

Article

What Should a Network-Aware Agent Observe? A Systematic Analysis of Metadata Features for Reinforcement Learning Control over Wireless Links

André Gilerson ^{1,*}  and Robert H. Schmitt ^{1,2}¹ Fraunhofer Institute for Production Technology IPT, Steinbachstr. 17, 52074 Aachen, Germany² Laboratory for Machine Tools and Production Engineering (WZL), Department Intelligence in Quality Sensing, RWTH Aachen University, Campus-Boulevard 30, 52074 Aachen, Germany

* Correspondence: andre.gilerson@ipt.fraunhofer.de; Tel.: +49-241-8904230

Abstract

Recent work has shown that exposing Deep Reinforcement Learning (DRL) agents to network impairments during training allows them to perform better when deployed in realistic networks than ones trained under idealized conditions. We investigate systematically whether and what kind of information about the state of the network is beneficial to add to the agent's observations. For this, we train PPO and SAC agents under a fixed impairment profile while varying the augmented observation space across eight feature groups on two common control tasks with different impairment sensitivity often seen in DRL research (CheetahRun and FingerTurnHard implemented in MuJoCo). Our results show that per-observation information about whether the packet was dropped or delayed recovers most of the performance of the trained algorithm, while adding statistics about latency, jitter, and packet loss provides little benefit or even destabilizes the training for PPO. SAC seems to be largely insensitive to the additional metadata. A window-size ablation shows that latency and jitter statistics provide no benefit across any tested window size, while the effects of the loss statistics are strongly window dependent. These results argue against a universal network metadata observation augmentation vector and in favor of algorithm- and task-specific feature selection.

Keywords: deep reinforcement learning; networked control system; network impairments; latency; jitter; packet-loss



Academic Editors: Chang Duan and Chengzhi Yuan

Received: 8 June 2026

Revised: 5 July 2026

Accepted: 9 July 2026

Published: 16 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

With the ongoing progress of Industry 4.0, cloud robotics, and autonomous systems, wireless control systems are becoming increasingly common [1,2]. Deep Reinforcement Learning (DRL) has emerged as a strong contender to create control policies. However, this holds only under idealized conditions, where observations and actions arrive immediately and reliably at the target [3,4]. Small perturbations or impairments to the communication, e.g., latency even in the tens of milliseconds or single digit rates of packet loss, can dramatically decrease the performance of DRL policies that were trained without special precautions [5,6].

Multiple papers have shown that simply training the agent under the same impaired conditions that it will face during deployment will dramatically increase its performance [7,8]. This raises a natural follow-up question: What, if any, information about the network state improves the agent's performance if included in the observation vector? Prior

work has shown, for example, that adding the instantaneous latency to the observation vector can improve the agent's performance [6,9]. We build on this work by considering additional augmentations to the observation vector, such as:

- Binary flag whether observation is fresh;
- Time since the last fresh observation;
- Windowed statistical summaries of latency (mean and jitter) or Loss rates.

We perform ablation tests of these signals, as each of them contains different information about a partially observable, time-varying communication channel. Their usefulness depends on both the setup of the channel (e.g., whether it is more loss- or latency-dominated) and the algorithm's and the task's sensitivity to impairments. Holding the network impairment profile fixed, we train Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) agents on two MuJoCo-based tasks common in DRL research, CheetahRun (17-dimensional locomotion) and FingerTurnHard (12-dimensional manipulation), while varying the augmented observation space across eight feature groups. We perform the ablation tests once with a purely statistical latency and packet loss model, and once with a location-dependent configuration, since this better resembles real-world wireless communication networks.

Our contributions are:

- A taxonomy of network-derived observation features, motivated by what is computable from the per-step network status emitted by wireless communication networks;
- A systematic ablation with training runs, reporting effect sizes for each feature group against a common baseline, with cross-task and cross-algorithm analysis;
- A window-size sensitivity analysis of all window-derived feature groups.

The paper proceeds with related work in Section 2, and the methodology of the ablation analysis and the experimental setup in Section 3. Section 4 describes the results, with a discussion in Section 5 and a conclusion in Section 6.

2. Background and Related Work

Usually, DRL assumes the entire control loop consisting of the controller/agent, system, and sensor to be a Markov Decision Process (MDP) [10]. This assumption is violated once the three components are connected via a randomly delayed and/or lossy communication network [11]. Altman and Nain showed that any MDP with constant delay can be reformulated as a standard MDP by augmenting the system state with a buffer of pending actions and observations, at a cost of exponential increase in the state size with increased delay [12]. Walsh et al. proved that planning in these MDPs is an NP-hard problem and therefore requires other approaches, such as model-based simulations [13]. Real-world communication networks, especially wireless networks, suffer, however, from non-deterministic delays, which were formalized by Bouteiller et al. as Random-Delay MDPs (RDMDP) [6]. Schenato et al. established limits on the stability of linear estimators under lossy communication conditions, showing that stability can no longer be guaranteed above critical Loss rates [3]. The loss of either observations or actions effectively turns the MDP into a partially observable MDP (POMDP): when a packet is dropped, either the agent or the system has to act on a held-over or imputed observation or action, respectively [14].

Various adaptations to regular DRL algorithms have been proposed to adapt to RDMDPs and POMDPs. Hausknecht and Stone propose replacing feedforward neural networks with Long Short-Term Memory (LSTM) networks, allowing the agents to use previous observation history to partially recover system state [15]. Ni et al. show that recurrent formulations outperform the standard model-free algorithms across a broad benchmark of tasks, but require careful hyperparameter tuning [16]. Bouteiller et al. propose the Delay

Correcting Actor Critic (DCAC) as an extension to the Soft Actor Critic algorithm, by first predicting the system state from delayed observations, but then performing the actor-critic loop on the actual system state [6]. Wu et al. propose Auxiliary-Delayed Reinforcement Learning (AD-RL), in which they train an auxiliary short-delay value function to bootstrap the critic network, to sidestep the initial variance that direct training on tasks with long delays creates [8]. Karamzade et al. propose an extension to a world model (Delayed-Dreamer) that keeps an action buffer that allows the latent world state to evolve with the buffer during the delay of observations [7]. Hu et al. train a transformer network with random masking of observations (DeFog) to teach it to deal with dropped packets during testing [17]. Transformer models have also emerged as an alternative memory architecture for partially observable control tasks. Esslinger et al. replace the recurrent state of value-based agents with self-attention over the observation history (DTQN), reporting faster and more stable learning on POMDP benchmarks [18]. Wu et al. train a transformer model that forecasts the undelayed state from a delayed observation and pending-action buffer, avoiding the compounding errors of recursive state predictions under random delays [19].

While these previous works propose different strategies for dealing with delays and lossy communication on the architectural side of the agents, they have not made a systematic analysis of which additional signals about the network state can benefit the agents during training. They propose architectural interventions, which confound the main question of this paper: Which observation-state features carry actionable signals for a given network condition? Instead of following the architectural adaptations, we take one step back and analyze how much performance can be recovered with standard DRL algorithms, PPO and SAC, to isolate the information value of each augmentation feature.

3. Experimental Setup

3.1. Simulation and Training Environment

The training environment is implemented using MuJoCo [20], with the training environment setups taken from mujoco_playground [21]. For all training runs, the JAX-based warp backend was chosen, as it allows for easy GPU acceleration for the physics simulation, increasing the training throughput. The Brax [22] implementation of PPO and SAC is used, as it offers the simplest integration with mujoco_playground and compatibility with the JAX warp backend. For better resolution of the network delay, the physics simulation is run with a resolution of $\Delta t_p = 1$ ms, with multiple physics steps performed between multiple agent updates.

Network impairments are implemented as composable wrappers around the underlying mujoco_playground modes, thin Python classes that intercept the step and reset function calls, transforming the observations and actions that are transferred between an agent and the simulator. The wrappers are applied in order: delay -> loss -> metadata. The delay wrapper maintains a separate ring buffer for observations and actions that releases each value after the corresponding sampled latency value has elapsed. The packet-loss wrapper gates the observation path and substitutes a dropped packet with the most recent successfully delivered value. The metadata wrapper concatenates the selected feature group onto the observation vector sent to the agent. The following subsections describe the implementation of each wrapper in greater detail.

3.2. Network Delay and Loss Model

The following description focuses on the packets going through the observation path for greater clarity. The action path is implemented analogously. The control rate of each agent is chosen at 40 Hz. Each Δt_p , the state of the physics simulation is sampled, placed in

the delay wrapper, and the last valid action is applied to the simulation. The observation can either be delayed or lost. Formally, the agent receives an observation

$$\hat{s}_t = \begin{cases} s_{t-d_t^{obs}} & \text{if the observation is delivered at } t \\ \hat{s}_{t-1} & \text{if the observation was lost (zero-order hold)} \end{cases} \quad (1)$$

With $d_t^{obs} = \lceil L_t / \Delta t_p \rceil$ being derived from the sampled delay L_t^{obs} and quantized to the nearest physics step. Note that the observations are sampled at a higher rate than the control rate. This is consistent with most real-world setups, where sensors can generally supply higher sampling rates than the one the control system runs at. Observations that are supposed to be delivered between two control steps are discarded from the ring buffer afterwards.

On the action path, the agent's command traverses the same delay mechanism. An action latency is drawn independently at each physics step and quantized to the nearest physics step, and the actuator applies the command issued d_t^{act} physics steps earlier. The actuators continue to execute the previously issued command (zero-order hold at the actuator) until a new action is applied. At the episode's start, the buffer is initialized with zero commands. Because the latency is redrawn per physics step, command arrivals are not first-in-first-out: an action with larger latency can be overtaken by a newer action with a lower draw.

3.2.1. Gaussian Delay Model

At every control step, a new one-way latency value is drawn independently from a Gaussian distribution $L_t^{obs} \sim \mathcal{N}(\mu^{obs}, \sigma^{obs})$ truncated to $[0, L_{max}^{obs}]$. The action path analogously draws from its own distribution, characterized by μ^{act} , σ^{act} and L_{max}^{act} . The upper bound keeps the ring-buffer length bounded and prevents rare extreme outliers in the latency distribution. The continuous delay is then quantized to the integer step count d_t^{obs} . We use an independent per-step draw for the delays, as the simplest model to capture random delays encountered in wireless networks.

3.2.2. Gilbert-Elliott Packet-Loss Model

Loss probability is implemented using a two-state discrete-time Markov chain with states Good (G , packet delivered) and Bad (B , packet lost). At each step, the chain can transition with probabilities $G \rightarrow B = p_{gb}$ and $B \rightarrow G = p_{bg}$ with the self-loop probabilities following $G \rightarrow G = 1 - p_{gb}$ and $B \rightarrow B = 1 - p_{bg}$, respectively. This implements a simplified Gilbert-Elliott [23,24] loss model that creates the bursty, non-independent loss characteristic of wireless network channels.

3.2.3. Summary of All Parameters

Table 1 lists all parameters used for the impaired training runs. The same configuration is used during evaluation, except that the random seeds are refreshed for each evaluation episode.

The asymmetry in observation and action delay was chosen based on previous work and the increased instability with higher action delays. The parameters were chosen to mimic a typical industrial Wi-Fi deployment as closely as possible. Section 6 discusses the restriction on the selected operating points in more detail.

Table 1. Delay and loss parameters for the training runs.

Description	Symbol	Value
Mean observation delay	μ_{obs}	15 ms
Standard deviation observation delay	σ_{obs}	5 ms
Max observation delay	L_{max}^{obs}	40 ms
Mean action delay	μ_{act}	10 ms
Standard deviation action delay	σ_{act}	2.5 ms
Max action delay	L_{max}^{act}	30 ms
Loss transition probability $G \rightarrow B$	p_{gb}	2.5%
Loss transition probability $B \rightarrow G$	p_{bg}	40%
Derived steady-state Loss rate	$\frac{p_{gb}}{p_{gb} + p_{bg}}$	5.9%
Control rate	-	40 Hz
Inner physics dt	Δt_p	1 ms

3.2.4. Loss-Dominated and Location-Dependent Model

Additionally, we performed a second round of experiments with the same feature group setups, but replacing the Gaussian latency distribution and Gilbert-Elliott packet loss model with a pre-generated 64×64 spatial grid generated using a basic Wi-Fi ns-3 simulation [25]. The simulation (Figure 1) was set up using a single 802.11 access point at the origin, with the agent moving through a slice at distance $x, y \in [50, 60]$ m. At this distance, the communication channel is heavily loss-dominated, with per-step packet loss spanning 21–75% across the slice, while latency stays roughly at 17–19 ms. This inverts the previous channel’s delay-dominated regime and adds an additional dimension, where the Loss rate is heavily dependent on the location of the system. For the physical location, the CheetahRun moved across a slice of the environment, while we added a simulated movement across a circle for the FingerTurnHard task.

3.3. Tasks

We use a locomotion and a manipulation task from the mujoco_playground suite [21,26]:

- CheetahRun (17-dim observation, 6-dim action): A planar locomotion task whose reward incentivizes forward velocity. Stiff ground contacts and a high-frequency oscillatory gait make it sensitive to delayed feedback.
- FingerTurnHard (12-dim observation, 2-dim action): A manipulation task in which a planar two-link arm rotates a free spinner toward a target angle. The dynamics are inertia-dominated and forgiving of short delays.

3.4. Algorithms

We use PPO and SAC as the default DRL algorithms for on-policy and off-policy continuous control baselines, in their default Brax implementation and with their default Brax hyperparameters [22]. We used a per-task training budget of 40 M steps for PPO and 5 M steps for SAC on CheetahRun, and 20 M and 3 M on FingerTurnHard, respectively. The training budgets were determined using a separate screening run for both tasks, which trained them for a longer period and were chosen at roughly $1.5\times$ the wall-clock time to reach 80% of the asymptotic reward that the algorithms reach without any network impairments. We separately confirm that the training converges within the budget on the clean baseline, to rule out under-training as a confounding variable for the feature groups.

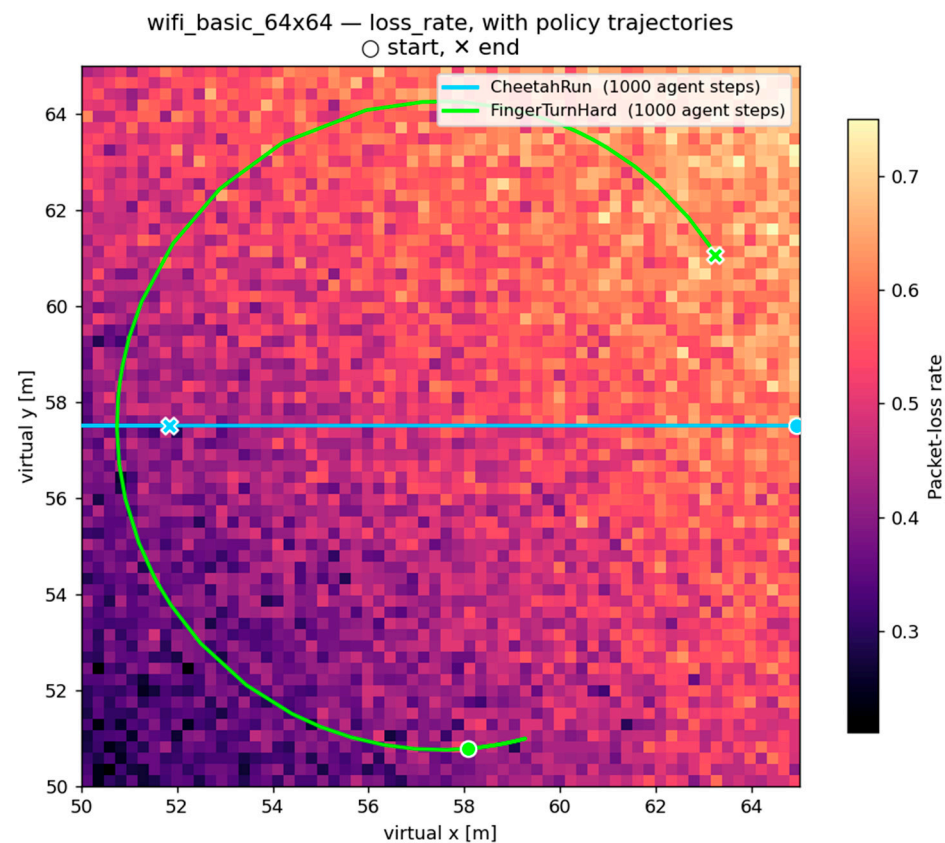


Figure 1. Location-dependent packet-loss map of the ns-3 wifi_basic scenario used as the second channel model, with the virtual trajectories (green—FingerTurnHard, blue—CheetahRun) of both tasks overlaid.

3.5. Feature Groups

We compare 8 feature groups denoted $F0$ – $F7$ for the ablation test. $F0$ is the no-metadata baseline. $F1$ – $F6$ isolate single features to research which aspect of the network state carries additional information, with $F7$ being the union of all six features. In the main experiment grid, statistical features were consistently calculated across a window W of the last 20 control steps, i.e., 0.5 s. This window size was chosen because it is the shortest window at which the 95th percentile latency feature in $F2$ is distinguishable from the window maximum, and is also close to the smallest window size with an expectation of a loss event ($1/5.9\% \approx 17$). Section 3.6 describes an additional dedicated ablation test that varies the window size. A dedicated tuning of the W is outside of the scope of this paper.

1. $F1$ Instantaneous (2 dimensions): Adds the most recent per-step latency and loss flag. This tests whether per-step network state is useful without any temporal aggregation to a controller lacking any temporal architecture adaptation.
2. $F2$ Latency statistics (5 dimensions): Adds mean, minimum, maximum, median, and 95th percentile of the latency inside W . This tests whether the distribution of recent latency, rather than just the instantaneous value, allows the controller to adapt to a specific network condition.
3. $F3$ Jitter (2 dimensions): The standard deviation of delay values and the mean absolute difference between consecutive delay samples within W . $F3$ isolates the variability of recent latency, since previous work has shown that DRL can adapt to constant delays.
4. $F4$ Loss rate (2 dimensions): The empirical Loss rate and the length of the longest loss burst inside W , as a loss analogue to $F2/F3$. This tests whether aggregate loss

information helps the agent beyond the information whether the current step was dropped, which is covered by $F1$.

5. $F5$ Time since last observation (1 dimension): Age of the last fresh observation. This should allow the agent to recognize a longer burst of lost packets.
6. $F6$ Observation age flag (2 dimensions): A binary one-hot indicator of whether the current observation is fresh or held over from a previous state.
7. $F7$ Union (13 dimensions): A deduplicated concatenation of feature groups $F1$ – $F6$. If $F7$ improves performance more than any other feature group, it indicates that there is complementary information inside each feature group, while a tie or worse performance indicates that a single feature contains enough information already, while the others either carry no signal or noisy signals that hinder the agent.

All feature groups are computed directly from the delay wrapper ring buffer and require no additional infrastructure. Each feature is either a per-step counter ($F1$, $F5$, $F6$), a single pass of elementary sum/min/max/comparison operations over W window ($F3$, $F4$, and the mean of $F2$), or, at worst, a sort of the values for the median and 95% percentile. The complete $F7$ union therefore requires on the order of a hundred elementary arithmetic and comparison operations per control step, which is several orders of magnitude below a single forward pass of even the small policy networks. They were also chosen in such a way that any real networked device can easily compute them based on information that is readily available to it, if it uses any communication protocol that has a sequence number indicator or similar. Additionally, we trained the algorithms using the same wrapper architecture but with 0 latency and loss. This provides us with a sanity check as well as an upper bound for both tasks.

3.6. Window-Size Ablation

To ensure that the results of the main set of experiments are not just an artifact of the chosen window size W , we repeat the training runs of the five feature groups $F2$, $F3$, $F4$, $F5$, and $F7$ at $W \in \{5, 10, 20, 50\}$ control steps (i.e., $\{0.125, 0.25, 0.5, 1.25\}$ s). $F0$, $F1$ and $F6$ are computed from just the current step and are therefore invariant to the window size. The window size ablation was performed using the CheetahRun task on the parametric channel with both PPO and SAC and is otherwise identical to the main experiment runs.

3.7. Training Protocol

Each condition was trained at 2048 parallel environments for 8 seeds, with 128 evaluation episodes at the end of training to produce a per-seed mean reward. The training ran on a single NVIDIA RTX 4080 under driver version 595.71.05, with CUDA 13.2 and Python 3.12.

4. Results

First, we establish the headroom that can be recovered by adding the features, by comparing the clean ceiling with the $F0$ baseline.

Table 2 shows the mean reward value across both environments, algorithms, and conditions. The reward values use the default reward function implemented by the Brax library, since reward shaping was not in the scope of this paper. One can immediately see that FingerTurnHard using PPO is a degenerate task, since even without any additional features, PPO solves this task to within 1% of the clean ceiling. There is therefore no room for any of the F -groups to improve the performance. We will therefore focus on the other configurations.

Table 2. Recoverable headroom across the two environments and algorithms. Reported values are the means over all seeds and evaluation episodes.

Environment	Algorithm	Clean	F0	Gap	F0/Clean
CheetahRun	PPO	841.5	670.4	171.1	0.8.0
CheetahRun	SAC	961.0	446.0	515.0	0.46
FingerTurnHard	PPO	969.3	958.9	10.4	0.99
FingerTurnHard	SAC	974.8	564.6	410.2	0.58

4.1. Window Size Sensitivity

Figure 2 shows the evaluation reward against the different window sizes W for the five feature groups on CheetahRun as described in Section 3.7. A table with all results was added to Appendix A. Latency and Jitter statistics provide no benefit at any tested window. Across all configurations $F2$ and $F3$, no CI excludes zero in the positive direction. $F4$ loss rate is dependent on the window size. For PPO, it shows a monotone decline, from a benefit at a short window size to a significant degradation with longer window sizes in comparison to $F0$. For SAC, no window sizes provide a benefit over $F0$, and the longer window sizes also perform significantly worse in comparison to the shorter ones. However, there is no monotone decline. The other feature groups do not show a significant dependency on W .

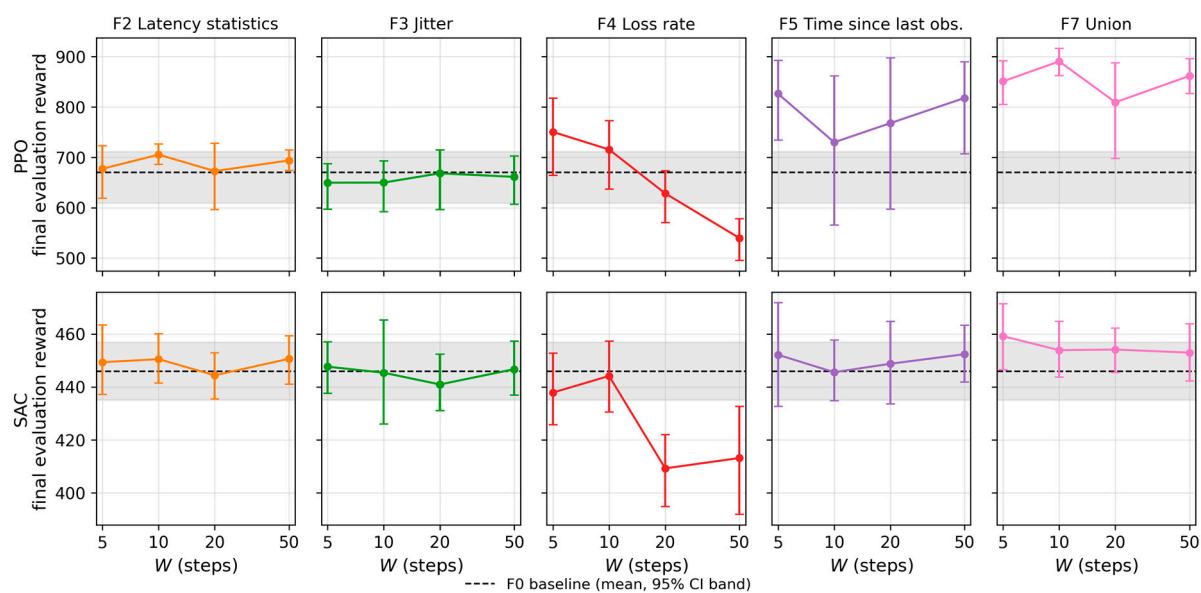


Figure 2. Evaluation reward vs. metadata window size W (control steps; 25 ms each) for the window-derived feature groups $F2$ – $F5$ and $F7$ on CheetahRun, parametric channel (top row: PPO, bottom row: SAC). Points are means over 8 seeds; whiskers are 95% bootstrap CIs.

The main experiment grid therefore retains the chosen $W = 20$ motivated in Section 3.5. The window ablation shows that the conclusions drawn at that window size are non-representative. The results of $F4$ are further discussed in Section 5.

4.2. Per F-Group Findings

Figure 3 plots the mean final rewards per F -group with 95% confidence intervals over all seeds, with the clean ceiling drawn in a dashed horizontal line. Figure 4 plots the median and interquartile range learning curves for all F -groups. Per cell breakdowns

with all results are placed in Appendix A; the rest of this chapter walks through the main qualitative results.

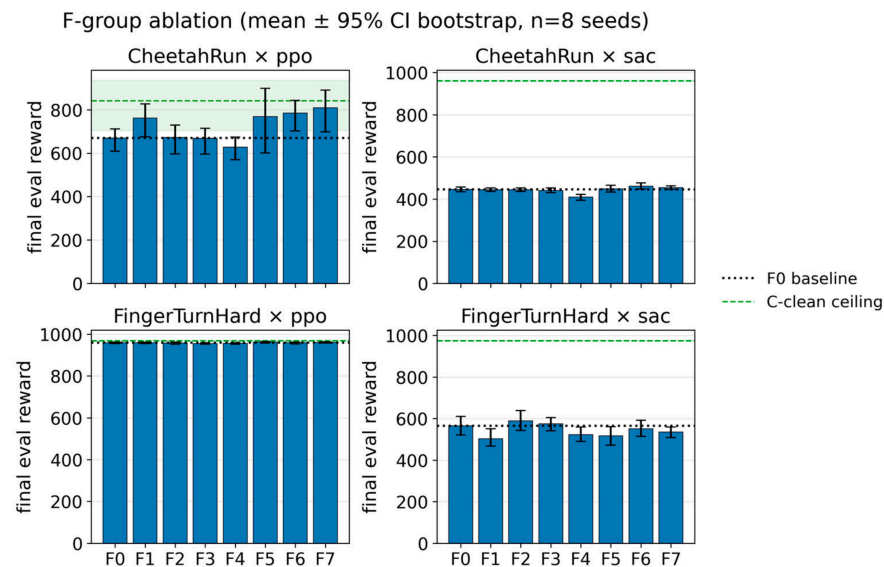


Figure 3. Final evaluation reward per feature group F0–F7 across the 2×2 (env $\times$ algo) grid. Bars are means over 8 seeds; whiskers are 95% bootstrap CIs; the dashed horizontal band marks the C-clean ceiling for each cell.

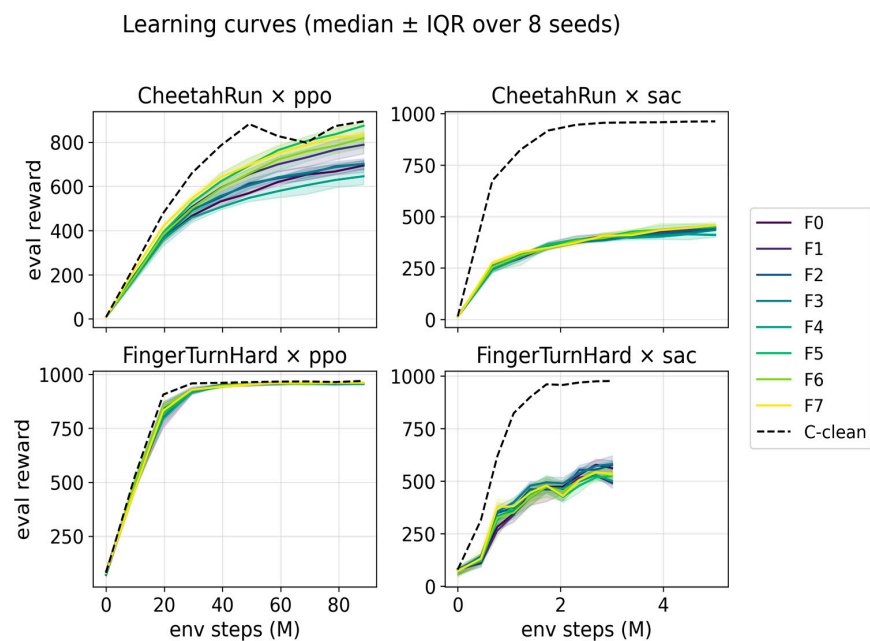


Figure 4. Learning curves per feature group F0–F7 across the 2×2 (env $\times$ algo) grid. The solid line represents the per-seed median, while the shaded band is the interquartile range across all seeds; the black dashed curve is the C-clean reference.

The clearest differences can be seen in the CheetahRun $\times$ PPO configurations. F1 Instantaneous, F5 Time since last observation, and F6 Observation age flag allow the algorithm to regain most of its performance, while F7 Union achieves the highest observed mean, recovering 81% of the gap between the F0 baseline and the clean result. Among the singleton feature groups, F6 has the highest mean, 784.7, with F5 767.9 and F1 762.2 close behind. F5 has the widest CI, driven by a single seed that collapsed to below 32% performance of the clean ceiling. This is not seen across any other run, and we therefore attribute it to random chance instability in the training process and do not interpret it

further. Window statistics features $F2$ and $F3$ stay mostly flat, while $F4$ Loss rate is even significantly negative, suggesting that the Loss rate acts as a noise channel rather than a useful context. The CheetahRun $\times$ SAC configuration had the biggest headroom across all configurations, but no F -group was reliably able to recover the performance beyond a few reward points. This is, however, not a noise artifact, as the standard deviation of the results across all seeds for both the clean run $\sigma = 4.6$ and all the F -groups $\sigma_{max} = 17.0$ is small in comparison to the final value, suggesting that SAC does not extract any recoverable signal from the observation metadata at this budget on this task. $F4$ Loss rate also hurts the performance of the SAC algorithm. As previously mentioned, FingerTurnHard $\times$ PPO shows no interesting behavior, since the algorithms can achieve full clean performance. However, FingerTurnHard $\times$ SAC has a few conflicting results. $F1$, $F4$, $F5$, $F6$ and $F7$ show lower means in comparison to $F0$, while $F2$ and $F3$ are nominally positive.

4.3. Position-Dependent Results

As described in Section 3.2.4, we performed the same feature group test across a second set of tests with a more loss-dominated channel, with additional location dependency of the network impairments. Figures 5 and 6 mirror Figures 2 and 3, respectively. In comparison to the parametric channel, the range of results between the feature groups is wider: Δ parametric channel = 181 vs. Δ ns-3 = 439 for CheetahRun $\times$ PPO and 52 to 207 for CheetahRun $\times$ SAC respectively.

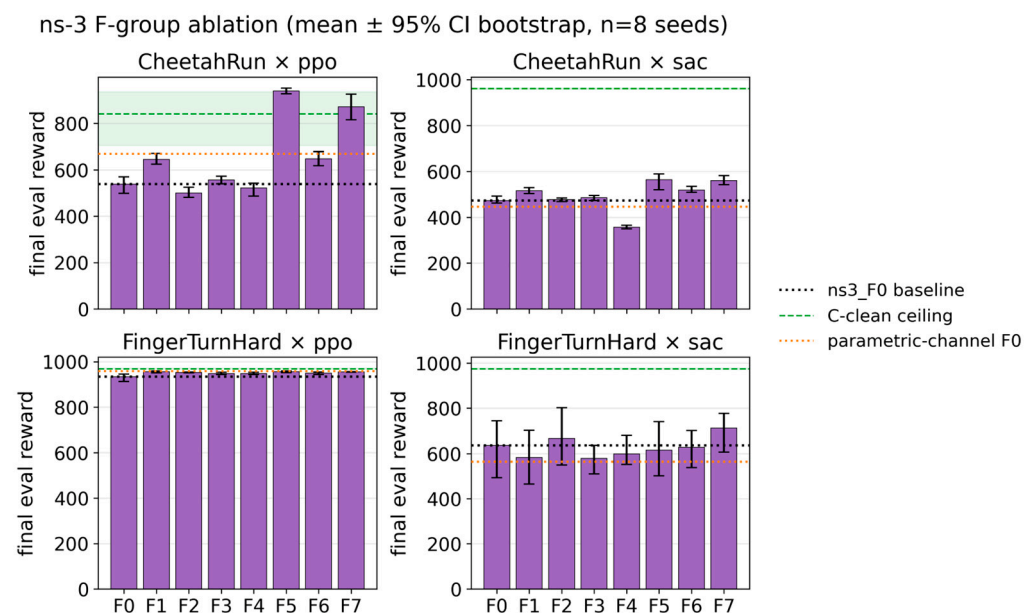


Figure 5. Final evaluation reward per feature group F0–F7 on the ns-3-based simulation. Bars are means over 4 seeds; whiskers are 95% bootstrap CIs; dashed horizontal bands mark the C-clean ceiling and the parametric-channel F0.

The choice of metadata feature, therefore, matters more on the loss-dominated location-dependent channel, suggesting that a purely parametric simulation of network impairments is insufficient to predict which feature a specific deployed network configuration favors. Furthermore, for PPO, the baseline $F0$ and most feature groups score lower than on the parametric channel, although the two strongest feature groups $F5$ (mean 939.7 vs. 767.9) and $F7$ (mean 870.8 vs. 809.4) improve on CheetahRun. SAC results stayed within a similar range as before, which suggests a different sensitivity of the algorithms to loss and delay specifically.

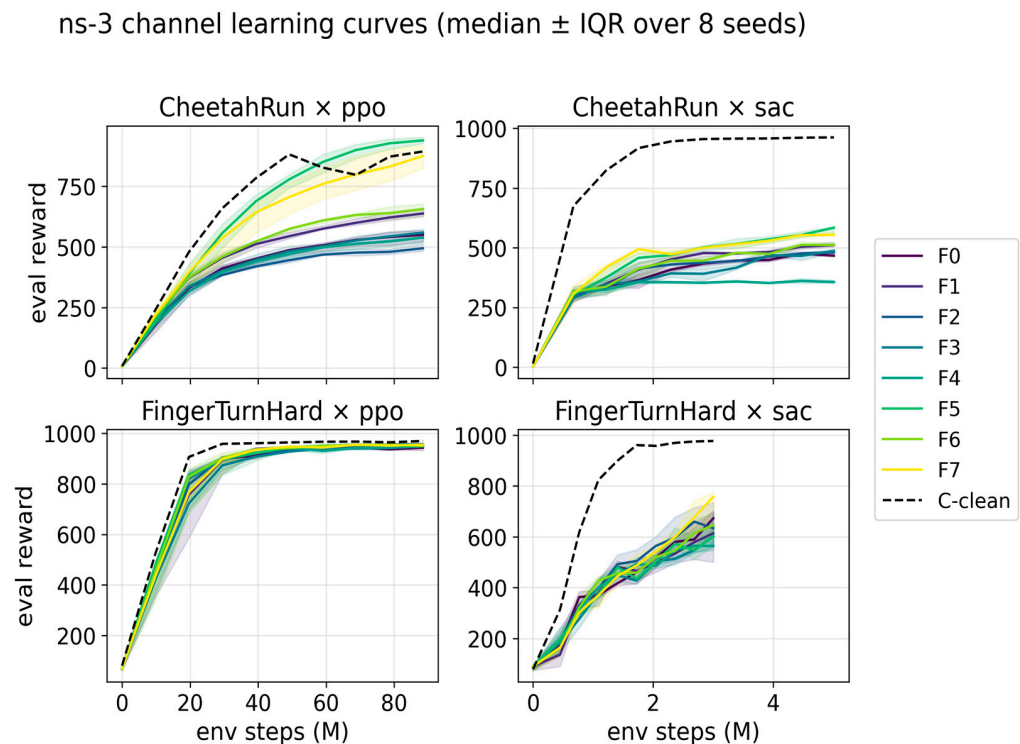


Figure 6. Learning curves per feature group F0–F7 on the ns-3 channel, across the 2×2 (env $\times$ algo) grid. Solid line is the per-seed median, shaded band is the IQR over 4 seeds; the gray curve is the C-clean reference.

F5 Time since last observation is now the best-performing feature across both CheetahRun configurations, with PPO reaching the level of the clean run. The effect is consistent across seeds rather than driven by an outlier. We note, however, that the clean mean is lowered by one weaker result, and excluding that outlier, the *F5* results (927–953) lie at the level of the best clean results (938–963). We theorize that under the high packet-loss rate, the explicit age signal lets the agent effectively lower its control rate, acting on a coarser time grid, which appears beneficial in this loss-dominated scenario. Since hyperparameter optimization was outside the scope of this paper, further investigation on this is left to future work. *F7* Union also shows consistently high results, except for the surprising result above, where *F5* clearly beats it. *F4* Loss rate also shows a consistently negative effect across these configurations, while multiple other features show a degradation only in this case.

5. Discussion

Analyzing the results, we put forward the following five findings:

1. Information on the freshness of the current observation carries the most recoverable signal for PPO algorithms in delay-dominated tasks. *F1*, *F5* and *F6*, i.e., the freshness features, provide the highest singleton uplift in the CheetahRun $\times$ PPO configuration in the parametric impairment configuration, second only to the union *F7*.
2. Sliding window latency and jitter summaries provide no benefit at any tested window size. No tested configuration significantly benefited from these metadata features. The window ablation (Section 4.1) shows no significant difference across the tested window sizes.
3. Loss-rate summaries actively destabilize training at long window sizes. *F4* actively hurt training with the primarily tested window size $W = 20$. The window size ablation, however, shows that at shorter window sizes, the Loss-rate feature can become positive for PPO or at least neutral for SAC.

4. The union of all feature matches is generally a safe addition. While the union of all features does not significantly outperform the best singleton feature group, it also does not add additional noise to the training. Its benefit is also robust across the full tested window range (Section 4.1), with the main experiment run $W = 20$ window showing the smallest point estimate of the four tested windows. $F7$ does not harm either algorithm at any other tested window size.
5. SAC is largely insensitive to the observation metadata. In the tested configurations, SAC generally did not benefit as much as PPO from the addition of the metadata features.

The consistency of $F4$'s negative effect across the non-degenerate configurations at $W = 20$ is the most striking negative finding. We offer two non-exclusive interpretations. First, the loss-rate window summary only describes the recent past of the channel but gives the agent no additional information about the current freshness of the packet. The longest burst length is also unactionable because it tells the agent nothing about whether the current step is inside one of the bursts. Second, the loss-rate features vary only slowly relative to the agent's update rate, i.e., they mostly act as an additional scalar input into the agent and therefore do not provide much actionable information as the state of the system changes. These interpretations are strengthened by the window size tests. With shorter window lengths $F4$ becomes closer to a freshness indicator, while longer lengths limit the per-step information the feature can carry. The monotone degradation between these two regimes for PPO is consistent with this reading. A causal verification, e.g., by performing a feature-attribution analysis, was outside the scope of this research.

The PPO vs. SAC difference is internally consistent across both environments and therefore is probably not an artifact of the training. Both algorithms converge well within the budget on the clean baseline; see Section 3.4. We propose a conjecture for the mechanics of this algorithm-level split based on our results, but do not verify it directly. PPO's on-policy update is computed from metadata and the immediate returns that an action produced, i.e., the gradient step directly attributes credit to each choice. SAC instead trains the critic algorithm using a replay buffer in which the metadata is sampled once during the training, as well as the bootstrap target, which uses independent draws of metadata. Therefore, each physics state in the buffer can be paired with many different samples of metadata, and the small per-step contribution of the metadata could tend toward zero in the mini-batches. In this view, the critic effectively ignores the added metadata, and the actor has no incentive to use the channels. Verifying this hypothesis requires additional analyses beyond the reward comparisons performed in this paper, and we leave it to future work. One possible experiment would be gradient-based feature attribution on the trained critics, comparing the sensitivity of the SAC and PPO value estimates to the metadata dimensions against their sensitivity to the physical state.

6. Conclusions

We performed a systematic ablation of observation-side metadata about the network state for reinforcement learning control over impaired network channels. The main results of the paper show that features related to the freshness of the current packet carry most of the actionable information for PPO algorithms, sliding-window summaries of the latency and jitter provide no significant benefit at any tested window size, and a summary of the Loss rate actively harms the training at longer aggregation windows. The union of all features generally is a safe default, but is not a strict winner over the best-performing singleton in each scenario. SAC is largely insensitive to the observation metadata. These results, taken together, argue against a universal "network-metadata vector" and in favor of a per-task and per-algorithm minimal singleton feature.

These findings are preliminary since this paper has several deliberate scope restrictions. Both network channel configurations were tested only at a single operating point, i.e., latency, jitter, and loss distributions were kept consistent across the environment. All conclusions are therefore established with one latency- and loss-dominated condition, and their generalization across that spectrum is untested. In particular, the null results for $F2$ latency and $F3$ jitter statistics are established under channels whose latency process is comparatively benign and stationary; whether these features remain uninformative when the latency distribution is heavier-tailed, non-stationary, or when the dominant impairment cannot be answered from our data. An impairment-parameter sweep that varies mean latency, jitter, and steady-state Loss rate independently is required before any feature can be recommended or ruled out. Our window-size ablation did not show a significant dependence on the window size, but the test itself is limited to a single task and the parametric channel. Several runs also had bigger confidence intervals, suggesting that there was more instability in the training. We also excluded any additional metadata about the network channel that transmits actions from the agents to the system. Finally, this study deliberately isolates the information content of the observation vector by holding the agent architecture fixed to standard feedforward PPO and SAC policies. We make no comparison against dedicated delay-aware methods discussed in Section 2. The effect sizes reported here therefore quantify what an unmodified policy can extract from each feature and do not establish how the resulting performance compares to that of architectural interventions. The results define the observation-side features for the follow-up comparison, in which delay-aware architectures are evaluated both with and without the beneficial features identified here, so that feature and architecture contributions can be separated. We read these results as evidence that the main question this paper poses, “Which observation-state features carry actionable signals for a given network condition?”, demands a substantially broader experimental treatment. Three directions are particularly important:

- Network impairments with positional and temporal structure: The results show a difference between the parametric and ns-3-based channel configurations. Therefore, there needs to be more analysis with richer mobility patterns, multi-AP geometries, and time-varying conditions.
- A wider network-feature taxonomy: The eight feature groups studied here cover instantaneous window-statistics and freshness signals, but we did not test predicted state features, e.g., a learned channel forecast or explicit action-history encodings. Since we show that which feature is best shifts across configurations, the feature library itself becomes an open design problem, not merely an implementation detail. The window-size ablation also suggests that for this particular setup, lower window lengths are more beneficial than higher ones, consistent with the claim that freshness features are more beneficial. Future work should examine whether this is task- or algorithm-dependent, and analyze whether this is an artifact of the simulation’s time steps.
- Architecture and training-strategy coupling: We tested only PPO and SAC in this paper. SAC’s insensitivity is plausibly due to an interaction of the features with the algorithm, rather than a property of the features themselves, but this requires further analysis. Furthermore, more advanced architectures, e.g., recurrent PPO/SAC with an LSTM or GRU encoder, or a transformer-based architecture with a learned memory system, may outperform the hand-crafted window statistics.

The main conclusion of this paper is therefore that, within our tested setups, the choice of observation-side metadata feature is an important design decision.

Author Contributions: Conceptualization, methodology, software, validation, investigation, writing, visualization, A.G.; project administration, R.H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the German Federal Ministry of Research, Technology and Space (BMFTR) in the course of the 6GEM+ Transfer Hub under grant 16KIS2410.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The per-configuration breakdown referenced in Section 4 is reproduced here in full. Each row reports the mean final reward over all seeds and evaluation episodes, the 95% bootstrap confidence interval (CI) on that mean, and the paired-bootstrap effect size against F0 with its 95% CI.

Appendix A.1. CheetahRun $\times$ PPO Parametric Channel

Table A1. Per-configuration breakdown of the window-size ablation test.

Feature Group	W Steps	Mean	95% CI	Δ F0	Δ 95% CI
Clean	—	841.5	[702, 933]	—	—
F0 None	—	670.4	[609, 711]	—	—
F2 Latency statistics	5	681.2	[625, 725]	+10.8	[−65, +95]
F2 Latency statistics	10	708.4	[690, 728]	+38.0	[−11, +109]
F2 Latency statistics	20	672.7	[597, 728]	+2.3	[−21, +29]
F2 Latency statistics	50	694.4	[675, 714]	+24.0	[−18, +97]
F3 Jitter	5	650.6	[598, 689]	−19.7	[−29, −9]
F3 Jitter	10	652.3	[594, 697]	−18.0	[−43, +1]
F3 Jitter	20	668.2	[595, 715]	−2.1	[−21, +16]
F3 Jitter	50	661.5	[606, 703]	−8.9	[−31, +8]
F4 Loss rate	5	750.4	[663, 818]	+80.1	[+44, +119]
F4 Loss rate	10	715.5	[637, 773]	+45.2	[+23, +67]
F4 Loss rate	20	628.3	[569, 673]	−42.0	[−56, −30]
F4 Loss rate	50	541.5	[496, 580]	−128.9	[−177, −85]
F5 Time since last observation	5	826.4	[733, 893]	+156.1	[+120, +187]
F5 Time since last observation	10	732.3	[565, 867]	+62.0	[−110, +178]
F5 Time since last observation	20	767.9	[600, 897]	+97.5	[−59, +197]
F5 Time since last observation	50	818.0	[706, 890]	+147.6	[+93, +193]
F7 Union	5	851.1	[805, 892]	+180.7	[+145, +220]
F7 Union	10	890.7	[862, 916]	+220.3	[+172, +280]

Appendix A.2. CheetahRun $\times$ PPO Parametric Channel

Table A2. Per-configuration breakdown for CheetahRun $\times$ PPO on the parametric channel.

Feature Group	Mean	95% CI	Δ F0	Δ 95% CI
Clean	841.5	[704, 936]	—	—
F0 None	670.4	[609, 712]	—	—
F1 Instantaneous	762.2	[675, 827]	+91.8	[+61, +121]
F2 Latency statistics	672.7	[597, 729]	+2.3	[−21, +29]
F3 Jitter	668.2	[596, 715]	−2.1	[−21, +16]
F4 Loss rate	628.3	[570, 674]	−42.0	[−56, −30]
F5 Time since last observation	767.9	[601, 898]	+97.5	[−59, +198]
F6 Observation age flag	784.7	[703, 843]	+114.3	[+91, +137]
F7 Union	809.4	[698, 890]	+139.1	[+84, +186]

*Appendix A.3. CheetahRun × SAC Parametric Channel***Table A3.** Per-configuration breakdown for CheetahRun × SAC on the parametric channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	961.0	[958, 963]	—	—
F0 None	446.0	[435, 457]	—	—
F1 Instantaneous	444.6	[437, 453]	−1.4	[−11, +8]
F2 Latency statistics	444.4	[436, 453]	−1.6	[−16, +14]
F3 Jitter	440.9	[431, 453]	−5.0	[−20, +9]
F4 Loss rate	409.2	[395, 422]	−36.8	[−47, −27]
F5 Time since last observation	448.7	[433, 465]	+2.8	[−19, +24]
F6 Observation age flag	461.1	[448, 476]	+15.1	[−3, +32]
F7 Union	454.1	[446, 462]	+8.1	[−3, +20]

*Appendix A.4. FingerTurnHard × PPO Parametric Channel***Table A4.** Per-configuration breakdown for FingerTurnHard × PPO on the parametric channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	969.3	[967, 972]	—	—
F0 None	958.9	[955, 962]	—	—
F1 Instantaneous	958.7	[955, 962]	−0.2	[−4, +2]
F2 Latency statistics	957.6	[953, 962]	−1.3	[−3, +1]
F3 Jitter	955.5	[952, 959]	−3.3	[−7, +2]
F4 Loss rate	956.0	[952, 959]	−2.8	[−7, +1]
F5 Time since last observation	961.6	[958, 965]	+2.7	[+0.5, +5.1]
F6 Observation age flag	959.2	[954, 963]	+0.3	[−2, +3]
F7 Union	960.4	[958, 963]	+1.5	[−1, +4]

*Appendix A.5. FingerTurnHard × SAC Parametric Channel***Table A5.** Per-configuration breakdown for FingerTurnHard × SAC on the parametric channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	974.8	[972, 978]	—	—
F0 None	564.6	[522, 611]	—	—
F1 Instantaneous	503.0	[467, 551]	−61.6	[−136, +11]
F2 Latency statistics	589.8	[542, 639]	+25.2	[−8, +58]
F3 Jitter	575.2	[542, 605]	+10.6	[−31, +48]
F4 Loss rate	522.4	[490, 560]	−42.2	[−69, −20]
F5 Time since last observation	517.3	[472, 561]	−47.2	[−117, +20]
F6 Observation age flag	550.9	[515, 592]	−13.7	[−39, +13]
F7 Union	535.8	[509, 560]	−28.8	[−69, +10]

*Appendix A.6. CheetahRun × PPO ns-3 Channel***Table A6.** Per-configuration breakdown for CheetahRun × PPO on the ns-3 channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	841.5	[704, 936]	—	—
F0 None	538.9	[499, 571]	—	—
F1 Instantaneous	645.7	[624, 671]	+106.8	[+81, +133]
F2 Latency statistics	501.0	[481, 525]	−37.9	[−90, +14]
F3 Jitter	556.6	[540, 573]	+17.7	[−24, +54]
F4 Loss rate	522.0	[487, 542]	−16.9	[−32, −2]
F5 Time since last observation	939.7	[927, 952]	+400.8	[+379, +434]
F6 Observation age flag	648.2	[618, 678]	+109.3	[+61, +167]
F7 Union	870.6	[815, 926]	+331.7	[+264, +403]

Appendix A.7. CheetahRun $\times$ SAC ns-3 Channel**Table A7.** Per-configuration breakdown for CheetahRun $\times$ SAC on the ns-3 channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	961.0	[958, 963]	—	—
F0 None	473.3	[462, 492]	—	—
F1 Instantaneous	515.8	[504, 530]	+42.5	[+16, +68]
F2 Latency statistics	477.3	[468, 484]	+3.9	[−13, +17]
F3 Jitter	484.8	[472, 496]	+11.4	[−5, +27]
F4 Loss rate	357.1	[349, 365]	−116.2	[−140, −98]
F5 Time since last observation	564.4	[520, 590]	+91.1	[+51, +124]
F6 Observation age flag	518.8	[509, 536]	+45.5	[+19, +70]
F7 Union	559.6	[542, 582]	+86.3	[+81, +90]

Appendix A.8. FingerTurnHard $\times$ PPO ns-3 Channel**Table A8.** Per-configuration breakdown for FingerTurnHard $\times$ PPO on the ns-3 channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	969.3	[967, 972]	—	—
F0 None	934.0	[913, 945]	—	—
F1 Instantaneous	955.7	[950, 960]	+21.7	[+12, +37]
F2 Latency statistics	952.3	[950, 955]	+18.4	[+7, +39]
F3 Jitter	947.8	[943, 953]	+13.9	[+2, +31]
F4 Loss rate	947.8	[943, 953]	+13.9	[−1, +38]
F5 Time since last observation	956.0	[953, 959]	+22.0	[+11, +41]
F6 Observation age flag	949.0	[944, 954]	+15.1	[+0.4, +40]
F7 Union	955.1	[954, 957]	+21.1	[+8, +44]

Appendix A.9. FingerTurnHard $\times$ SAC ns-3 Channel**Table A9.** Per-configuration breakdown for FingerTurnHard $\times$ SAC on the ns-3 channel.

Feature Group	Mean	95% CI	$\Delta F0$	Δ 95% CI
Clean	974.8	[972, 978]	—	—
F0 None	636.8	[493, 744]	—	—
F1 Instantaneous	583.3	[464, 702]	−53.4	[−107, −5]
F2 Latency statistics	665.4	[549, 803]	+28.7	[−121, +153]
F3 Jitter	578.3	[509, 636]	−58.5	[−118, +2]
F4 Loss rate	598.4	[552, 680]	−38.3	[−164, +87]
F5 Time since last observation	615.1	[501, 740]	−21.7	[−69, +26]
F6 Observation age flag	627.8	[537, 701]	−8.9	[−44, +36]
F7 Union	712.4	[606, 777]	+75.6	[+6, +117]

References

1. Chinchali, S.; Sharma, A.; Harrison, J.; Elhafsi, A.; Kang, D.; Pergament, E.; Cidon, E.; Katti, S.; Pavone, M. Network offloading policies for cloud robotics: A learning-based approach. *Auton. Robot.* **2021**, *45*, 997–1012. [\[CrossRef\]](#)
2. Pezzutto, M.; Dey, S.; Garone, E.; Gatsis, K.; Johansson, K.H.; Schenato, L. Wireless control: Retrospective and open vistas. *Annu. Rev. Control* **2024**, *58*, 100972. [\[CrossRef\]](#)
3. Schenato, L.; Sinopoli, B.; Franceschetti, M.; Poolla, K.; Sastry, S.S. Foundations of Control and Estimation Over Lossy Networks. *Proc. IEEE* **2007**, *95*, 163–187. [\[CrossRef\]](#)
4. Bernardo, G.T.T.; Miranda, G., Jr.; Macedo, D.F. Analysis of Network Performance over Deep Reinforcement Learning Control Loops for Industry 4.0. In Proceedings of the Anais Do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2022), Fortaleza, Brazil, 23–27 May 2022; pp. 1–14.

5. Park, P.; Coleri Ergen, S.; Fischione, C.; Lu, C.; Johansson, K.H. Wireless Network Design for Control Systems: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 978–1013. [\[CrossRef\]](#)
6. Bouteiller, Y.; Ramstedt, S.; Beltrame, G.; Pal, C.; Binas, J. Reinforcement Learning with Random Delays. In Proceedings of the International Conference on Learning Representations, Virtual, 3–7 May 2021. Available online: <https://openreview.net/forum?id=QFYnKIBJYR> (accessed on 8 July 2026).
7. Karamzade, A.; Kim, K.; Lanier, J.B.; Corsi, D.; Fox, R. Model-Based Reinforcement Learning under Random Observation Delays. *arXiv* **2025**, arXiv:2509.20869.
8. Wu, Q.; Zhan, S.S.; Wang, Y.; Wang, Y.; Lin, C.W.; Lv, C.; Zhu, Q.; Schmidhuber, J.; Huang, C. Boosting reinforcement learning with strongly delayed feedback through auxiliary short delays. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024.
9. Sandha, S.S.; Garcia, L.; Balaji, B.; Anwar, F.; Srivastava, M. Sim2Real Transfer for Deep Reinforcement Learning with Stochastic State Transition Delays. In Proceedings of the 2020 Conference on Robot Learning, Virtual, 16–18 November 2020; pp. 1066–1083. Available online: <https://proceedings.mlr.press/v155/sandha21a.html> (accessed on 8 July 2026).
10. Sutton, R.S.; Barto, A. *Reinforcement Learning: An Introduction*; The MIT Press: Cambridge, MA, USA; London, UK, 2018.
11. Katsikopoulos, K.V.; Engelbrecht, S.E. Markov decision processes with delays and asynchronous cost collection. *IEEE Trans. Autom. Control* **2003**, *48*, 568–574. [\[CrossRef\]](#)
12. Altman, E.; Nain, P. Closed-loop control with delayed information. In Proceedings of the 1992 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, Newport, RI, USA, 1–5 June 1992; pp. 193–204.
13. Walsh, T.J.; Nouri, A.; Li, L.; Littman, M.L. Learning and planning in environments with delayed feedback. *Auton. Agent Multi-Agent Syst.* **2009**, *18*, 83–105. [\[CrossRef\]](#)
14. Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. Planning and acting in partially observable stochastic domains. *Artif. Intell.* **1998**, *101*, 99–134. [\[CrossRef\]](#)
15. Hausknecht, M.; Stone, P. Deep Recurrent Q-Learning for Partially Observable MDPs. *AAAI Fall Symp.* **2015**, *45*, 141.
16. Ni, T.; Eysenbach, B.; Levine, S.; Salakhutdinov, R. Recurrent Model-Free RL Is a Strong Baseline for Many POMDPs. Available online: <https://openreview.net/forum?id=E0zOKxQsZhN> (accessed on 8 July 2026).
17. Hu, K.; Zheng, R.C.; Gao, Y.; Xu, H. Decision Transformer under Random Frame Dropping. *arXiv* **2023**, arXiv:2303.03391.
18. Esslinger, K.; Platt, R.; Amato, C. Deep Transformer Q-Networks for Partially Observable Reinforcement Learning. 2022. Available online: <https://arxiv.org/pdf/2206.01078> (accessed on 8 July 2026).
19. Wu, Q.; Wang, Y.; Zhan, S.S.; Wang, Y.; Lin, C.W.; Lv, C.; Zhu, Q.; Schmidhuber, J.; Huang, C. Directly Forecasting Belief for Reinforcement Learning with Delays. In Proceedings of the Forty-Second International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025. Available online: <https://openreview.net/forum?id=S9unJQditt> (accessed on 8 July 2026).
20. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 5026–5033.
21. Zakka, K.; Tabanpour, B.; Liao, Q.; Haiderbhai, M.; Holt, S.; Luo, J.Y.; Allshire, A.; Frey, E.; Sreenath, K.; Kahrs, L.A.; et al. MuJoCo Playground: An Open-Source Framework for GPU-Accelerated Robot Learning and Sim-to-Real Transfer. GitHub. Available online: https://github.com/google-deepmind/mujoco_playground (accessed on 8 July 2026).
22. Freeman, C.D.; Frey, E.; Raichuk, A.; Girgin, S.; Mordatch, I.; Bachem, O. Brax—A Differentiable Physics Engine for Large Scale Rigid Body Simulation. Available online: <http://github.com/google/brax> (accessed on 8 July 2026).
23. Gilbert, E.N. Capacity of a Burst-Noise Channel. *Bell Syst. Tech. J.* **1960**, *39*, 1253–1265. [\[CrossRef\]](#)
24. Elliott, E.O. Estimates of Error Rates for Codes on Burst-Noise Channels. *Bell Syst. Tech. J.* **1963**, *42*, 1977–1997. [\[CrossRef\]](#)
25. Riley, G.F.; Henderson, T.R. The Ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*; Wehrle, K., Güneş, M., Gross, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 15–34.
26. Tunyasuvunakool, S.; Muldal, A.; Doron, Y.; Liu, S.; Bohez, S.; Merel, J.; Erez, T.; Lillicrap, T.; Heess, N.; Tassa, Y. dm_control: Software and tasks for continuous control. *Softw. Impacts* **2020**, *6*, 100022. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.