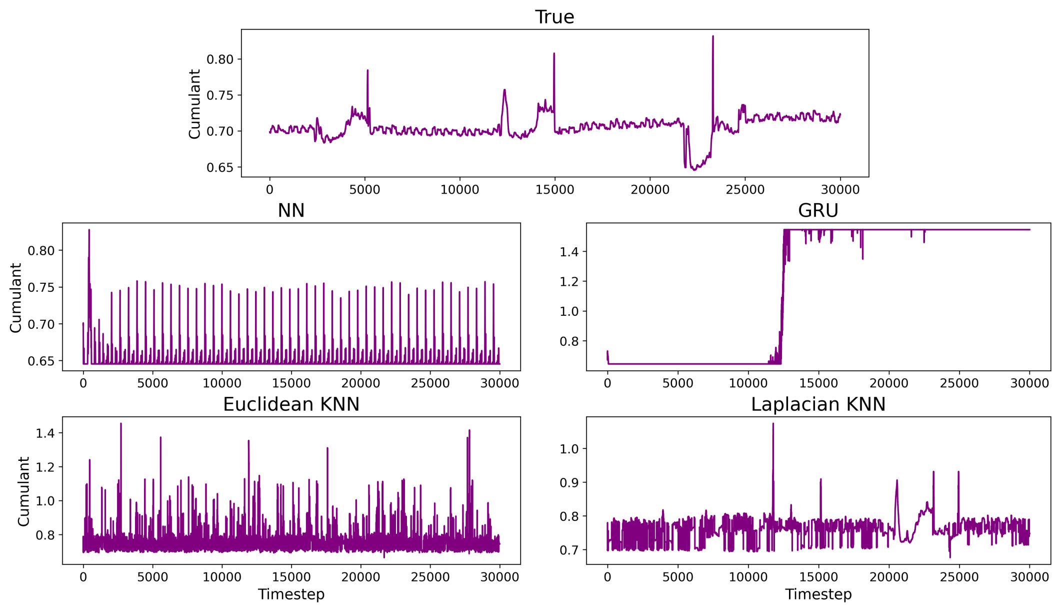

Figure 7: TUIT101 sensor (influent turbidity) rollouts in the true environment and calibration models. Each model is rolled out for 30k steps, beginning from the same start state.

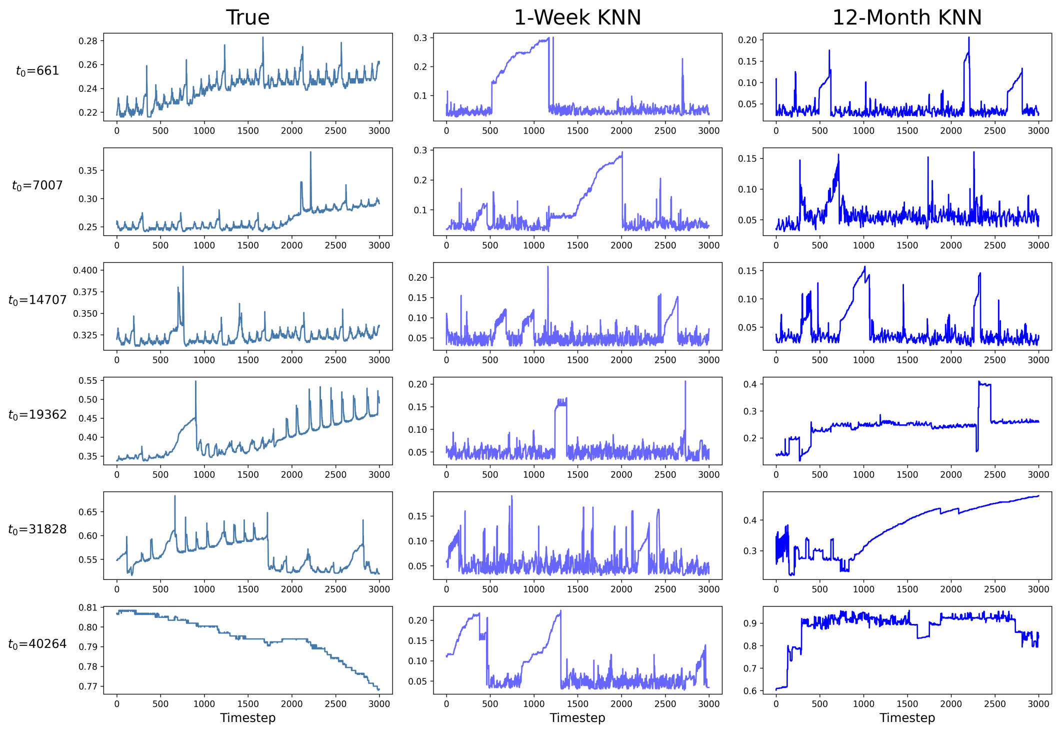

Figure 8: TIT101 rollouts for the 12-month and 1-week WTP KNN calibration models using start states t_0 from the May 2023 dataset. True rollouts from those start states are shown in the leftmost plots.