ONLINE FINE-TUNING OF PRETRAINED CONTROLLERS FOR AUTONOMOUS DRIVING VIA REAL-TIME RECURRENT RL

Julian Lemmel,* Felix Resch,* Mónika Farsang
TU Wien

Ramin Hasani, Daniela Rus
MIT CSAIL, Liquid AI

Radu Grosu
TU Wien

ABSTRACT

Deploying pretrained policies in real-world applications presents substantial challenges that fundamentally limit the practical applicability of learning-based control systems. When autonomous systems encounter environmental changes in system dynamics, sensor drift, or task objectives, fixed policies rapidly degrade in performance. We show that employing Real-Time Recurrent Reinforcement Learning (RTRRL), a biologically plausible algorithm for online adaptation, can effectively fine-tune a pretrained policy to improve autonomous agents' performance on driving tasks. We further show that RTRRL synergizes with a recent biologically inspired recurrent network model, the Liquid-Resistance Liquid-Capacitance RNN. We demonstrate the effectiveness of this closed-loop approach in a simulated `CarRacing` environment and in a real-world line-following task with a RoboRacer car equipped with an event camera.

1 INTRODUCTION

Learning-based control policies often struggle when deployed outside their training environments. Subtle differences in system dynamics, or sensor characteristics – referred to as distribution shifts – can cause pretrained policies to perform poorly or even fail entirely Cobbe et al. (2018); Witty et al. (2021); Korkmaz (2024). This vulnerability limits the practical use of autonomous agents in real-world settings – particularly in robotics and autonomous driving – where environmental conditions are constantly evolving, and mission objectives may change over time Tobin et al. (2017); Voogd et al. (2023); Li & Okhrin (2024).

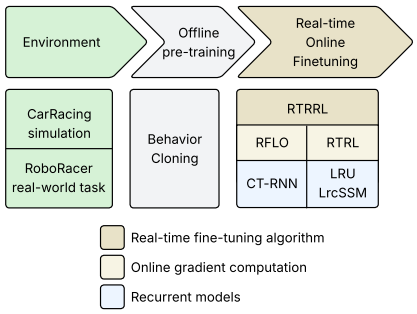


Figure 1: Overview of our method and experiments. A policy is pretrained via behavioral cloning from human control data, then fine-tuned online with RTRRL. Gradients are computed using RTRL or RFLO for diagonal or fully connected RNNs, respectively.

The traditional offline learning paradigm – in which policies are trained once and deployed without further adaptation – proves fundamentally inadequate for handling such non-stationary environments Levine et al. (2020). In such scenarios, online fine-tuning is required: the ability of deployed policies to continuously adapt to changing conditions by updating their parameters in real-time (ideally at each time step), without requiring retraining from scratch or access to large offline datasets.

Real-Time Recurrent Reinforcement Learning (RTRRL) Lemmel & Grosu (2025) offers a solution to this deployment challenge. RTRRL was initially introduced to enable online parameter

*denotes equal contribution

updates in recurrent neural network policies, supporting both Continuous-Time Recurrent Neural Networks (CT-RNNs) Funahashi & Nakamura (1993) and Linear Recurrent Units (LRUs) Orvieto et al. (2023). The framework naturally extends to Liquid-Resistance Liquid-Capacitance networks (LRCs) Farsang et al. (2024b), a class of bio-inspired continuous-time neural network models that possess desirable stability and expressivity properties for control tasks. Critically, RTRRL leverages the computation of gradients in an online manner using biologically plausible updates. This gradient computation method avoids the memory-intensive backward passes required by backpropagation through time (BPTT).

In this work, we combine offline behavioral cloning with online RTRRL fine-tuning to create adaptive policies capable of maintaining performance under distribution shift. Our experimental results demonstrate that RTRRL-based online fine-tuning naturally adjusts policy behavior to maintain task performance in the face of distribution shift, whereas fixed policies deteriorate. We demonstrate our approach in the `CarRacing` simulation environment and on a 1:10-scale real-world line-following application. While both demonstrations fall within the domain of autonomous driving, the `CarRacing` environment uses conventional RGB-based observations, whereas our real-world demonstration uses an event camera.

Our work highlights a well-rounded, bio-inspired approach that combines bio-inspired sensors and bio-inspired RNNs, together with a biologically plausible online fine-tuning algorithm. In addition – to the best of our knowledge – this is the first high-frequency closed-loop application of a machine learning algorithm with an event camera on standard (non-spiking) hardware.

Our contributions are the following:

- Extending the Real-Time Recurrent Reinforcement Learning (RTRRL) framework with a novel bio-inspired non-linear state-space model (LrcSSM).
- Proposing a hybrid learning pipeline that combines offline behavioral cloning with online real-time RTRRL-based fine-tuning, allowing deployed policies to adapt continuously (at each time step) to non-stationary environments without retraining from scratch.
- Validating the proposed approach both in simulation and on a real-world 1:10-scale autonomous driving platform, highlighting the feasibility of real-time online learning on embedded robotic systems.
- To the best of our knowledge, we present the first high-frequency non-spiking machine learning application using event camera observations in a closed-loop control setting.
- Showcasing the synergy between bio-inspired sensors, bio-inspired recurrent neural networks, and biologically plausible learning rules.

2 Background

In this section, we briefly describe the methods used in our experiments. We first introduce behavioral cloning for offline training, then present online learning for adaptive improvement using RTRRL, and finally describe the integration of recurrent models into our framework.

2.1 Behavioral Cloning

Imitation Learning deals with learning a controller for a given task from expert demonstrations. A novice policy is trained to predict the expert action from a sequence of observations from the environment. Behavioral cloning is a simple strategy for Imitation Learning where a dataset of expert demonstrations is collected, and then used to train a policy offline – via supervised learning. The policy can be either *deterministic* or *stochastic*, which determines the loss function used for training. While a deterministic policy predicts actions directly – without any notion of confidence, stochastic policies output probability distributions such as a Gaussian consisting of mean and standard deviation for each action component.

2.2 ONLINE LEARNING VIA RTRRL

Real-Time Recurrent Reinforcement Learning (RTRRL) Lemmel & Grosu (2025) is a biologically plausible online learning framework using recurrent neural networks. RTRRL combines online temporal-difference learning with eligibility traces (TD(λ)) with efficient online gradient computation via RTRL or RFLO. The framework performs all learning and weight updates in a single forward pass without separate backward phases, making it computationally and biologically more realistic than backpropagation through time (BPTT). In the next subsections, we provide brief summaries for the individual building blocks of RTRRL and present the algorithm in Appendix 1. For further algorithmic details, we refer the reader to Lemmel & Grosu (2025).

2.2.1 RTRRL RNN ARCHITECTURE

RTRRL employs a RNN backbone with linear output heads for actor π and critic $\hat{v}$ functions. At each timestep, the RNN receives observation o_t as input, together with the approximate Jacobian $\hat{J}_{t-1}$ computing a latent state representation h_t and updated Jacobian $\hat{J}_t$: $h_t, \hat{J}_t = \mathrm{RNN}_{\theta_R}(o_t, h_{t-1}, \hat{J}_{t-1})$ from which the actor outputs an action distribution $\pi_{\theta_A}(a|h)$ and the critic outputs a value estimate $\hat{v}_{\theta_C}(h)$.

2.2.2 TEMPORAL-DIFFERENCE LEARNING WITH ELIGIBILITY TRACES

RTRRL uses temporal-difference learning Sutton & Barto (2018) combined with eligibility traces (ETs) to enable online, sample-efficient updates that handle delayed rewards. After each action, the reward r_t, and past and current states h_t and h_{t+1} are used to compute the TD-error δ_t:

$$\delta_t = r_t + \gamma \hat{v}_{\theta_C,t}(h_{t+1}) - \hat{v}_{\theta_C,t}(h_t) \tag{1}$$

While standard TD(0) fails with delayed rewards, RTRRL addresses this through TD(λ), which maintains decaying traces of each parameter's gradient history. The trace e_θ decays with factor $\gamma\lambda$ where γ is the discount factor. The eligibility trace for the critic $\hat{v}$ is computed as:

$$e_{\theta_C,t} = \gamma\lambda_C \, e_{\theta_C,t-1} + \nabla_{\theta_C} \hat{v}_t(h_t) \tag{2}$$

RTRRL uses a linear value-function $\hat{v}_{\theta_C}(h_t) = w^\top h_t$ with parameters w, like in the original TD(λ), the gradient of the loss with respect to w is simply $\nabla_w \hat{v}_{\theta_C} = h_t$. The eligibility trace for the actor π is computed by taking a small step in the direction of the log of the action probability:

$$\begin{aligned} \pi_t &= \pi_{\theta_A,t}(h_t) \\ a_t &\sim \pi_t \\ e_{\theta_A,t} &= \gamma\lambda_A \, e_{\theta_A,t-1} + \nabla_{\theta_A} \log \pi_t[a_t] \end{aligned} \tag{3}$$

The eligibility trace for the RNN part is more sophisticated as it incorporates Jacobian approximations $\hat{J}$:

$$e_{\theta_R,t} = \gamma\lambda_R \, e_{\theta_R,t-1} + \hat{J}_t B \varepsilon_t \tag{4}$$

Once, the eligibility traces and the TD-error δ_t are computed, the parameters of θ_C, θ_A and θ_R are updated via:

$$\theta_{t+1} \leftarrow \theta_t + \eta \, \delta_t \, e_{\theta,t} \tag{5}$$

with their corresponding η step size and e_θ trace.

2.2.3 ONLINE GRADIENT COMPUTATION

RTRRL's key innovation is computing RNN gradients online without BPTT. This can be achieved by RTRL or RFLO.

Real-Time Recurrent Learning (RTRL) maintains an approximate Jacobian trace updated during forward computation:

$$\hat{J}_{t+1} = \hat{J}_t \left(\mathbb{I} + \nabla_{h_t} f(x_t, h_t)\right) + \bar{J}_t \tag{6}$$

allowing gradient updates at each step. However, RTRL has $O(n^4)$ complexity in neuron count n, making it impractical for large networks.

Random Feedback Local Online (RFLO) learning approximates RTRL for CT-RNNs with $O(n^2)$ complexity by enforcing biological plausibility: it avoids weight transport by using fixed random feedback matrices B (feedback alignment) instead of weight-transport gradients, and enforces locality by dropping non-local gradient terms:

$$\hat{J}^W_{t+1} \approx (1 - \tau^{-1})\, \hat{J}^W_t + \tau^{-1} \varphi'(W\xi_t)^\top \xi_t \tag{7}$$

The update is $\Delta W(t) = \hat{J}^W_t B \varepsilon_t$ using a fixed random matrix B. Prior work shows that this simplified formulation remains effective for learning Murray (2019b); Marschall et al. (2020).

2.3 Recurrent Models

Here, we provide a brief overview of the models integrated into the RTRRL framework.

CT-RNN Introduced by Funahashi & Nakamura (1993), continuous-time recurrent neural networks (CT-RNNs) can be interpreted as electrical equivalent circuits modeling electrical synapses Gerstner et al. (2014); Farsang et al. (2024a). Their dynamics are described by the following ordinary differential equation:

$$\dot{h}(t) = \frac{1}{\tau}(-h(t) + \varphi(W\xi(t))) \tag{8}$$

where $h \in \mathbb{R}^N$ denotes the hidden state, $\varphi(\cdot)$ is a nonlinear activation function, and $\tau \in \mathbb{R}^N$ is a vector of neuron-specific time constants. The augmented input $\xi \in \mathbb{R}^{I+N+1}$ concatenates the external input $x(t) \in \mathbb{R}^I$, the current hidden state at time step t, and a constant bias term. The weight matrix $W \in \mathbb{R}^{N\times(I+N+1)}$ jointly parameterizes input, recurrent, and bias connections. The network output is obtained via a linear readout, $y(t) = W_{out}h(t)$.

LRU Linear Recurrent Units (LRUs) – introduced by Orvieto et al. (2023) – describe its latent dynamics through a linear system, enabling highly efficient state updates. They define the following state-space model (SSM) equations:

$$\dot{h}(t) = Ah(t) + Bx(t) \tag{9}$$

where $A \in \mathbb{C}^{N\times N}$ is a diagonal matrix, and $B \in \mathbb{C}^{N\times I}$, $C \in \mathbb{C}^{O\times N}$, and $D \in \mathbb{R}^{O\times I}$ are learnable parameter matrices. The hidden state $h_t \in \mathbb{C}^N$ is complex-valued. The output $y(t) = \Re\left[Ch(t)\right] + Dx(t)$ is obtained by combining the real part of the latent state with the input $x(t)$ at time t.

LRC Liquid-Resistance Liquid-Capacitance models (LRCs) Farsang et al. (2024b) and their diagonal variant, LrcSSMs Farsang et al. (2025), have an intermediate position between CT-RNNs and LRUs. From a biological perspective, LRCs are bio-inspired models of chemical synapses, whereas CT-RNNs can be interpreted as models of electrical synapses. A key distinction between the two lies in their treatment of temporal dynamics: CT-RNNs employ fixed time constants, while LRCs introduce input- and state-dependent time constants, allowing each neuron to adapt its temporal behavior dynamically.

From a state-space modeling viewpoint, both LRUs and LrcSSMs fall within the SSM framework. While LRUs are characterized by purely linear recurrence, LrcSSMs extend this formulation by introducing nonlinear state dynamics. Despite this increased expressiveness, LrcSSMs retain computational efficiency due to their diagonal structure, which enables decoupled and parallelizable state updates. The dynamics of LRCs can be expressed as

$$\dot{h}(t) = A_{h,x}(t)\, h(t) + b_{h,x}(t) \tag{10}$$

where $A_{h,x}(t) \in \mathbb{R}^{N\times N}$ is a diagonal matrix and $b_{h,x}(t) \in \mathbb{R}^N$ is a vector-valued function, capturing biologically interpretable, state- and input-dependent dynamics.

3 Experimental Setup

3.1 Integrating LRC into the Online Fine-tuning Framework

The biological plausibility of RTRRL stems form the fact that BPTT is avoided when computing gradients for the recurrent neural network. As mentioned above, gradients for recurrent models with

diagonal connectivity can be computed exactly – and without a loss in performance – using RTRL (Eq. 6).

The diagonalized version of LRC (LrcSSMs) can be used as a drop-in replacement of LRU in the original RTRRL algorithm. The resulting algorithm retains the benefits of cheap computation of exact gradients, while also regaining the biological interpretation that the LRU model lacks.

3.2 Model Architecture

Figure 2 shows the model structure used for our experiments. Core components are the convolutional encoder and the recurrent policy, which are pretrained first using supervised learning, and later fine-tuned using reinforcement learning. The convolutional decoder and the recurrent critic are used only during pretraining and fine-tuning respectively.

During pretraining, the convolutional encoder is part of an autoencoder that is trained to minimize the image reconstruction loss $\mathcal{L}_{rec} = ||o_t - \hat{o}_t||$. The actor is modeled as single-layer recurrent neural network and receives the encoded observation x_t as input. It's output is transformed by a fully connected layer that maps to a number of outputs equal to double the number of action dimension. The outputs correspond to the mean and standard deviation of a Gaussian distribution that is transformed by a hyperbolic tangent bijector, and scaled to the maximum action magnitude afterwards. This setup ensures that the policy can only predict admissible actions. The policy is trained by maximizing the log-probability of the action in the dataset under the predicted distribution $\mathcal{L}_{act} = -\log P(a_t|x_t)$. The combined loss for pretraining is the sum of the action prediction loss and the weighted reconstruction loss.

$$\mathcal{L} = \mathcal{L}_{act} + \eta_{rec}\mathcal{L}_{rec}$$

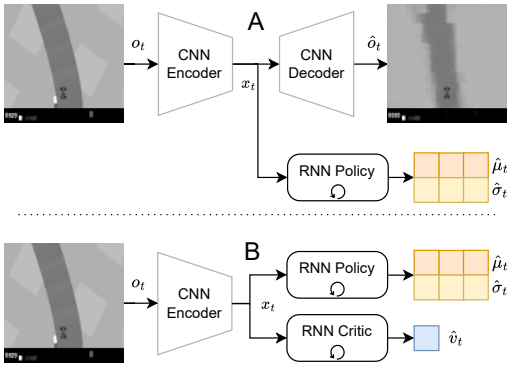


Figure 2: Model architecture. A: Pretraining – a CNN autoencoder is used to encode the image observation o_t; the encoded vector x_t serves as input to the RNN policy predicting action distribution $\mathcal{N}(\hat{\mu}_t, \hat{\sigma}_t)$. B: Fine-tuning using RL – a RNN value function is used for predicting value estimates $\hat{v}_t$ from x_t.

For fine-tuning via RTRRL, a recurrent value function is randomly initialized and used alongside the CNN encoder and the RNN policy. The RNN Critic receives the same encoded vector x_t as input and it's output is mapped by an all-to-one fully connected layer. The Critic's output corresponds to a deterministic prediction of the expected cumulative state-value with discounting. Note that – contrary to Lemmel & Grosu (2025), who used a shared RNN encoder followed by linear policy and value functions – we use a separate RNN for each function.

3.3 Parameter Change Penalty

When fine-tuning a policy on a single environment with an effective batch-size of 1, overfitting is likely to appear. To combat this issue, we add an additional penalty to the policy loss that penalizes the L_2 norm of the difference between the pretrained and the current policy parameters, similar to Lemmel et al. (2025). The resulting optimization objective is $(\mathcal{L} + L_\theta)$ with $L_\theta = \beta\ ||\theta_{pre} - \theta_t||_2$.

3.4 Simulation Environment

For our simulation experiments, we introduce a modified version of the `CarRacing` environment from OpenAI Gym: we augment the environment with an additional penalty term that penalizes deviations of the vehicle's wheels from the track center. Specifically, each segment of the track is defined by a keypoint representing its center of mass. We compute the Euclidean distance of the car's center to all keypoints of the track and take the minimum of these distances. This deviation measure is scaled and subtracted as a penalty term from the original environment reward. The modified reward encourages agents to maintain a trajectory that keeps the vehicle's wheels close to the track center, effectively shaping the policy toward safer and more comfortable driving behavior.

We first collected human demonstration data by letting an average human player complete three laps in the environment. Each trajectory consists of a sequence of 1000 image observations paired with the corresponding (discrete) control actions. This dataset serves as the source of supervised pretraining for the agent. For each model type, we initialize parameters with five different random seeds and pre-train each on the recorded human trajectories. These pretrained models are subsequently used as initializations for the fine-tuning phase, where the agents continue training within the modified environment.

3.5 Real-World Deployment

In our real-world deployment, we used a RoboRacer car – an autonomous platform based on a 1:10-scale racecar with an Asus-NUC-based compute platform – equipped with a Sony/Prophesee IMX636 Dynamic Vision Sensor (DVS). The compute platform is connected to a motor and steering controller (see Fig. 7a).

Unlike an RGB optical sensor, the DVS captures changes in pixel intensity and generates a stream of intensity change events, triggered when the intensity exceeds a predefined threshold. We use aggregated events to generate frame-based representations for use with conventional (non-spiking) neural networks. The representation we use, *event frames*, retains only the polarity of each event per pixel and discards all sub-frame timing information. Figure 7 shows this representation with the corresponding RGB frame.

Typically, filtering is applied to each representation to remove noise from the event stream, and some representations also flatten event polarities. Gallego et al. (2022) describe the different representations in more detail and typical algorithms for event data.

In the `LineTracking` experiment, we use the dataset collected by Resch et al. (2025) from real-world driving sessions conducted by non-expert drivers. The dataset consists of 500772 event frames, generated at 100 Hz, each consisting of event data representations of size 128 $\times$ 64 $\times$ 2 paired with the corresponding control actions $\in \mathbb{R}^3$. Similar to the `CarRacing` setup, this dataset is used for supervised pretraining of each model across multiple random seeds before fine-tuning within the simulation environment.

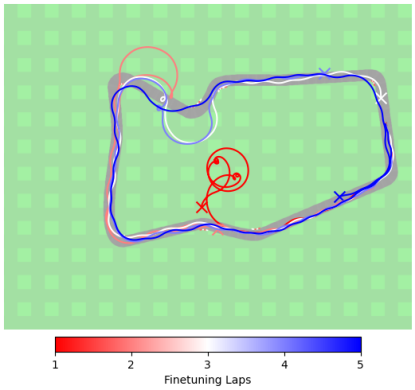


Figure 3: Trajectories over five fine-tuning laps: the car initially leaves the track (red) but improves each lap, eventually completing it (blue).

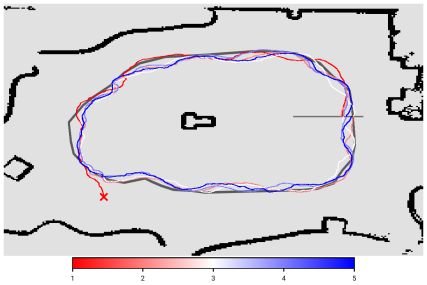


Figure 4: Trajectories of five laps of fine-tuning of an LrcSSM policy on the real-world application. Please note that the trajectories have been smoothed slightly to account for noise introduced by the state estimation.

3.5.1 Fine-Tuning

For fine-tuning, we use a reward function based on the agent's distance to the reference line, the heading difference, and the rate of change of the steering angle. To allow for slight deviations, each reward modality uses a slack value that indicates whether the task is achieved and is scaled with a weight for each reward modality. To determine the car's pose, we first map the experimental area with `slam_toolbox`[1] and record key points along the marked line. Using the mapped environment, we estimate the agent's pose via a Monte Carlo-based approach inspired by Walsh & Karaman (2017).

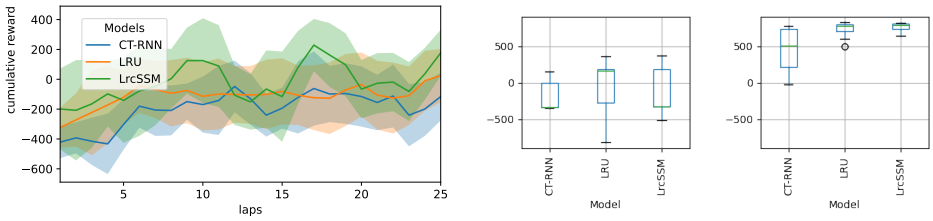
[1] `https://github.com/SteveMacenski/slam_toolbox`

4 RESULTS

The aim of this paper was to show that models that were pretrained on limited data can be fine-tuned using RTRRL. Our results show that this holds true for both our experiments. In this section, we first discuss pretraining and fine-tuning results for the `CarRacing` simulation, followed by our results for the real-world `LineTracking` application.

4.1 CARRACING

Pretraining results Figure 8 in the Appendix shows the validation loss curves for pretraining the `CarRacing` policies, aggregated by model type. Figure 5b shows the aggregated evaluation reward for the pretrained models when employed in the environment without any tuning. At this point, all pretrained models performed similarly.



(a) (b) (c)

Figure 5: (a) Lineplot of median evaluation reward on three different tracks for all pretrained models, aggregated per type. Shaded regions depict the standard deviation. (b)-(c) Boxplots of evaluation reward on three different tracks for five different pretrained models, aggregated per type. (b) shows rewards before fine-tuning – (c) after.

Fine-Tuning After pretraining, we fine-tuned the models using RTRRL by updating the policy and value functions after each step in the environment. Since RTRRL – the algorithm we use for fine-tuning – is sensible to hyperparameter choices, we conducted a number of hyperparameter sweeps to find a good configuration. We first did two rough grid searches over learning rates for the actor and critic followed by a grid search over entropy regularization strengths. Finally, we did a large-scale Bayesian hyperparameter tuning run that encompassed the learning rates and decay schedules for them, as well as the strength for the parameter change penalty introduced above.

We found that a learning rate around 10^{-6} for the actor is best. The critic learning rate appeared to be of less importance with values in the range of 10^{-3} to 10^{-5} being acceptable. Entropy regularization did show negligible impact overall and we decided to move forward without any. Finally, the best choice for the parameter change penalty coefficient resulted to be 10^{-5}. Table 1 shows the final hyperparameter configuration for RTRRL that was used for the remainder of the experiments.

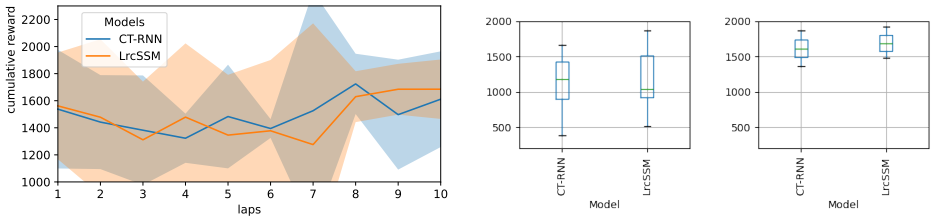
Figure 5a shows the median cumulative lap reward over 10 laps of fine-tuning for 5 random seeds per model type. A clear upwards trend is visible for all the model types, with LRCs improving the most.

Figure 5c shows the aggregated evaluation reward for the pretrained models after 10 laps of fine-tuning. Note, that during evaluation, the mode of the predicted action policy is executed, as opposed to sampling from it during fine-tuning. All models have improved significantly over the initial reward before fine-tuning. Moreover, LRC results improved the most and showed very consistent results across all models and seeds.

To illustrate the effect of fine-tuning in the `CarRacing` environment, we picked one LrcSSm policy for plotting lap trajectories shown in Figure 3. While the policy was not able to successfully complete the track at first, it quickly adjusted for it within five laps.

4.2 REAL-WORLD DEPLOYMENT

For the `LineTracking` task, the agent is placed on a pre-determined starting point on a line marked on the floor with clearly distinguishable tape. The goal of this task is to follow the line as closely as possible, while avoiding rapid steering inputs. For evaluation purposes, we chose an arbitrary point on the line as the start/finish line for this task, to segment the trajectories of the agent into laps. In this instance, we picked the middle of a short straight, as most policies successfully completed this part of the circuit.



(a) (b) (c)

Figure 6: (a) Median cumulative rewards per lap for the `LineTracking` experiment. Shaded regions show the standard deviation. (b)-(c) Boxplots of cumulative rewards for the `LineTracking` experiment of five different pretrained models, aggregated per type. (b) shows rewards before fine-tuning – (c) after.

Pretraining results Figure 9 shows the validation loss curves that were recorded during pretraining of the `LineTracking` dataset. While LrcSSMs and CT-RNNs achieved comparable results after 2000 epochs, LRUs appeared to be significantly worse. This shortcoming of LRUs became evident when deploying the pretrained models to the real-world car, as none of theses models were able to complete a lap without a large number of human interventions. A boxplot of the number of human interventions is shown in Figure 13 in the Appendix. We suspect that the linear state-space model semantics of the LRU model are not expressive enough to satisfactorily capture the steering behavior when used as a single-layer RNN. Due to the poor performance of LRUs, we omitted them from the fine-tuning altogether. Figure 6b shows the cumulative rewards of the pretrained models over three laps without any fine-tuning.

Fine-Tuning For the real-world fine-tuning experiment, we selected the three best pretrained models for LrcSSMs and CT-RNNs and deployed them on the autonomous vehicle. A human operator monitored the car during the experiment and intervened as soon as a significant deviation from the track was observed, maneuvering the car back to a point where the camera has last had clear visibility of the line.

Figure 6c shows the cumulative rewards achieve by the models after fine-tuning and Figure 6a shows the cumulative rewards over 10 laps aggregated per model. Surprisingly, we witnessed an immediate improvement in the first lap, hinting at an instantaneous effect of the fine-tuning. From there, the models' performance seemed to stagnate at first, followed by a significant reduction in standard deviation over the course of the fine-tuning. As an example of the effectiveness of RTRRL on the real-world task, Figure 4 shows the trajectories of one of the LrcSSM policies. While the policy initially fails to follow the line (in the lower left of the figure), it eventually follows it in successive laps. Further laps also show reduced oscillations around the target line.

5 DISCUSSION

In this paper, we asked the question whether online fine-tuning of a pretrained policy can be done by using RTRRL – a biologically plausible forward-only reinforcement learning algorithm. To this end, we created a static dataset of human control inputs, and employed behavioral cloning to pretrain a combined convolutional and recurrent policy that predicts actions from pixels. We then fine-tune the policy online using RTRRL – updating parameters after each step – to improve the policy on the fly. Our experimental results show substantial improvements in the simulated `CarRacing` environment that carry over to our real-world `LineTracking` example.

ACKNOWLEDGMENTS

J.L. was partially supported by the Doctoral College Resilient Embedded Systems (DC-RES) of TU Wien. F.R. and R.G. have received funding from the European Union's Horizon Europe research and innovation program with Grant Agreement No. 10039070. M.F. has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034277. Computational results have been achieved in part using the Vienna Scientific Cluster (VSC).

REFERENCES

Sotirios N. Aspragkathos, Evangelos Ntouros, George C. Karras, B. Linares-Barranco, T. Serrano-Gotarredona, and Kostas J. Kyriakopoulos. An event-based tracking control framework for multirotor aerial vehicles using a dynamic vision sensor and neuromorphic hardware. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6349–6355, 2023. doi: 10.1109/IROS55552.2023.10342437.

Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

Thierry Catfolis. A method for improving the real-time recurrent learning algorithm. *Neural networks*, 6(6):807–821, 1993.

Guang Chen, Hu Cao, Jorg Conradt, Huajin Tang, Florian Rohrbein, and Alois Knoll. Event-based neuromorphic vision for autonomous driving: A paradigm shift for bio-inspired visual sensing and perception. *IEEE Signal Processing Magazine*, 37(4):34–49, 2020. doi: 10.1109/MSP.2020.2985815.

Keru Chen, Honghao Wei, Zhigang Deng, and Sen Lin. Towards fast safe online reinforcement learning via policy finetuning. *Transactions on Machine Learning Research*, 2026.

Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, 2018.

Mónika Farsang, Mathias Lechner, David Lung, Ramin Hasani, Daniela Rus, and Radu Grosu. Learning with chemical versus electrical synapses does it make a difference? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15106–15112. IEEE, 2024a.

Mónika Farsang, Sophie A. Neubauer, and Radu Grosu. Liquid Resistance Liquid Capacitance Networks. In *The First Workshop on NeuroAI @ NeurIPS2024*. arXiv, November 2024b.

Mónika Farsang, Ramin Hasani, Daniela Rus, and Radu Grosu. Parallelization of non-linear state-space models: Scaling up liquid-resistance liquid-capacitance networks for efficient sequence modeling. *arXiv preprint arXiv:2505.21717*, 2025.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, January 1993. ISSN 0893-6080. doi: 10.1016/S0893-6080(05)80125-X.

Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022. doi: 10.1109/TPAMI.2020.3008413.

Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, Cambridge, 2014. ISBN 978-1-107-44761-5. doi: 10.1017/CBO9781107447615.

Ezgi Korkmaz. A survey analyzing generalization in deep reinforcement learning. *arXiv preprint arXiv:2401.02349*, 2024.

Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Addressing distribution shift in online reinforcement learning with offline datasets, 2021.

Julian Lemmel and Radu Grosu. Real-time recurrent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18189–18197, 2025.

Julian Lemmel, Manuel Kranzl, Adam Lamine, Philipp Neubauer, Radu Grosu, and Sophie Neubauer. Online Fine-Tuning of Carbon Emission Predictions using Real-Time Recurrent Learning for State Space Models, August 2025.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Dianzhao Li and Ostap Okhrin. A platform-agnostic deep reinforcement learning framework for effective sim2real transfer towards autonomous driving. *Communications Engineering*, 3(1):147, 2024.

Changyu Liu, Yiyang Liu, Taowen Wang, Qiao Zhuang, James Chenhao Liang, Wenhao Yang, Renjing Xu, Qifan Wang, Dongfang Liu, and Cheng Han. On-the-fly vla adaptation via test-time reinforcement learning. *arXiv preprint arXiv:2601.06748*, 2026.

Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5419–5427, 2018.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A Unified Framework of Online Learning Algorithms for Training Recurrent Neural Networks. *Journal of Machine Learning Research*, 21 (135):1–34, 2020. ISSN 1533-7928.

James M Murray. Local online learning in recurrent networks with random feedback. *eLife*, 8: e43299, may 2019a. ISSN 2050-084X. doi: 10.7554/eLife.43299.

James M Murray. Local online learning in recurrent networks with random feedback. *eLife*, 8: e43299, May 2019b. ISSN 2050-084X. doi: 10.7554/eLife.43299.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, pp. 26670–26698. JMLR.org, 2023.

Federico Paredes-Vallés, Jesse J Hagenaars, Julien Dupeyroux, Stein Stroobants, Yingfu Xu, and Guido CHE de Croon. Fully neuromorphic vision and control for autonomous drone flight. *Science Robotics*, 9(90):eadi0591, 2024.

Yansong Peng, Yueyi Zhang, Zhiwei Xiong, Xiaoyan Sun, and Feng Wu. Get: Group event transformer for event-based vision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6038–6048, 2023.

Felix Resch, Mónika Farsang, and Radu Grosu. Mmdvs-lf: Multi-modal dynamic vision sensor and eye-tracking dataset for line following, 2025.

Stephane Ross and Drew Bagnell. Efficient Reductions for Imitation Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, March 2010.

Amirhossein Rostami, Bernhard Vogginger, Yexin Yan, and Christian G Mayr. E-prop on spinnaker 2: Exploring online learning in spiking rnns on neuromorphic hardware. *Frontiers in Neuroscience*, 16:1018006, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018. ISBN 0-262-03924-9.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Antonio Vitale, Alpha Renner, Celine Nauer, Davide Scaramuzza, and Yulia Sandamirskaya. Event-driven vision and control for uavs on a neuromorphic chip. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 103–109, 2021. doi: 10.1109/ICRA48506.2021.9560881.

Kevin L Voogd, Jean Pierre Allamaa, Javier Alonso-Mora, and Tong Duy Son. Reinforcement learning from simulation to real world autonomous driving using digital twin. *IFAC-PapersOnLine*, 56 (2):1510–1515, 2023.

Corey Walsh and Sertac Karaman. Cddt: Fast approximate 2d ray casting for accelerated localization. abs/1705.01167, 2017.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Sam Witty, Jun K Lee, Emma Tosch, Akanksha Atrey, Kaleigh Clary, Michael L Littman, and David Jensen. Measuring and characterizing generalization in deep reinforcement learning. *Applied AI Letters*, 2(4):e45, 2021.

Ruihan Wu, Chuan Guo, Yi Su, and Kilian Q Weinberger. Online adaptation to label distribution shift. *Advances in Neural Information Processing Systems*, 34:11340–11351, 2021.

Zehao Xiao and Cees GM Snoek. Beyond model adaptation at test time: A survey. *arXiv preprint arXiv:2411.03687*, 2024.

Shoukai Xu, Mingkui Tan, Liu Liu, Zhong Zhang, Peilin Zhao, et al. Test-time adapted reinforcement learning with action entropy regularization. In *Forty-second International Conference on Machine Learning*, 2025.

Xu Zheng, Yexin Liu, Yunfan Lu, Tongyan Hua, Tianbo Pan, Weiming Zhang, Dacheng Tao, and Lin Wang. Deep learning for event-based vision: A comprehensive survey and benchmarks. *arXiv preprint arXiv:2302.08890*, 2023.

Ruiqi Zhu, Endong Sun, Guanhe Huang, and Oya Celiktutan. Efficient continual adaptation of pretrained robotic policy with online meta-learned adapters. *arXiv preprint arXiv:2503.18684*, 2025.

Nicolas Zucchet, Robert Meier, Simon Schug, Asier Mujika, and Joao Sacramento. Online learning of long-range dependencies. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10477–10493. Curran Associates, Inc., 2023.

A Related Work

Interactive Imitation Learning Behavioral cloning can lead to unsatisfactory performance when deploying the novice due to overfitting on the training set. The DAgger algorithm Ross & Bagnell (2010) provides a solution to this problem that relies on interactively extending the initial dataset by allowing the pretrained novice to act at times during task execution, with the expert providing action labels for the resulting observations. This leads to the inclusion of trajectories where the expert is forced to recover from suboptimal states. DAgger approaches still rely on offline supervised learning when updating policy parameters.

Existing Approaches to Online Adaptation Existing approaches to online adaptation in deployed policies span several paradigms that differ fundamentally from our real-time recurrent approach. Meta-learning trains policies to rapidly adapt to new tasks or dynamics via few-shot gradient updates or context inference Finn et al. (2017), but requires task-specific training data and often suffers from meta-overfitting in continuous control settings Zhu et al. (2025). Test-time adaptation techniques enable parameter updates during inference using self-supervised objectives or reward shaping Xiao & Snoek (2024); Liu et al. (2026), with recent work in robotics applying test-time RL to vision-language action models for on-the-fly policy refinement in unseen environments Liu et al. (2026); Xu et al. (2025). Online label shift adaptation from supervised learning dynamically reweights training examples to match evolving test-time label distributions Wu et al. (2021), more general approaches like balanced offline-online replay address full state-action shift Lee et al. (2021).

Online Reinforcement Learning Methods Online reinforcement learning approaches used for fine-tuning policies typically perform rollouts of full episodes with the current policy, store the collected transitions in a replay buffer, and then update the policy by sampling mini-batches from this buffer at discrete training intervals Chen et al. (2026). This paradigm introduces an intrinsic latency between interaction and learning, since gradient updates occur only after sufficient experience has been accumulated. Alternatively, the distribution shift problem can be addressed by fine-tuning on offline datasets, maintaining separate replay buffers for offline and online data and carefully balancing the sampling ratio between them during policy updates Lee et al. (2021). Compared to these approaches, Real-Time Recurrent Reinforcement Learning (RTRRL) Lemmel & Grosu (2025) performs parameter updates at every time-step in a fully online manner, directly propagating temporal-difference errors through a recurrent backbone via real-time automatic differentiation. This design eliminates the need to wait for episodes to complete or to form mini-batches, thereby reducing update latency and enabling strictly causal, fine-grained adaptation in partially observable environments.

Online Gradient Computation A central challenge for fully online learning in recurrent networks is how to compute or approximate gradients in a causal, memory-efficient way. Real-Time Recurrent Learning (RTRL) Williams & Zipser (1989) computes the exact gradient of the loss with respect to all recurrent weights in a strictly forward manner but at the cost of $O(n^4)$ memory complexity in the number of units n, which makes it impractical for large networks Williams & Zipser (1989); Catfolis (1993). To address this, e-prop Bellec et al. (2020) decomposes the gradient into products of neuron-specific eligibility traces and top-down learning signals, providing an online, local learning rule for spiking and rate-based recurrent networks that approaches the performance of BPTT while being far more suitable for neuromorphic hardware Bellec et al. (2020); Rostami et al. (2022). Similarly, Random Feedback Local Online Learning (RFLO) Murray (2019a) derives an approximate gradient rule for RNNs that is local in space and time, and replaces exact backpropagated errors with fixed random feedback weights, yielding a biologically plausible, online update rule of $O(n^2)$ that performs competitively with BPTT on short horizon tasks Murray (2019a); Catfolis (1993); Marschall et al. (2020). Most importantly, RTRL becomes equally efficient with models of diagonal connectivity, as demonstrated in Zucchet et al. (2023).

RTRRL builds on this RTRL-style, step-wise gradient propagation to update policy parameters at every time-step directly from the incoming reward signal, thereby inheriting the strict online character of RTRL while tailoring the update rule to temporal-difference RL rather than supervised sequence prediction Lemmel & Grosu (2025).

Event-Based Vision in Machine Learning Event-based vision leverages neuromorphic sensors for high-speed, low-latency processing in robotics and control, building on early DVS hardware and algorithms for optic flow and tracking Gallego et al. (2022); Maqueda et al. (2018). Deep learning advancements, including transformers like GET Peng et al. (2023) and comprehensive benchmarks, have enabled tasks such as object detection and deblurring from sparse event streams Zheng et al. (2023). Its applications in autonomous vehicles Chen et al. (2020) often use dedicated neuromorphic chips running spiking neural networks Vitale et al. (2021); Aspragkathos et al. (2023); Paredes-Vallés et al. (2024). In contrast, in this work we target classical hardware and employ efficient recurrent models, demonstrating that they can also operate on DVS data.

B RTRRL Algorithm Details

Algorithm 1 below outlines the steps of RTRRL.

Algorithm 1 Real-Time Recurrent Reinforcement Learning Lemmel & Grosu (2025)

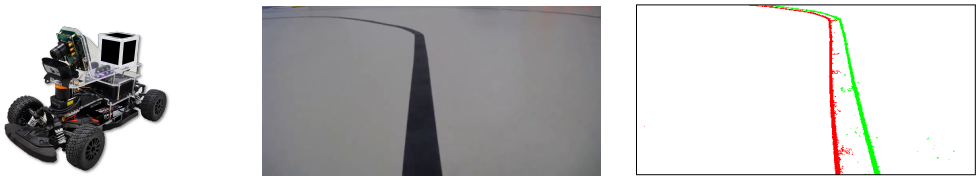
Require: Linear actor policy: $\pi_{\theta_A}(a|h)$, linear critic value-function: $\hat{v}_{\theta_C}(h)$, and recurrent layer: $\text{RNN}_{\theta_R}([o, a, r], h, \hat{J})$

$\theta_A, \theta_C, \theta_R \leftarrow$ initialize network parameters
$B_A, B_C \leftarrow$ initialize feedback matrices
$h, e_A, e_C, e_R \leftarrow \mathbf{0}$
$o \leftarrow$ reset environment
$h, \hat{J} \leftarrow \text{RNN}_{\theta_R}([o, \mathbf{0}, 0], h, \mathbf{0})$
$v \leftarrow \hat{v}_{\theta_C}(h)$
while not done **do**
  $\pi \leftarrow \pi_{\theta_A}(h)$
  $a \leftarrow$ sample(π)
  $o, r \leftarrow$ take action a
  $h', \hat{J}' \leftarrow \text{RNN}_{\theta_R}([o, a, r], h, \hat{J})$
  $e_C \leftarrow \gamma\lambda_C e_C + \nabla_{\theta_C}\hat{v}$
  $e_A \leftarrow \gamma\lambda_A e_A + \nabla_{\theta_A} \log \pi[a]$
  $g_C \leftarrow B_C \mathbf{1}$
  $g_A \leftarrow B_A \nabla_\pi \log\ \pi[a]$
  $e_R \leftarrow \gamma\lambda_R e_R + \hat{J}(g_C + \eta_A g_A)$
  $v' \leftarrow \hat{v}_{\theta_C}(h')$
  $\delta \leftarrow r + \gamma v' - v$
  $\theta_C \leftarrow \theta_C + \alpha_C \delta e_C$
  $\theta_A \leftarrow \theta_A + \alpha_A \delta e_A$
  $\theta_R \leftarrow \theta_R + \alpha_R \delta e_R$
  $v \leftarrow v', \quad h \leftarrow h', \quad \hat{J} \leftarrow \hat{J}'$
end while

Hyperparameter	Value
Discount factor γ	0.99
Eligibility trace decay factor λ	0.95
Actor learning rate α_A	10^{-6}
Critic learning rate α_C	10^{-5}
Entropy rate η_H	0
Parameter change penalty rate η_P	10^{-5}

Table 1: RTRRL hyperparameter configuration for fine-tuning.

C Real-World Deployment



(a) RoboRacer car. (b) RGB image. (c) Event frame.

Figure 7: (a) RoboRacer car equipped with Sony/Prophesee IMX636 sensor for the real-world deployment of the proposed algorithm. (b) RGB frame and the corresponding (c) DVS event frame representation.

D Additional Pre-training Results

We show the validation loss curves from pre-training on the `CarRacing` and `LineTracking` dataset in Fig. 8 and 9, respectively.

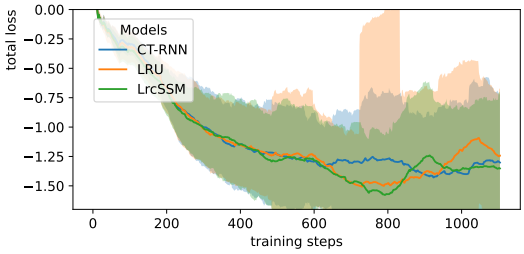


Figure 8: Mean validation loss during pretraining on the `CarRacing` dataset. Shown is the mean reward of five seeds per model type with standard deviation shown as shaded regions.

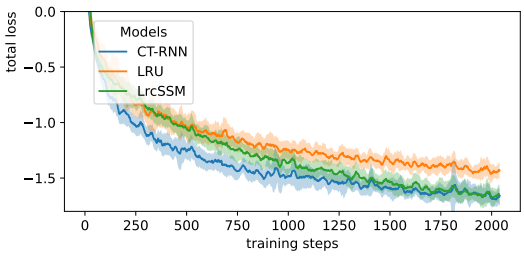


Figure 9: Mean validation loss during pretraining on the `LineTracking` dataset. Shown is the mean reward of five seeds per model type with standard deviation shown as shaded regions. CT-RNN

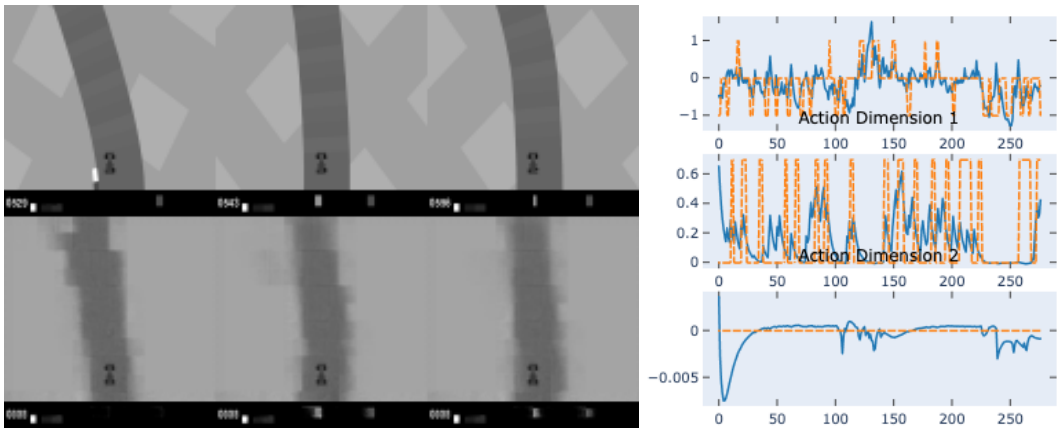


Figure 10: Left: Exemplary decoded images predicted by the CNN autoencoder after pretraining on the `CarRacing` dataset. Right: Actions predicted by the pretrained policy.

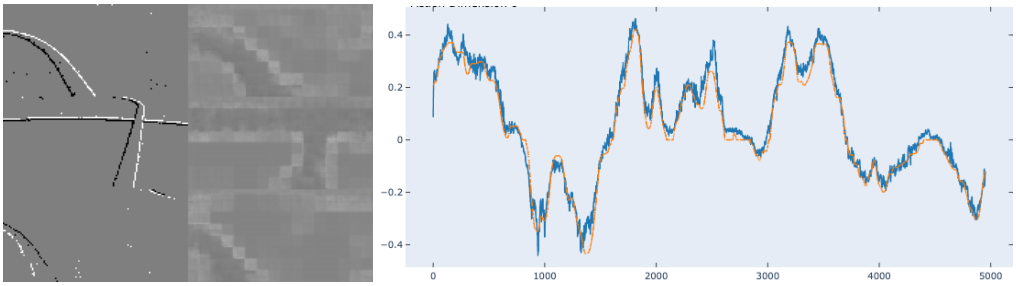
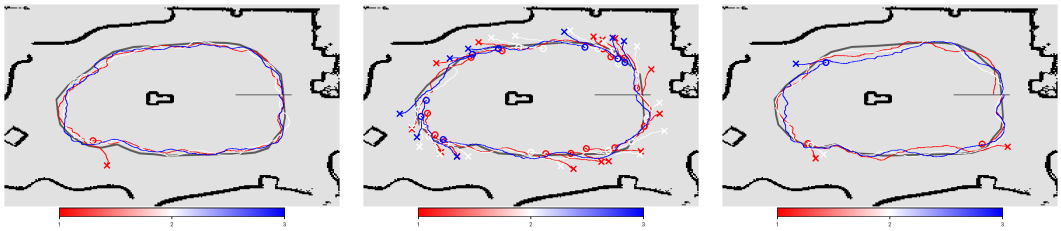


Figure 11: Left: Exemplary decoded images predicted by the CNN autoencoder after pretraining on the `LineTracking` dataset. Right: Actions predicted by the pretrained policy.

E *LineTracking* Fine-tuning Progress

E.1 Pre-trained Trajectories



(a) CT-RNN. (b) LRU. (c) LRC.

Figure 12: Trajectories of policies without fine-tuning. Crosses indicate manual intervention, and circles indicate the resumption by the policy.

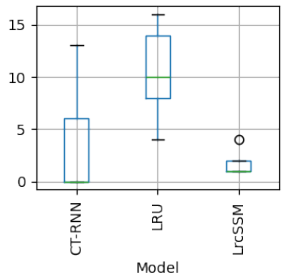
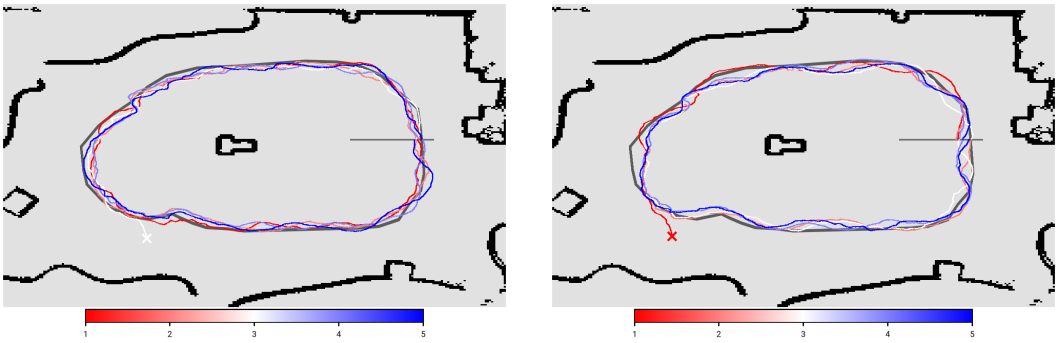


Figure 13: Number of interventions per lap of the pre-trained `LineTracking` models.

E.2 Policies During Fine-tuning



(a) CT-RNN. (b) LRC.

Figure 14: Trajectories of policies with fine-tuning. Laps that required manual intervention were terminated upon intervention.