# Deep Reinforcement Learning Without Experience Replay, Target Networks, or Batch Updates

**Mohamed Elsayed**[†⋆] **Gautham Vasan**[†⋆] **A. Rupam Mahmood**[†‡⋆]

[†]University of Alberta [⋆]Alberta Machine Intelligence Institute [‡] CIFAR Canada AI Chair
{mohamedelsayed,vasan,armahmood}@ualberta.ca

## Abstract

Natural intelligence processes experience as a continuous stream, sensing, acting, and learning moment-by-moment in real time. Streaming learning, the modus operandi of classic reinforcement learning (RL) algorithms like Q-learning and TD, mimics natural learning by using the most recent sample without storing it. This approach is also ideal for resource-constrained, communication-limited, and privacy-sensitive applications. However, in deep RL, learners almost always use batch updates and replay buffers, making them computationally expensive and incompatible with streaming learning. Although the prevalence of batch deep RL is often attributed to its sample efficiency, a more critical reason for the absence of streaming deep RL is its frequent instability and failure to learn, which we refer to as *stream barrier*. This paper introduces the *stream-x* algorithms, the first class of deep RL algorithms to overcome stream barrier for both prediction and control and match sample efficiency of batch RL. Through experiments in Mujoco Gym, DM Control, and Atari Games, we demonstrate stream barrier in existing algorithms and successful stable learning with our stream-x algorithms: stream Q, stream AC, and stream TD, achieving the best model-free performance in DM Control Dog environments. A set of common techniques underlies the stream-x algorithms, enabling their success with a single set of hyperparameters and allowing for easy extension to other algorithms, thereby reviving streaming RL.

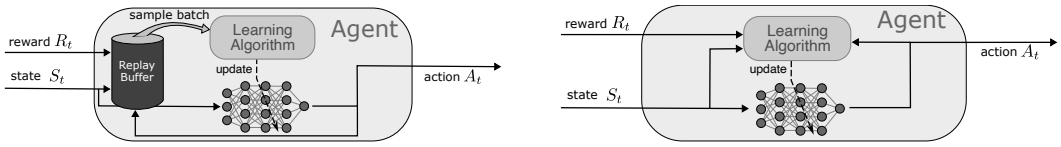
## 1 Introduction

Learning from a continuous stream of experience as it arrives is a paramount challenge, mirroring natural learning (Hayes et al. 2021), and is relevant to many applications involving on-device learning (Hayes & Kanan 2022, Neuman et al. 2022, Verma et al. 2023). For instance, learning from recent experience can help systems adapt quickly to changes (e.g., wear and tear) compared to learning from potentially obsolete data. In streaming reinforcement learning, such as Q-learning or temporal difference (TD) learning, the agent receives an observation and reward at each step, taking action and making a learning update immediately without storing the sample. This scenario is practical since retaining raw samples is often infeasible due to limited computational resources (Hayes & Kanan 2022), lack of communication access, or concerns about data privacy (Van de Ven et al. 2020).

While classic RL algorithms like Q-learning, SARSA, Actor-Critic, and TD were originally developed for streaming learning (see Sutton & Barto 2018), recent advancements have shifted focus primarily toward batch learning. Indeed, advancements in recent deep RL rely heavily on computationally extensive batch learning as demonstrated in many domains, such as games (e.g., Mnih et al. 2015, Silver et al. 2017), simulated environments (e.g., Haarnoja et al. 2018) and various robotics tasks (e.g., Smith et al. 2023, Haarnoja et al. 2024). Batch RL algorithms store past samples in a storage called replay buffer and draw samples from it in a batch to make updates. Figure 1 highlights the difference between agents in the problem settings of streaming RL and batch RL[1]. Unlike batch RL, streaming RL does not permit the use of a replay buffer or batch updates.

[1]We use the word batch to refer to methods that use batch updates, not to be confused with offline RL.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

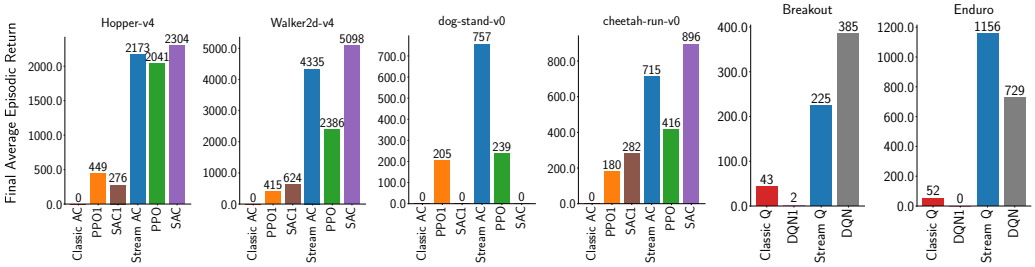



(a) Batch RL (b) Streaming RL

Figure 1: Agents in streaming RL and batch RL problem settings. Streaming RL requires updates from immediate individual samples without storing past samples, whereas batch RL relies on batch updates from past samples stored in a replay buffer.

The success of batch RL is often attributed to its efficiency with data and modern hardware, as argued by Riedmiller (2005), Mnih et al. (2015, 2016), and Lillicrap et al. (2016), among many others. Averaging samples in a batch may enable more reliable updates, and reusing samples multiple times may potentially extract more information from the same sample. Moreover, batch updates allow efficient use of parallel environments and modern hardware accelerators like GPUs. However, the prohibitive computational requirements of batch learning methods render them unsuitable for on-board learning in resource-constrained systems, such as edge devices or Mars rovers (Wang et al. 2023), or when rapid decision-making is necessary (e.g., latency arbitrage). For example, storing high-dimensional images for replay demands substantial memory, and batch updates slow down real-time prediction and decision-making (see Yuan & Mahmood 2022). When computation is constrained or samples cannot be stored, streaming learning becomes essential. And yet, streaming learning remains largely unadopted in deep RL, and currently, there is a noticeable absence of streaming RL applications in practice, making deep RL under resource constraints unachieved.

Deep streaming RL is understood to be inherently sample inefficient since samples cannot be reused (ct. D'Oro et al. 2023, Schwarzer et al. 2023). Another reason for sample inefficiency is that credit assignment is typically propagated slowly by one-step methods, which bootstrap fully, compared to their multi-step counterparts, which use rewards from multiple steps (Sutton & Barto 2018). Although methods using multi-step returns have better credit assignment, they cannot make updates at immediate time step or without storing (Mahmood 2017). Eligibility traces (Sutton 1988, van Hasselt et al. 2014, Mahmood & Sutton 2015, van Seijen et al. 2016, White & White 2016, Thodoroff et al. 2019, van Hasselt et al. 2021) attempt to balance the benefits of using multi-step returns with updating at every time step. However, they are rarely used in deep RL.




(a) MuJoCo Gym (b) DM Control (c) Atari

Figure 2: Stream barrier. Both classic streaming methods (e.g., Classic Q) and streaming versions of batch RL methods (e.g., PPO1) perform poorly due to stream barrier. In contrast, our stream-x algorithms (e.g., stream Q) overcome stream barrier and perform competitively with their batch RL counterparts, demonstrating its stability and robustness. The performance is shown as zero if some of the runs for an algorithm diverged.

Although the absence of streaming deep RL is attributed to its sample inefficiency, a more critical reason is that existing deep learning methods experience learning instabilities and even failures in the streaming learning setting (see Elfwing et al. 2018), which we refer to as *stream barrier* (see Figure 2). Deep RL methods already struggle with online updates, facing issues such as loss of plasticity (Lyle et al. 2023, Dohare et al. 2024), poor learning dynamics (Lyle et al. 2024), failure to achieve further improvement (e.g., Lyle et al. 2023, 2022), and gradual performance degradation (e.g., Dohare et al. 2023, Abbas et al. 2023, Elsayed & Mahmood 2024). In addition, streaming

deep RL presents unique challenges since the observation and reward distributions used for updating change rapidly over time, exacerbating the issues. The lack of application of eligibility traces with neural networks can also be attributed to the issues of instability (see Anand & Precup 2021, Harb & Precup 2017), which can even lead to divergence (Veeriah et al. 2017). As a result, deep RL methods face stream barrier and have largely been overlooked. However, a few studies (Elfwing et al. 2018, Young & Tian 2019) have shown nascent performance with streaming learning, suggesting that this area holds potential for further exploration and development.

In this paper, we address stream barrier by introducing streaming deep RL methods—stream TD($\lambda$), stream Q($\lambda$), and stream AC($\lambda$)—that are collectively called the *stream-x* algorithms and utilize eligibility traces. Our approach enables learning from the most recent experiences without using replay buffers, batch updates, or target networks. Contrary to the common belief, we demonstrate that streaming deep RL can be stable and as sample efficient as batch RL. The effectiveness of our approach hinges on a set of key techniques that are common to all stream-x algorithms. They include a novel optimizer to adjust step size for stability, appropriate data scaling, a new initialization scheme, and maintaining a standard normal distribution of pre-activations. Our approach requires no hyperparameter tuning, and the results with different algorithms on the electricity consumption prediction task (Zhou et al. 2021), MuJoCo Gym (Todorov et al. 2012), DM Control Suite (Tunyasuvunakool et al. 2020), MinAtar (Young & Tian 2019), and Atari 2600 (Bellemare et al. 2013) environments are achieved using the same set of hyperparameters. The results demonstrate our approach's ability to work as an off-the-shelf solution, overcome stream barrier, provide results previously unattainable with streaming methods, and even surpass the performance of batch RL, achieving the best model-free performance on some complex environments.

## 2 Background

The interaction between the *agent* and the *environment* is modeled as a Markov decision process (MDP). We consider in this paper episodic interactions, the episodic MDP of which is given by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, d_0, \mathcal{H})$, where $\mathcal{S}$ is the set of *states*, $\mathcal{A}$ is the set of *actions*, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$ is the transition dynamics model in which $\Delta(\mathcal{X})$ is a distribution over the set $\mathcal{X}$, $\mathcal{R}$ denotes the set of reward signals, $d_0$ is the distribution of starting states, $\gamma \in [0, 1]$ is the discount factor, and $\mathcal{H}$ is the set of terminal states. The agent interacts with the environment according to a *policy* $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that gives a distribution over actions conditioned on the state. The interaction in each episode starts when the environment samples a state from the starting state distribution: $S_0 \sim d_0$. At each time step $t$, the agent receives a state $S_t$ from the environment, takes an action $A_t \sim \pi(\cdot|S_t)$, and the environment samples the next state and reward using the transition model: $S_{t+1}, R_{t+1} \sim \mathcal{P}(\cdot, \cdot|S_t, A_t)$. The agent keeps interacting with the environment until it reaches one of the terminal states $S_T \sim \mathcal{H}$, where $T$ is the termination time step. The episodic *return* is defined as the sum of discounted rewards starting from time step $t$: $G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$. The goal of the agent in the *prediction* problem is to estimate, for a given policy $\pi$, the *value* function $v_\pi \doteq \mathbb{E}_\pi [G_t|S_t = s], \forall s \in \mathcal{S}$ with an estimator $\hat{v}(s, \boldsymbol{w})$ or the *action-value* function $q_\pi(s, a) \doteq \mathbb{E}_\pi [G_t|S_t = s, A_t = a], \forall s \in \mathcal{S}, a \in \mathcal{A}$ with an estimator $\hat{q}(s, a, \boldsymbol{w})$, where $\boldsymbol{w}$ is a parameter vector. The goal of the agent in the *control* problem is to find the optimal policy $\pi_*$ using action-value estimates such that $q_{\pi_*}(s, a) = \max_\pi q_\pi(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ or to optimize the objective $J(\boldsymbol{\theta}) \doteq \mathbb{E}_{S_0 \sim d_0} [v_{\pi_{\boldsymbol{\theta}}}(S_0)]$ wrt $\boldsymbol{\theta}$ that parameterizes the policy $\pi_{\boldsymbol{\theta}}$.

**Temporal Difference Learning.** To estimate the value function for prediction or learn the optimal policy for control, we can use a Monte Carlo estimate based on the return $G_t$, which requires waiting until the episode is terminated, resulting in the update rule $\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha(G_t - \hat{v}(S_t, \boldsymbol{w}_t))\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t)$. *Temporal difference* (TD) learning (Sutton 1988) alleviates this issue by relying on the idea of *bootstrapping*. In TD learning, the return $G_t$ is replaced by the bootstrapped target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$ called one-step return, resulting in the TD error: $\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$. The TD error can be used to update a value estimate as soon as the next state and reward are observed.

**Policy Gradient Theorem.** When the agent is learning a parameterized policy to maximize the objective $J(\boldsymbol{\theta})$, model-free gradient updates can be used according to the policy gradient theorem (Sutton et al. 1999): $\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_{S \sim d_\pi^\gamma, A \sim \pi} [q_\pi(S, A) \nabla_{\boldsymbol{\theta}} \log \pi(A|S, \boldsymbol{\theta})]$, where $d_\pi^\gamma$ is the discounted stationary-state distribution. In practice, the action-value function $q_\pi$ is replaced by an, often biased, estimate, and states are sampled on-policy or from a replay buffer without considering discounting and including further bias (Thomas 2014, Nota & Thomas 2019, Zhang et al. 2022, Che et al. 2023).

The estimator $\delta_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}) \approx \nabla J(\boldsymbol{\theta})$ is used in one-step actor-critic (AC), which learns both a policy or an *actor* and a value function or a *critic* (see Barto et al. 1983).

**Eligibility Traces.** Eligibility traces are short-term memory vectors that can be used to form *multi-step* methods in a streaming form, achieving better credit assignment than their one-step counterparts. The idea of eligibility traces (Sutton & Barto 1981) is influenced by the biological neuroscience model by Klopf (1972). The eligibility trace vector is initialized to zero at the start of the episode; then, it accumulates the value gradient faded by $\gamma\lambda$, where $\lambda \in [0, 1]$ is the eligibility trace parameter. Specifically, given the eligibility trace vector $\boldsymbol{z}_t$, its update rule is given by $\boldsymbol{z}_t \doteq \gamma\lambda \boldsymbol{z}_{t-1} + \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w})$, where $\boldsymbol{z}_{-1} \doteq \mathbf{0}$. The idea of eligibility traces is powerful and can be combined with almost all temporal difference methods (Sutton & Barto 2018) by replacing the value or policy gradients with their trace counterparts. For example, the TD(0) algorithm estimates a value function by using the update rule: $\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha\delta_t \nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t)$, which can be replaced by $\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha\delta_t \boldsymbol{z}_t$ for the TD($\lambda$) algorithm that uses eligibility traces.

Although the eligibility trace looks like a momentum term at first glance, they have distinct functionality. In SGD with momentum (Polyak 1964), a trace of gradients is maintained: $\boldsymbol{z}_{t+1} \doteq \beta \boldsymbol{z}_t + \nabla_{\boldsymbol{w}} \mathcal{L}$ and $\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t - \alpha \boldsymbol{z}_t$, where $\mathcal{L}$ is the loss, $\alpha$ is the step size, and $\beta$ is some decay factor. Thus, the momentum term is a trace of the past gradients of the loss. There are two mechanistic differences between the eligibility trace and the momentum term: 1) in eligibility traces, we maintain a trace of past gradients of the function output itself, whereas we maintain a trace of the past gradients of the loss in the momentum term, and 2) the momentum term is never reset to zero, whereas eligibility traces are reset after the end of each episode since there is no meaningful credit assignment across different episodes. And unlike the momentum term, updates with eligibility traces are equivalent to those of multi-step returns (e.g., $\lambda$-return), achieving fixed points superior to one-step updates (Mahmood 2017, Sutton & Barto 2018). Lastly, eligibility traces have been found to be effective primarily in tabular settings or with linear function approximation, while none of their deep-learning counterparts are known to perform well (Veeriah et al. 2017).

**Neural Networks.** Learning representations from data is one of the crucial tasks that allow agents to work on arbitrary problems without relying on domain knowledge (e.g., via hand-crafted representations). Neural networks are a natural choice for learning those representations since we can use them in a data-driven approach. Typically, neural networks are structured as a composition of non-linear functions where the output of each function is the input of the next, and so on, to learn a hierarchical structure of features. When the number of functions composing a neural network becomes large, the neural network is often referred to as a deep neural network. For simplicity, we focus here on fully connected neural networks. Consider a neural network $f$ parametrized by the set of weights $\mathcal{W} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_L\}$, where $W_{l,i,j}$ is the entry in the $i$-th row and the $j$-th column of the matrix weight matrix at the $l$-th layer. In the forward pass, we get the *post-activation* vector $\boldsymbol{h}_l$ by applying the activation function $\sigma$ to the *pre-activation* vector $\boldsymbol{a}_{l-1}$: $\boldsymbol{h}_l \doteq \sigma(\boldsymbol{a}_{l-1})$. We simplify the notation by starting post-activation vector $\boldsymbol{h}_1$ with the neural network input $\boldsymbol{x}$: $\boldsymbol{h}_1 \doteq \boldsymbol{x}$. We obtain the pre-activation vector $\boldsymbol{a}_l$ by applying a matrix-vector multiplication between the weight matrix $\boldsymbol{W}_l$ and the post-activation vector $\boldsymbol{h}_l$: $\boldsymbol{a}_l \doteq \boldsymbol{W}_l \boldsymbol{h}_l$. The bias terms can be included if we appended the matrix by an additional column and appended 1 to each post-activation vector.

## 3 Method

In this section, we introduce our method and describe the necessary components for successful streaming reinforcement learning agents. The agents, under the streaming reinforcement learning problem, are required to process one sample at a time without storing any samples for future reuse.[2] Such requirements create additional hurdles compared to batch deep reinforcement learning, even though both learn from a non-stationary stream of data. We list the issues that hinder learning as 1) learning instability due to occasional large updates, 2) learning instability due to activation nonstationarity, and 3) improper scaling of data. These issues are already present in batch methods causing several detrimental effects such as drop in performance (Dohare et al. 2023, Abbas et al. 2023), high variance (Bjorck et al. 2021), or inability to improve performance (Lyle et al. 2023).

[2]Streaming learning methods mainly require CPUs instead of GPUs since no batch updates are used, unless a very large neural network is used in which they might benefit from GPUs. In such case, the overhead of context switching between CPU and GPU might be negligible compared to the GPU computational cost required for the forward and backward passes in very large networks.

However, they are exacerbated with streaming learning as updates can fluctuate more from one step to another due to non-i.i.d. sampling for updates. For example, streaming learning is more prone to instability as successive per-sample gradients can point in different directions, making it difficult to choose a single working step size. In contrast, batch methods mitigate this issue by averaging gradients from an i.i.d.-sampled batch drawn from a large pool. Moreover, we use additional techniques for sample efficiency, which we describe first.

### 3.1 Sample efficiency with sparse initialization and eligibility traces

Since steaming learning methods must discard the sample once used, they can potentially be sample inefficient. Here, we present two techniques to improve sample efficiency of streaming learning methods: 1) sparse initialization and 2) eligibility traces.

Sparse representations induce locality when updating the network, which reduces the amount of interference between dissimilar inputs. Many works have shown that sparsity reduces forgetting, which helps improve sample efficiency in reinforcement learning (Liu et al. 2019, Pan et al. 2021, Sokar et al. 2022, Lan & Mahmood 2023). For example, tile coding (Albus 1971) has been shown to reduce forgetting in RL (see Ghiassian et al. 2020). We use a simple technique to introduce sparsity at initialization by randomly initializing most weights to zeros. Specifically, we impose a sparsity level $s$ (e.g., $0.9$) at each layer representing the proportion of zero-initialized weights. The remaining weights are initialized according to the LeCun initialization scheme (LeCun et al. 2002). Although sparsity-based initialization has not been investigated for reinforcement learning before, it has been shown to improve optimization in supervised learning (Martens 2010). Algorithm 1 shows our proposed sparse initialization technique—*SparseInit*. This sparse initialization scheme can be used for both fully-connected and convolutional layers.

**Algorithm 1** SparseInit

**Require:** network $f$, sparsity level $s$
**for** weight $\boldsymbol{W}$ and bias $\boldsymbol{b}$ **do**
  $n \leftarrow s \times \texttt{fan_in}$
  Permutation set $\mathcal{P}$ of size $\texttt{fan_in}$
  Index set $\mathcal{I}$ of size $n$ (subset of $\mathcal{P}$)
  $W_{i,j} \sim U\left[-1/\sqrt{\texttt{fan_in}}, 1/\sqrt{\texttt{fan_in}}\right], \forall i, j$
  $W_{i,j} \leftarrow 0, \forall i \in \mathcal{I}, \forall j$
  $b_i \leftarrow 0, \forall i$
**Return:** initialized network $f$

Credit assignment is a fundamental challenge in learning from interaction. Eligibility traces (Sutton 1988) provide a compact approach for better credit assignment than one-step methods. In this paper, we use accumulating traces for both value functions and policies. Given a value function $\hat{v}$ parameterized by the weight vector $\boldsymbol{w}_t$, its eligibility trace vector $\boldsymbol{z}_t$ is defined as: $\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \nabla\hat{v}_{\boldsymbol{w}}(S_t, \boldsymbol{w}_t)$, where $\gamma$ is the discount factor and $\lambda$ is the eligibility trace parameter. Given a policy $\pi$ parameterized by the weight vector $\boldsymbol{\theta}$, its eligibility trace is defined as: $\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t, \boldsymbol{\theta}_t)$. We refer the reader to Appendix E to show how we can incorporate entropy regularization with eligibility traces. Note that since we accumulate values in the trace vector, the traces can be arbitrarily large, potentially causing divergence (Veeriah et al. 2017). Thus, a careful update rule must be used to prevent eligibility traces from causing instability.

### 3.2 Adjusting step sizes for maintaining update stability

Instability in deep reinforcement learning is an issue that persisted for a long time (Bjorck et al. 2021). Recently, many works (e.g., Asadi et al. 2023, Lyle et al. 2023, Dohare et al. 2023) have identified the Adam optimizer (Kingma & Ba 2015) as one of the main sources of instability. In this section, we aim to develop a stable optimizer that is more suitable for streaming reinforcement learning.

In optimization, a well-known strategy for avoiding large updates and choosing an appropriate step size is the backtracking line search method (Armijo 1966), which for each iteration typically chooses the step size that maximizes the expected or batch-based objective. Likewise, backtracking line search has been shown to be effective in stabilizing on-policy batch reinforcement learning (e.g., TRPO, Schulman et al. 2015). In the streaming case, it is not clear if choosing a step size that reduces the error in the current sample is the best strategy. A more pertinent goal in streaming learning is to de-emphasize an update if it is too large, for example, if the update *overshoots* the target on a single sample (Mahmood 2010, Mahmood et al. 2012). More specifically, given a scalar error $\delta(S)$ on a

sample (e.g., say an input-output pair), an update overshoots if the post-update error on the same sample $\delta_+(S)$ changes its sign, that is, $\delta(S)\delta_+(S) < 0$. A change in the error sign indicates that the error has been over-corrected or the update has overshot the target. Kearney (2023) defined a related quantity, the *effective step size*, that measures the amount of progress the learner achieved based on the update, given as follows:

$$\xi \doteq \frac{\delta(S) - \delta_+(S)}{\delta(S)}, \tag{1}$$

where $\xi > 1$ indicates overshooting or over-correction, $\xi < 1$ indicates partial correction, and $\xi = 0$ indicates no correction. The effective step size quantity can be used to control the amount of error correction, for example, well before overshooting occurs. We can compute the effective step size with a counterfactual update using some starting step size $\alpha = \alpha_{\text{Init}} \in (0, 1]$. If the effective step size is larger than the maximum effective step size, $\xi > \xi_{\text{max}}, \xi_{\text{max}} \in (0, 1]$, then we reduce the step size by a factor of $\beta$, $\alpha = \beta\alpha, \beta \in (0, 1)$. This backtracking line search continues until the condition $\xi \leq \xi_{\text{max}}$ is met. We call this process *bounding effective step size with backtracking* and provide its details in Algorithm 2. This overshooting prevention strategy was originally explored by Mahmood et al. (2012) to improve the stability of meta-gradient based supervised learning. The idea was then applied in reinforcement learning as well (see Dabney & Barto 2012, Kearney 2023, Javed et al. 2024). In both settings, only linear function approximation was previously considered.

**Algorithm 2** Bounding Effective Step Size with Backtracking

**Initialize**: maximum effective step size $\xi_{\text{max}}$ (e.g., $0.05$)
**Require**: eligibility trace $\boldsymbol{z_w}$, weight vector $\boldsymbol{w}$, error function $\delta$, and starting step size $\alpha$
$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \alpha\delta_{\boldsymbol{w}}\boldsymbol{z_w}$ ▷ Counterfactual weights
**while** $\frac{\delta - \delta}{\delta} > \xi_{\text{max}}$ **do** ▷ Exact effective step size detection
  $\alpha \leftarrow \beta\alpha$ ▷ Backtracking line search until the condition is met
  $\boldsymbol{w}' \leftarrow \boldsymbol{w} + \alpha\delta_{\boldsymbol{w}}\boldsymbol{z_w}$ ▷ Note that $\boldsymbol{z_w} = \nabla_{\boldsymbol{w}} f$ for supervised learning
**return** $\boldsymbol{w}'$

The drawback of such backtracking is that they require multiple iterations per time step, each iteration requiring an additional forward pass for each new $\delta_+$. This can be expensive when many forward passes are required until we find a step size that satisfies the criteria. A less expensive alternative without additional forward passes would be much more desirable.

To avoid expensive computation of backtracking, we develop an approximate mechanism for effective step size control. For simplicity of analysis, we concatenate all weights' entries of the network into a single vector $\mathbf{w}$. Using first-order Taylor approximation and assuming local linearity, we write the learner's post-update prediction on the same sample as

$$\begin{aligned} f(\boldsymbol{x}; \mathbf{w}^+) &= f(\boldsymbol{x}; \mathbf{w} - \boldsymbol{u}(\boldsymbol{x}; \boldsymbol{w})) \\ &= f(\boldsymbol{x}; \mathbf{w}) - \nabla_{\mathbf{w}} f(\boldsymbol{x}; \mathbf{w})^\top \boldsymbol{u}(\boldsymbol{x}; \boldsymbol{w}); \qquad \text{under local linearity,} \end{aligned} \tag{2}$$

where $\mathbf{w}^+$ and $\mathbf{w}$ are the parameters vectors after and before making the update, respectively, $\boldsymbol{x}$ is the input, and $\boldsymbol{u}$ is the update vector. The local linearity assumption holds approximately when the updates are small, which is partly what we are aiming to achieve.

Using the effective step size given in Eq. 1, we provide the analysis to get the condition for effective step size control for TD($\lambda$) and refer the reader to Appendix C for supervised regression. For semi-gradient TD($\lambda$), the $\boldsymbol{u}$ update vector is $\alpha\delta\boldsymbol{z}$, where $\delta$ is the TD error and $\boldsymbol{z}$ is the eligibility trace vector. The effective step size of TD($\lambda$) under nonlinear function approximation is given by

$$\begin{aligned} \xi &= \frac{(r + \gamma v(\boldsymbol{w}; \boldsymbol{x}') - v(\boldsymbol{w}; \boldsymbol{x})) - (r + \gamma v(\boldsymbol{w}_+; \boldsymbol{x}') - v(\boldsymbol{w}_+; \boldsymbol{x}))}{\delta} \\ &= \frac{\alpha\delta\gamma\boldsymbol{z}^\top \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \alpha\delta\boldsymbol{z}^\top \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})}{\delta} \\ &= \alpha\boldsymbol{z}^\top \left(\gamma\nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})\right). \end{aligned}$$

Since we need to find the gradient of the value function at a different observation $\boldsymbol{x}'$, it would still require an additional backward pass. To remove the extra computation, we further approximate the

effective step size as follows:

$$
\begin{aligned}
\xi &= \alpha \boldsymbol{z}^\top \left(\gamma \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})\right) \\
&\leq \alpha |\boldsymbol{z}|^\top |\gamma \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})| \leq \alpha |\boldsymbol{z}|^\top \mathbf{1} \\
&= \alpha \|\boldsymbol{z}\|_1 \leq \kappa \alpha \|\boldsymbol{z}\|_1, \qquad \text{where } \kappa > 1 \\
&\leq \kappa \alpha \bar{\delta} \|\boldsymbol{z}\|_1, \qquad \text{where } \bar{\delta} = \max(|\delta|, 1).
\end{aligned} \tag{3}
$$

Here, $|\boldsymbol{z}|$ is a vector containing the modulus of each element of $\boldsymbol{z}$, and we assume that all entries of $|\gamma \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})|$ are less than or equal to 1. We motivate the assumption with an argument on Lipschitz's continuity. If the gradient of the value function is $k$-Lipschitz continuous, we can write $\|\nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})\|_1 \leq k \|\boldsymbol{x}' - \boldsymbol{x}\|_1$, when $\gamma = 1$. Thus, for $\gamma \approx 1$ (e.g., 0.99) and $k\|\boldsymbol{x}' - \boldsymbol{x}\|_1 < 1$ (e.g., nearby states), the inequality $|\gamma \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x}') - \nabla_{\boldsymbol{w}} v(\boldsymbol{w}; \boldsymbol{x})|_i \leq 1, \forall i$ holds. Our analysis emphasizes that, unlike in supervised regression, the bootstrapped target in temporal difference learning changes by changing the prediction output. This perspective aligns with the approach of Dabney & Barto (2012) and Kearney (2023) but contrasts with the approach of Javed et al. (2024), who used the same condition that Mahmood et al. (2012) used for supervised regression. In Algorithm 3, we show how we can build an optimizer that uses this condition on the effective step size to control the update size. While this algorithm is based on SGD, an adaptive version similar to the Adam optimizer is given in Appendix B.

**Algorithm 3** **O**vershooting-**b**ounded **G**radient **D**escent (ObGD)

**Require**: Eligibility trace $\boldsymbol{z_w}$, weight vector $\boldsymbol{w}$, error $\delta$, step size $\alpha$, scaling factor $\kappa$
$\bar{\delta} = \max(|\delta|, 1)$
$M \leftarrow \alpha \kappa \bar{\delta} \|\boldsymbol{z_w}\|_1$ ▷ Note that $\boldsymbol{z_w} = \nabla_{\boldsymbol{w}} f$ for supervised learning
$\alpha \leftarrow \min\left(\frac{\alpha}{M}, \alpha\right)$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \delta \boldsymbol{z_w}$
**return** $\boldsymbol{w}$

### 3.3 Stabilizing activation distribution under non-stationarity

The change in weight distribution across layers can cause trainability issues (Xu et al. 2019). Thus, many normalization techniques exist to normalize each layer's pre-activations and give them similar distributions, which has shown advantages in both stationary (Xu et al. 2019) and nonstationary settings (Lyle et al. 2023, Gallici et al. 2024) to maintain favorable learning dynamics. Nauman et al. (2024) have shown that using layer normalization is crucial for achieving good performance in challenging environments, even with deep RL methods (e.g., Haarnoja et al. 2018). LayerNorm (Ba et al. 2016) standardizes the pre-activations by subtracting their mean and dividing by their variance. Other approaches normalize by the $L_2$ norm of the pre-activation (L2Norm, Nguyen et al. 2017) or the root mean square of the pre-activation vector (RMSNorm, Zhang & Sennrich 2019).

In our approach, we use LayerNorm (Ba et al. 2016), which we apply to the pre-activation of each layer (before applying the activation $\sigma$) without learning any scaling or bias parameters. Specifically, the LayerNorm normalization $\phi$ we use is given by

$$
\phi(\boldsymbol{a}) \doteq \frac{\boldsymbol{a} - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad \text{where } \mu \doteq \frac{1}{n} \sum_{i=1}^{n} a_i \text{ and } \sigma^2 \doteq \frac{1}{n} \sum_{i=1}^{n} (a_i - \mu)^2, \tag{4}
$$

where $n$ is the dimensionality of $\boldsymbol{a}$ and $\epsilon$ is a small number used for numerical stability. We call a network that applies LayerNorm at each layer a *LayerNorm network*.

### 3.4 Proper scaling of data

Properly scaling the training data is essential for effective learning (Sutton 1988, Schraudolph 2002, LeCun et al. 2002). Training data are typically normalized in supervised learning since all data points are available beforehand. This assumption breaks in reinforcement learning where learning is done online based on interactions in environments with unbounded state spaces. Recently, Lyle et al. (2023) argued that large-scale targets can reduce trainability. Thus, data-scaling techniques are often used in deep RL (e.g., Schulman et al. 2017). Scaling the targets, for example, the rewards,

(Engstrom et al. 2020) or the TD errors (Schaul et al. 2021), and scaling the observations (e.g., normalization, Andrychowicz et al. 2020) are well-established strategies that have shown success and are incorporated into widely used algorithms such as PPO (Schulman et al. 2017) and A2C (Mnih et al. 2016), helping improve their performance and stability (Rao et al. 2020, Huang et al. 2022a). The problem of learning data scaling for reinforcement learning has been studied before, and mechanisms other than observation or reward normalization have been introduced. For example, van Hasselt et al. (2016) proposed a method that adaptively normalizes the targets used in the learning updates, allowing the agent to learn robustly across many orders of magnitude. However, estimating target scaling as part of the optimization process can be more involved, and therefore, simple normalization techniques might be desirable.

**Algorithm 4** SampleMeanVar (Welford 1962)

**Require**: Input $x$, mean $\mu$, estimate $p$, and counter $n$.
$n \leftarrow n + 1$
$\bar{\mu} \leftarrow \mu + \frac{1}{n}(x - \mu)$
$p \leftarrow p + (x - \mu)(x - \bar{\mu})$
$\sigma^2 \leftarrow \frac{p}{n-1}$ if $n \geq 2$, otherwise $\sigma^2 \leftarrow 1$
**return** $\bar{\mu}, \sigma^2, n$

In this work, we transform observation and reward to address the scaling issue using the algorithm by Welford (1962), which computes unbiased estimates of the mean and variance (also see Knuth 2014, pp. 232). Algorithm 5 shows how rewards are scaled, and Algorithm 6 shows how observations are normalized.

**Algorithm 5** ScaleReward

**Initialize**: $u \leftarrow 0$
**Require:** $r, \gamma, p, T, n$
$u \leftarrow \gamma(1 - T)u + r$
$_, p, \sigma^2, n \leftarrow$ SampleMeanVar$(u, 0, p, n)$
**Return:** $\frac{r}{\sqrt{\sigma + \epsilon}}, p$

**Algorithm 6** NormalizeObservation

**Require:** $S, \mu, p, n$
$\mu, \sigma^2, p, n \leftarrow$ SampleMeanVar$(S, \mu, p, n)$
**Return:** $\frac{S - \mu}{\sqrt{\sigma + \epsilon}}, \mu, p$

**Algorithm 7** Stream TD($\lambda$)

**Given** LayerNorm state-value network $\hat{v}(s; \boldsymbol{w})$ with vectorized weights vector $\boldsymbol{w}$ and initialized with SparseInit
**Initialize** discount factor $\gamma$ (e.g. 0.99) and eligibility traces parameter $\lambda$ (e.g. 0.9)
**Initialize** step size $\alpha$ (e.g., 1) and scaling factor $\kappa$ (e.g., 2)
**Initialize** $\mu_S, t$ to zero, and $p_S, p_r$ to one
**for** each episode **do**
  $\boldsymbol{z_w} \leftarrow \mathbf{0}$
  Initialize $S$ (first state of the episode)
  $S, \mu_S, p_S, \leftarrow$ NormalizeObservation$(S, \mu_S, p_S, t)$
  **for** each time step in the episode **do**
    $t \leftarrow t + 1$
    Observe $S'$, $R$, $T$ ▷ $T$ indicates whether $S'$ is a terminal state
    $S', \mu_S, p_S \leftarrow$ NormalizeObservation$(S', \mu_S, p_S, t)$
    $R, p_r \leftarrow$ ScaleReward$(R, \gamma, p_r, T, t)$
    $\delta \leftarrow R + \gamma\hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$ ▷ if $S'$ is terminal, then $\hat{v}(S', \boldsymbol{w}) \doteq 0$
    $\boldsymbol{z_w} \leftarrow \gamma\lambda\boldsymbol{z_w} + \nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow$ ObGD$(\boldsymbol{z_w}, \boldsymbol{w}, \delta, \alpha, \kappa)$
    $S \leftarrow S'$

### 3.5 Stable streaming deep reinforcement learning methods

Here, we combine the above techniques to provide novel streaming deep reinforcement learning algorithms, which we call *stream-x* algorithms. We use the following color scheme for better readability: purple for layer normalization, blue for observation normalization, orange for reward scaling, teal for step size scaling, and brown for sparse initialization. Algorithms 8, 9, 10, 7, show algorithms—stream AC($\lambda$) for actor-critic-based control, stream Q($\lambda$) for off-policy action-value-based control based on Watkin's Q($\lambda$), stream SARSA($\lambda$) for on-policy action-vallue-based control, and stream TD($\lambda$) for value prediction, respectively.

**Algorithm 8** Stream AC($\lambda$)

**Given** LayerNorm policy network $\pi(a|s;\boldsymbol{\theta})$ parametrizing a normal distribution with vectorized weights vector $\boldsymbol{\theta}$ and initialized with SparseInit
**Given** LayerNorm state-value network $\hat{v}(s;\boldsymbol{w})$ with vectorized weights vector $\boldsymbol{w}$ and initialized with SparseInit
**Initialize** discount factor $\gamma$ (e.g. 0.99) and eligibility traces parameter $\lambda$ (e.g. 0.9)
**Initialize** policy step size $\alpha_\pi$ (e.g., 1), value step size $\alpha_{\hat{v}}$ (e.g., 1), entropy coefficient $\tau$ (e.g. 0.01), policy scaling factor $\kappa_\pi$ (e.g., 3), and value scaling factor $\kappa_{\hat{v}}$ (e.g., 2)
**Initialize** $p_r, p_S$ to zero and $\mu_S, t$ to one
**for** each episode **do**
  $\boldsymbol{z_w} \leftarrow \mathbf{0}, \boldsymbol{z_\theta} \leftarrow \mathbf{0}$
  Initialize $S$ (first state of the episode)
  $S, \mu_S, p_S, \leftarrow$ NormalizeObservation$(S, \mu_S, p_S, t)$
  **for** each time step in the episode **do**
    $t \leftarrow t+1$
    $A \sim \pi(.|S, \boldsymbol{\theta})$
    Take action $A$, observe $S'$, $R$, $T$ ▷ $T$ indicates whether $S'$ is a terminal state
    $S', \mu_S, p_S \leftarrow$ NormalizeObservation$(S', \mu_S, p_S, t)$
    $R, p_r, \leftarrow$ ScaleReward$(R, \gamma, p_r, T, t)$
    $\delta \leftarrow R + \gamma\hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$ ▷ if $S'$ is terminal, then $\hat{v}(S', \boldsymbol{w}) \doteq 0$
    $\boldsymbol{z_w} \leftarrow \gamma\lambda\boldsymbol{z_w} + \nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w})$
    $\boldsymbol{z}_\theta \leftarrow \gamma\lambda\boldsymbol{z}_\theta + \nabla_{\boldsymbol{\theta}}\left(\log\pi(A|S, \boldsymbol{\theta}) + \tau\text{sign}(\delta)H(.|S, \boldsymbol{\theta})\right)$ ▷ $H(.|S, \boldsymbol{\theta})$ is the entropy
    $\boldsymbol{\theta} \leftarrow$ ObGD$(\boldsymbol{z_\theta}, \boldsymbol{\theta}, \delta, \alpha_\pi, \kappa_\pi)$
    $\boldsymbol{w} \leftarrow$ ObGD$(\boldsymbol{z_w}, \boldsymbol{w}, \delta, \alpha_{\hat{q}}, \kappa_{\hat{q}})$
    $S \leftarrow S'$

**Algorithm 9** Stream Q($\lambda$)

**Given** LayerNorm action-value network $\hat{q}(s, a; \boldsymbol{w})$ with vectorized weights vector $\boldsymbol{w}$ and initialized with SparseInit
**Initialize** discount factor $\gamma$ (e.g. 0.99) and eligibility traces parameter $\lambda$ (e.g. 0.9)
**Initialize** step size $\alpha$ (e.g., 1), and scaling factor $\kappa_{\hat{q}}$ (e.g., 2)
**Initialize** $p_r, p_S$ to zero and $\mu_S, t$ to one
**for** each episode **do**
  $\boldsymbol{z_w} \leftarrow \mathbf{0}$
  Initialize $S$ (first state of the episode)
  $S, \mu_S, p_S, \leftarrow$ NormalizeObservation$(S, \mu_S, p_S, t)$
  **for** each time step in the episode **do**
    $t \leftarrow t+1$
    Choose $A$ from $S$ using policy derived from $\hat{q}$ (e.g., $\epsilon$-greedy)
    **if** $A$ is non-greedy **then**
      $\boldsymbol{z_w} \leftarrow \mathbf{0}$
    Take action $A$, observe $S'$, $R$, $T$ ▷ $T$ indicates whether $S'$ is a terminal state
    $S', \mu_S, p_S \leftarrow$ NormalizeObservation$(S', \mu_S, p_S, t)$
    $R, p_r, \leftarrow$ ScaleReward$(R, \gamma, p_r, T, t)$
    $\delta \leftarrow R + \gamma\max_a \hat{q}(S', a, \boldsymbol{w}) - \hat{q}(S, A, \boldsymbol{w})$ ▷ if $S'$ is terminal, then $\hat{q}(S', ., \boldsymbol{w}) \doteq 0$
    $\boldsymbol{z_w} \leftarrow \gamma\lambda\boldsymbol{z_w} + \nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow$ ObGD$(\boldsymbol{z_w}, \boldsymbol{w}, \delta, \alpha_{\hat{q}}, \kappa_{\hat{q}})$
    $S \leftarrow S'$

## 4 Experiments

In this section, we demonstrate the effectiveness of our stream-x algorithms. We start by showing stream barrier in different challenging environments where classic methods fail, but our stream-x algorithms overcome this barrier and are competitive with other batch methods. We study stream AC($\lambda$) and compare it against classic AC($\lambda$), PPO, SAC, PPO1 (streaming version of PPO), and SAC1 (streaming version of SAC) in MuJoCo Gym and DM Control environments. Next, we study stream Q($\lambda$) and compare it against their classic versions in addition to DQN and DQN1 (streaming version of DQN) in MinAtar and the Atari 2600 arcade environments. We then demonstrate the importance of each component in our approach with a thorough ablation study. Finally, we study stream TD($\lambda$) and show its effectiveness in time series prediction with real-life data. We focus in this section on the key results here and give the full experimental details in Appendix F.

### 4.1 Overcoming stream barrier

Streaming deep RL methods often experience instability and failure to learn, which we refer to as stream barrier. Figure 2 shows stream barrier in three different challenging benchmarking tasks: MuJoCo Gym, DM Control, and Atari. The performance of each algorithm is averaged over 30 independent runs, each of 20M steps, on Mujoco Gym and DM Control tasks and over 10 independent runs, each of 200M steps, on Atari tasks. The performance is shown as zero if some of the runs for an algorithm diverged. Classic streaming methods, namely Q($\lambda$) (Watkins 1989) and SARSA($\lambda$) (Rummery & Niranjan 1994), AC($\lambda$) (Williams 1992), perform poorly in these challenging tasks. Similarly, batch RL methods such as PPO, SAC, and DQN struggle when used in streaming learning, which is achieved with a buffer and a batch size of 1 and dubbed as PPO1, SAC1, and DQN1, respectively. Our stream-x methods not only overcome the stream barrier, that is, learn stably and effectively in these tasks, but also become competitive with batch RL methods and even outperform in some environments. For example, Figure 2b shows the performance in the Dog environments where our stream AC($\lambda$) outperforms both PPO and SAC by large margins, achieving the best known performance of any model-free algorithm on this environment. Figure 2c shows the performance in Atari Enduro game where stream Q($\lambda$) outperforms DQN even though it uses a fraction of the memory and compute required by DQN.

### 4.2 Sample efficiency of stream-x algorithms

Here, we study the sample efficiency of our stream-x methods by comparing the learning curves of different algorithms. Figure 3 shows the performance of different deep RL methods on four continuous control MuJoCo Gym tasks. We compare stream AC against the streaming variants PPO1 and SAC1 in addition to their original batch forms, PPO and SAC. We omit Classic AC from this comparison since we found it is extremely unstable such that even with a tiny step size (e.g., $10^{-11}$), it still diverges. Figure 3 shows that stream AC with $\lambda = 0.8$ outperforms PPO1 and SAC1 in all environments and is more sample efficient than PPO in Humanoid-v4, HumanoidStandup-v4, and Ant-v4. Our results present clear evidence contrary to the common belief that streaming methods ought to be sample inefficient.

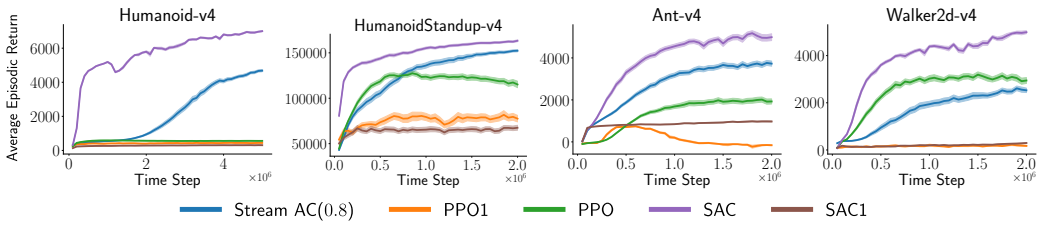



Figure 3: Sample efficiency of stream AC($\lambda$) in MuJoCo Gym environments. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

In Figure 4, we show the performance of stream Q(0.8) against its classic counterpart in addition to DQN1 and DQN on MinAtar tasks. In contrast to the previous experiment, neither classic streaming Q(0.8) nor DQN1 failed in MinAtar tasks. This matches with the observation made by Young & Tian

(2019) where streaming deep RL methods succeeded in MinAtar. We hypothesize that the MinAtar tasks are not challenging enough to study stream barrier, which is observed in other benchmark tasks. Nonetheless, our stream Q(0.8) achieves performance comparable to DQN and better than DQN1 and classic Q(0.8) in most environments. Our results suggest that stream Q(0.8) is as sample efficient as DQN in MinAtar tasks. We repeat this experiment and the next with SARSA in Appendix G.

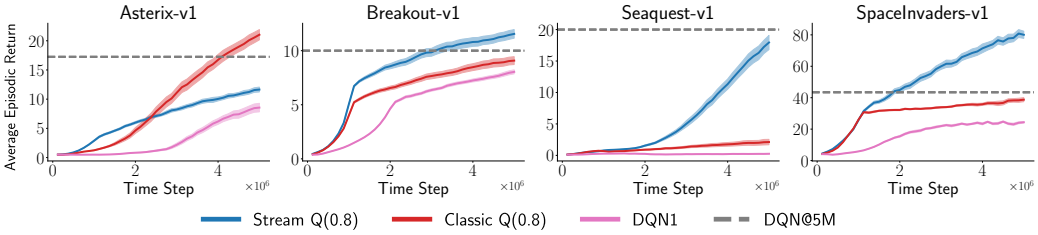



Figure 4: Sample efficiency of stream Q($\lambda$) on MinAtar environments. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

### 4.3 Stability of stream-x algorithms in extended runs

Next, we investigate the stability of our stream-x algorithms when running for an extended period. Such a setup is effective in revealing whether a method can be run for an extended period without any issues. Dohare et al. (2023, also see 2024) studied this setting and showed that PPO experiences some amount of instability that may lead the performance to degrade. In Figure 5, we compare stream AC against SAC, PPO, SAC1, and PPO1 in a number of MuJoCo Gym and DM control tasks where the agents are run for 20M time steps. PPO indeed suffered performance degradation in all tasks and SAC in one of them. We observe that stream AC remains stable and improves its performance, outperforming SAC1, PPO1, and PPO in all tasks. Additionally, stream AC even outperforms SAC in the Dog domain environments (stand and walk), in which SAC diverges before finishing the 20M time steps. Our results demonstrate the superior stability of our method, even when compared to batch RL methods. Stream AC does not experience any instability, sustains extended runs, and continues to improve with more experience, suggesting its potential for lifelong learning applications where extended runs are integral.

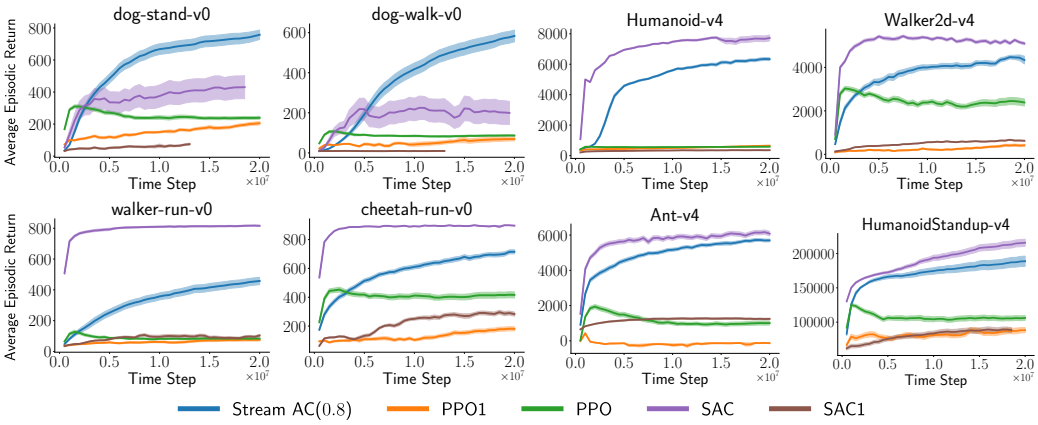



Figure 5: Stability of stream AC($\lambda$) on MuJoCo Gym and DMC environments. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

Now, we are ready to investigate how our stream-x methods perform on Atari's challenging problems. The Atari arcade learning environments are considered hard for multiple reasons, including high dimensional observation space, exploration challenges, and complex dynamics. Thus, the common belief is batch RL becomes necessary to have a data-efficient approach since samples can be reused multiple times (Mnih et al. 2015). Here, we test this assumption by comparing stream Q(0.8) against DQN in addition to the streaming methods, DQN1 and classic Q(0.8). Figure 6 shows the performance of different agents on Atari games experiencing 200M frames in total, which is the standard number

of time steps used in multiple works (e.g., Hessel et al. 2018). The DQN data points are taken from Table 6 in Hessel et al. (2018). We observe that DQN1 fails quickly in all environments, and classic Q(0.8) struggles and suffers from instability. Only stream Q(0.8) represents a strong competitor to DQN. Notably, we found that stream Q(0.8) outperforms DQN in 9 environments and falls behind DQN in 6 environments (see Figure 13 in Appendix G).

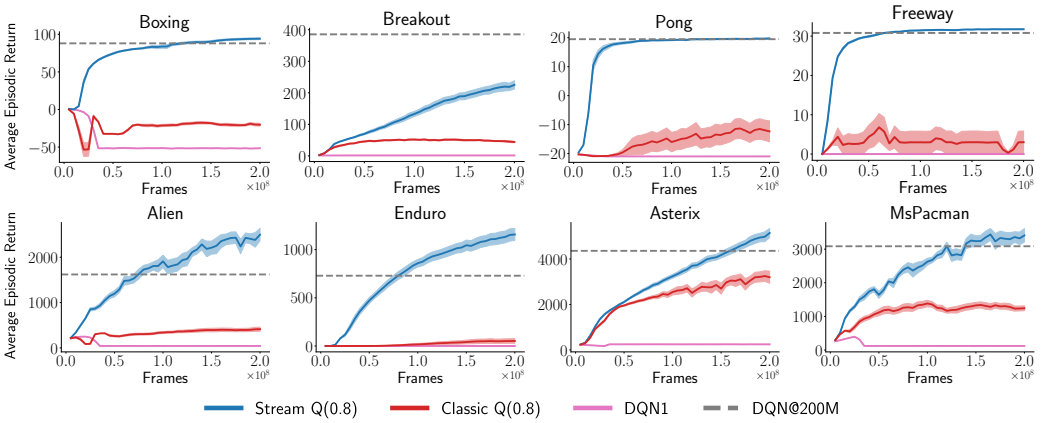



Figure 6: Performance of stream Q($\lambda$) on Atari environments. The results are averaged over 10 independent runs. The shaded area represents a 90% confidence interval.

### 4.4 Understanding the importance of each component in stream-x algorithms

Next, we investigate what makes stream-x algorithms perform well. First, we take stream AC(0.8) and remove each component to determine which contributes the most to performance. Specifically, we remove one of the following components one at a time: ObGD, observation normalization and reward scaling, layer normalization, and sparse initialization. We also compare these variants with classic AC(0.8). Second, we study the role of eligibility traces on performance by comparing stream AC(0) and stream AC(0.8) along with classic AC(0) and classic AC(0.8).

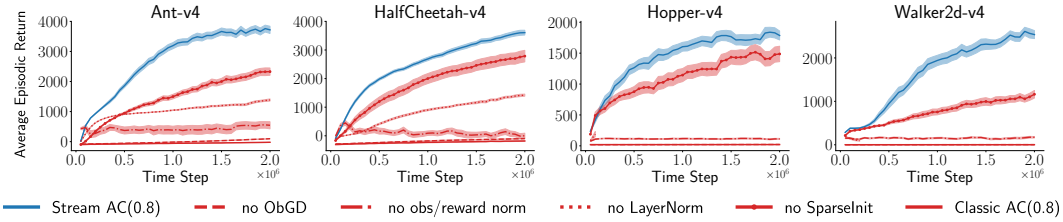



Figure 7: Ablation on the components of stream AC($\lambda$): ObGD, LayerNorm, SparseInit, and data normalization. The shaded area represents a 90% confidence interval.

Figure 7 shows an ablation on the components of stream AC($\lambda$). When we removed sparse initialization and replaced it with LeCun initialization (LeCun et al. 2002), the agent was still able to learn, but slower, confirming the role of sparse initialization in sample efficiency. When we removed layer normalization from stream AC, the performance suffered significantly in all environments, especially Hopper-v4 and Walker-v4. Finally, when we removed observation normalization and reward scaling or replace ObGD with well-tuned Adam, the agent was no longer able to improve its performance. Notably, the largest effect on performance comes from ObGD, indicating its crucial role in achieving stable learning.

Figure 8 shows an ablation of eligibility traces in stream AC($\lambda$). We observe that stream AC benefits if we use eligibility traces, which is visible from the performance improvement of stream AC(0.8) over stream AC(0). On the other hand, we observe that although classic AC with well-tuned Adam is unable to improve its performance in all environments, the addition of eligibility traces in the Ant-v4 even hurts performance, which indicates the additional instability caused by eligibility traces without proper step size adjustment.

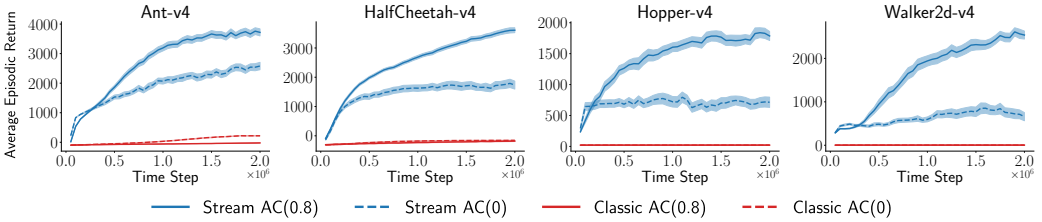



Figure 8: Ablation on the role of eligibility traces in stream AC($\lambda$). The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

### 4.5 Learning how to predict the future

Lastly, we finish with temporal prediction using TD($\lambda$) (Sutton 1988). We use the electricity transformer temperature dataset (Zhou et al. 2021), which has 6 external load features to predict power consumption. The dataset provides the oil temperature readings, which correlate with the power consumption. The goal of the learner is to predict future temperatures, which would help anticipate future power consumption. The dataset is referred to as $\text{ETTm}_2$ (see Figure 9), which represent 2 years worth of data measured every 15 minutes, resulting in a total of 2 year $\times$ 365 days $\times$ 24 hours $\times$ 4 times $= 70,080$ data-points. The environment provides the agent with an observation vector of the 6 feature in addition to the oil temperature from the previous time step. The goal of the agent is to predict future oil temperature using a general value function (GVF, Sutton et al. 2011). This is achieved by extending the return definition to include any scalar signal: $G_t \doteq \sum_{j=0}^{\infty} \gamma^k c_{t+k+1}$, where $c_t$ is some scalar signal known as the cumulant. The cumulant can be chosen to be an entry to the observation vector to perform nexting (see Modayil et al. 2014). For example, a prediction with a horizon of 100 time steps approximately corresponds to a GVF with $\gamma = 0.99$, since the prediction horizon is about $\frac{1}{1-\gamma}$ (see Sutton et al. 2011). In our problem, we use $\gamma = 0.99$, corresponding to a prediction horizon of 25 hours into the future. To allow for such a far prediction horizon, the agent needs some form of encoded information about the history. Following Janjua et al. (2023), we construct memory traces of observations using exponential moving averages (also see Tao et al. 2023, Rafiee et al. 2023). Specifically, given the $i$th entry in the observation vector $O_{t,i}$ at time step $t$, we form the memory trace as $S_{t,i} = \beta S_{t-1,i} + (1-\beta)O_{t,i}$, where $\beta$ is a trace decay factor of 0.999. Those memory traces are used as the agent state in Algorithm 7.

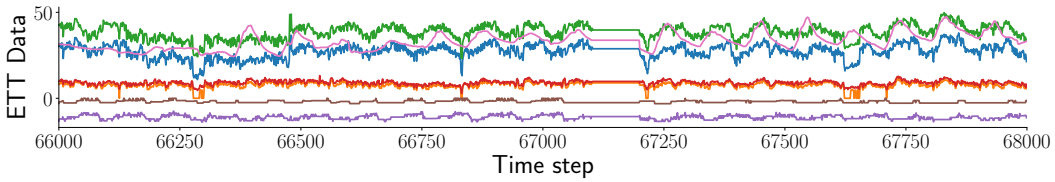
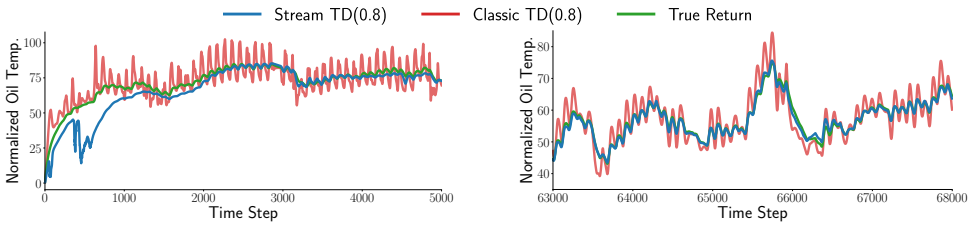



Figure 9: The Electricity Transformer Temperature (ETT) prediction problem. The goal is to predict the oil temperature (purple) using the other 6 external load features.

In Figure 10, we show the performance of stream TD(0.8) against classic TD(0.8) at the beginning and the end of the $\text{ETTm}_2$ time series. We plot the value prediction at each time step in comparison with the true return based on the cumulants. Instead of using the actual temperature values, we normalize them by their minimum and maximum for better visuals; however, this step is not needed. The prediction performance at initialization is poor, but it improves with learning until it closely matches the true return values. Since we scale the rewards in Algorithm 7 by $1/\sigma$, we multiply the standard deviation $\sigma$ to each value function prediction.

## 5 Related works

**Continual Learning.** The goal of continual learning research is to develop algorithms that allow agents to keep learning, potentially forever. The two obstacles in continual learning are loss of plasticity (Dohare et al. 2024) and catastrophic forgetting (McCloskey & Cohen 1989, Hetherington & Seidenberg 1989). Loss of plasticity reduces the agent's ability to learn gradually over time, while




(a) Start of the time series

(b) End of the time series

Figure 10: Performance of stream TD($\lambda$) on the $\text{ETTm}_2$ prediction task. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

catastrophic forgetting prevents it from retaining and utilizing past memories, ultimately hindering its performance improvement. Some works focus on maintaining plasticity primarily in nonstationary supervised learning (e.g., Dohare et al. 2024, Kumar et al. 2023, Lewandowski et al. 2023, Elsayed & Mahmood 2024, Lewandowski et al. 2024, Lee et al. 2024), whereas others focus on reinforcement learning (e.g., Delfosse et al. 2024, Xu et al. 2024, Ma et al. 2024, Lyle et al. 2024a, Lyle et al. 2024b, Lyle et al. 2023, Elsayed et al. 2024b). On the other hand, a few works (e.g., Elsayed & Mahmood 2024, Anand & Precup 2023) address catastrophic forgetting and loss of plasticity at the same time. In particular, Elsayed & Mahmood (2024) addresses catastrophic forgetting by protecting useful weights from drastic change when using gradient-based updates. Another approach to address forgetting is to promote sparse representations. Sparse representations (e.g., Lan & Mahmood 2023) address forgetting since their gradient-based updates make significant changes to the weights of fewer number of connections than dense representations.

Although we use stationary RL tasks in our work, there are commonalities between instability issues in single-task learning and continual learning issues such as loss of plasticity. For example, LayerNorm, recently found effective against loss of plasticity, is also found beneficial against instability in our work. We expect some of the other ideas for addressing loss of plasticity may also be beneficial for single-task streaming RL. Moreover, as our step-size adjustment technique was found effective against learning instability and in alleviating step-size tuning, this benefit may even amplify in continual learning, where hyperparameter tuning is a major stumbling block. Our step-size adjustment technique may also be beneficial for addressing catastrophic forgetting to some extent as it may alleviate interference by reducing the step size when a large update is attempted.

**Step-size adjusting techniques.** One goal in optimization is to find a step size that can make significant but stable progress. Several approaches exist to find a proper step size. For example, Martens & Grosse (2014) introduced Newton's update rule with an efficient approximation to the Fisher block diagonal matrix. Elsayed et al. (2024a) used a Hessian diagonal approximation to determine if the step size is too large and uses it to stabilize RL methods. Sutton (1992) showed how to adjust the step size under nonstationarity using a meta-gradient descent approach called IDBD, which was made more stable with the overshooting prevention mechanisms by many works (Mahmood 2010, Mahmood et al. 2012, Kearney 2023, Dabney & Barto 2012, Javed et al. 2024, McLeod et al. 2021). In addition, Sharifnassab et al. (2024), Schraudolph (1999), and Jacobsen et al. (2019), among many others, presented extensions to the IDBD idea of step size adaptation in neural networks. Our work builds on prior work by Mahmood et al. (2012) and Kearney (2023), extending it to RL and neural networks, and applying a more conservative bound for stability.

**Streaming reinforcement Learning.** Most algorithms introduced by Sutton & Barto (2018) are reinforcement learning algorithms for streaming settings. However, their usage remained limited to tabular and linear approximation cases. Later, several attempts were made to use neural networks with those streaming methods. There are a few works that target the problem setting of streaming reinforcement learning. For example, Elfwing et al. (2018) proposed a new activation function that gives more stability and showed promise with deep Q-learning but still with limited performance. Similarly, Young & Tian (2019) demonstrated the effectiveness of the approach by Elfwing et al. (2018) with AC($\lambda$) and Q($\lambda$) on their MinAtar benchmark, a simplified version of the Atari benchmark (Bellemare et al. 2013). De Asis et al. (2020) introduced an incremental version of REINFORCE that works well in small problems (see also Kimura et al. 1995). Javed et al. (2024) introduced the SwiftTD algorithm for prediction problems by applying SwiftTD to the last linear layer of a deep network and using TD($\lambda$) to all other layers. Modayil & Abbas (2023) introduced a streaming method that learns from unstructured observations.

A closely-related concurrent work is that of Vasan et al. (2024), who developed a streaming deep policy gradient method that also overcomes stream barrier. The main differences are twofold: 1) their method is specific to reparameterization policy gradient whereas we provide a class of algorithms except for reparameterization policy gradient, and 2) unlike their work, all results of our algorithms were obtained using a single set of hyperparameters. Our work is the first comprehensive step toward reviving streaming deep RL by providing a class of algorithms for prediction and both value-based and policy-based control, showing success on most commonly used complex benchmark tasks.

**TinyML.** Performing machine learning algorithms on tiny devices (e.g., microcontrollers) is a challenging task that spans a wide range of applications, such as tiny robots (Neuman et al. 2022) and edge devices (Lin et al. 2022). The main challenge of TinyML is how to fit complex machine learning algorithms into such tiny devices. Most TinyML works are only focused on supporting inference, which does not allow learning. However, on-device training is crucial for continual learning and allows systems to adapt to new changes. Recently, there have been advancements that address this issue and support efficient backpropagation that fits into those tiny devices (Cai et al. 2020, Profentzas et al. 2022, Patil et al. 2022, Lin et al. 2023). Scaling down machine learning to the level of microcontrollers is powerful since there are billions of such devices that can benefit from such technology (Zhu et al. 2022).

## 6 Limitations and future works

Although we have explored a few representative streaming RL algorithms, our approach is compatible with many other algorithms, such as double Q-learning (van Hasselt 2010), dueling Q-learning networks (Wang et al. 2016), noisy networks Q-learning (Fortunato et al. 2018), or even in the continuing setting (e.g., Naik et al. 2024). Our paper focuses on model-free methods, which are less sample-efficient compared to model-based ones; thus, a promising direction is to discover how the agent can incrementally learn a model of the environment to improve sample efficiency following the success of batch model-based methods (e.g., Hafner et al. 2023, Samsami et al. 2024, Liu et al. 2024). Another promising direction is to combine our approach with real-time recurrent learning (Williams & Zipser 1989) to handle partial observability, especially with the recent scalable approaches (e.g., Irie et al. 2024, Zucchet et al. 2024, Elelimy et al. 2024, Javed et al. 2023). In addition, we focus mainly on on-policy methods (except for stream Q) and leave comprehensive experimenting with our approach to a broader range of off-policy methods, such as with importance sampling (e.g., Sutton et al. 2016, He et al. 2023) for future work. Recent step-size adaptation techniques, such as those along the lines of Young et al. (2018), are a potentially useful path to improve our optimizer. Our insights are also relevant to other problem settings, such as parallelized reinforcement learning (e.g., Gallici et al. 2024), to improve the stability or include eligibility traces to other methods.

## 7 Conclusion

In this paper, we addressed—stream barrier—the severe issue of learning instability, often leading to excessive sample inefficiency, and even failure faced by existing streaming reinforcement learning algorithms. We developed *stream-x* algorithms, a class of novel streaming deep RL algorithms based on a set of common techniques that overcome stream barrier. The stream-x algorithms work robustly using a single set of hyperparameters on several benchmark tasks from multiple commonly used suites. Our results with stream AC, which achieves learning efficiency and performance similar to PPO, challenge the prevailing notion that streaming learning algorithms are inherently sample inefficient. The stream-x algorithms are just the beginning of a broader wave of innovations yet to come, serving as a catalyst to revitalize streaming deep RL.

## Acknowledgment

We gratefully acknowledge funding from the Canada CIFAR AI Chairs program, the Reinforcement Learning and Artificial Intelligence (RLAI) laboratory, the Alberta Machine Intelligence Institute (Amii), and the Natural Sciences and Engineering Research Council (NSERC) of Canada. We would also like to thank the Digital Research Alliance of Canada for providing the computational resources needed.

## References


Abbas, Z., Zhao, R., Modayil, J., White, A., & Machado, M. C. (2023). Loss of plasticity in continual deep reinforcement learning. *Conference on Lifelong Learning Agents* (pp. 620-636). (cited on pages 2 and 4)

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., ... & Bachem, O. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*. (cited on page 8)

Anand, N., & Precup, D. (2021). Preferential Temporal Difference Learning. International Conference on Machine Learning (pp. 286-296). (cited on page 3)

Anand, N., & Precup, D. (2023). Prediction and control in continual reinforcement learning. *Advances in Neural Information Processing Systems*, *36*. (cited on page 14)

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, *10*(1-2), 25-61. (cited on page 5)

Armijo, L. (1966). Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, *16*(1), 1-3. (cited on page 5)

Asadi, K., Fakoor, R., & Sabach, S. (2023). Resetting the optimizer in deep rl: An empirical study. *Advances in Neural Information Processing Systems*, *36*. (cited on page 5)

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*. (cited on pages 7 and 7)

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *5*, 834-846. (cited on page 4)

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253-279. (cited on pages 3 and 14)

Bjorck, J., Gomes, C. P., & Weinberger, K. Q. (2021). Is High Variance Unavoidable in RL? A Case Study in Continuous Control. *International Conference on Learning Representations*. (cited on pages 4 and 5)

Cai, H., Gan, C., Zhu, L., & Han, S. (2020). Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, *33*, 11285-11297. (cited on page 15)

Che, F., Vasan, G., & Mahmood, A. R. (2023). Correcting discount-factor mismatch in on-policy policy gradient methods. *International Conference on Machine Learning* (pp. 4218-4240). (cited on page 3)

Dabney, W., & Barto, A. (2012). Adaptive step-size for online temporal difference learning. *AAAI Conference on Artificial Intelligence* (pp. 872-878). (cited on pages 6, 7, and 14)

Delfosse, Q., Schramowski, P., Mundt, M., Molina, A., & Kersting, K. (2024). Adaptive Rational Activations to Boost Deep Reinforcement Learning. *International Conference on Learning Representations*. (cited on page 14)

De Asis, K., Chan, A., Wan, Y., & Sutton, R. S. (2020). Incremental Policy Gradients for Online Reinforcement Learning Control. (cited on page 14)

Dohare, S., Lan, Q., & Mahmood, A. R. (2023b). Overcoming policy collapse in deep reinforcement learning. *European Workshop on Reinforcement Learning*. (cited on pages 2, 4, 5, 11, and 11)

Dohare, S., Hernandez-Garcia, J. F., Rahman, P., Mahmood, A. R., & Sutton, R. S. (2024). Loss of plasticity in deep continual learning. *Nature*, *632*, 768–774. (cited on pages 2, 11, 13, and 14)

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P. L., Bellemare, M. G., & Courville, A. (2023). Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. *International Conference on Learning Representations*. (cited on page 2)

Elsayed, M., & Mahmood, A. R. (2024). Addressing Loss of Plasticity and Catastrophic Forgetting in Continual Learning. *International Conference on Learning Representations*. (cited on pages 2, 14, 14, and 14)

Elsayed, M., Farrahi, H., Dangel, F., & Mahmood, A. R. (2024a). Revisiting Scalable Hessian Diagonal Approximations for Applications in Reinforcement Learning. *International Conference on Machine Learning*. (cited on page 14)

Elsayed, M., Lan, Q., Lyle, C., Mahmood, A. R. (2024b). Weight clipping for deep continual and reinforcement learning. *Reinforcement Learning Conference* (cited on page 14)

Elelimy, E., White, A., Bowling, M., & White, M. (2024). Real-Time Recurrent Learning using Trace Units in Reinforcement Learning. *arXiv preprint arXiv:2409.01449*. (cited on page 15)

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, *107*, 3-11. (cited on pages 2, 3, 14, and 14)

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2020). Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*. (cited on page 8)

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., Legg, S. (2018). Noisy Networks For Exploration. *International Conference on Learning Representations*. (cited on page 15)

Gallici, M., Fellows, M., Ellis, B., Pou, B., Masmitja, I., Foerster, J. N., & Martin, M. (2024). Simplifying Deep Temporal Difference Learning. *arXiv preprint arXiv:2407.04811*. (cited on pages 7 and 15)

Ghiassian, S., Rafiee, B., Lo, Y. L., & White, A. (2020). Improving Performance in Reinforcement Learning by Breaking Generalization in Neural Networks. *International Conference on Autonomous Agents and MultiAgent Systems* (pp. 438-446). (cited on page 5)

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. International Conference on Machine Learning (pp. 1861-1870). (cited on pages 1 and 7)

Haarnoja, T., Moran, B., Lever, G., Huang, S. H., Tirumala, D., Humplik, J., ... & Heess, N. (2024). Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, *9*(89). (cited on page 1)

Hayes, T. L., Kanan, C. (2022). Online continual learning for embedded devices. *Conference on Lifelong Learning Agents*, *PMLR* 199:744–766. (cited on pages 1 and 1)

Hayes, T. L., Krishnan, G. P., Bazhenov, M., Siegelmann, H. T., Sejnowski, T. J., & Kanan, C. (2021). Replay in deep learning: Current approaches and missing biological elements. *Neural Computation*, *33*(11), 2908-2950. (cited on page 1)

Harb, J., & Precup, D. (2017). Investigating recurrence and eligibility traces in deep Q-networks. *arXiv preprint arXiv:1704.05495*. (cited on page 3)

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*. (cited on page 15)

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *AAAI conference on artificial intelligence* (Vol. 32, No. 1). (cited on pages 12 and 12)

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *International Conference on Computer Vision* (pp. 1026-1034). (cited on pages **??**, **??**, and **??**)

He, J., Che, F., Wan, Y., & Mahmood, A. R. (2023). Loosely consistent emphatic temporal-difference learning. In *Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence*. (cited on page 15)

Hetherington, P. A., & Seidenberg, M. S. (1989). Is there 'catastrophic interference' in connectionist networks? *Conference of the Cognitive Science Society* (pp. 26-33). (cited on page 13)

Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., & Wang, W. (2022a). The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*. (cited on page 8)

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & AraÃšjo, J. G. (2022b). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, *23*(274), 1-18. (cited on pages 26, 26, and **??**)

Irie, K., Gopalakrishnan, A., & Schmidhuber, J. (2024). Exploring the Promise and Limits of Real-Time Recurrent Learning. *International Conference on Learning Representations*. (cited on page 15)

Janjua, M. K., Shah, H., White, M., Miahi, E., Machado, M. C., & White, A. (2023). GVFs in the real world: making predictions online for water treatment. *Machine Learning*, 1-31. (cited on page 13)

Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A., & White, M. (2019). Meta-descent for online, continual prediction. *AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 3943-3950). (cited on page 14)

Javed, K., Sharifnassab, A., Sutton, R. S. (2024). SwiftTD: A Fast and Robust Algorithm for Temporal Difference Learning. *Reinforcement Learning Conference*. (cited on pages 6, 7, 14, and 14)

Javed, K., Shah, H., Sutton, R. S., & White, M. (2023). Scalable real-time recurrent learning using columnar-constructive networks. *The Journal of Machine Learning Research*, *24*(1), 12024-12057. (cited on page 15)

Kearney, A. K. (2023). Letting the Agent Take the Wheel: Principles for Constructive and Predictive Knowledge. *PhD Disseretation*. University of Alberta. (cited on pages 6, 6, 7, 14, and 14)

Kingma, D. P. & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*. (cited on page 5)

Kimura, H., Yamamura, M., & Kobayashi, S. (1995). Reinforcement learning by stochastic hill climbing on discounted reward. In Proceedings of the 12th *International Conference on Machine learning*. (cited on page 14)

Knuth, D. E. (2014). *The Art of Computer Programming: Seminumerical Algorithms*, Volume 2. Addison-Wesley Professional. (cited on page 8)

Klopf, A. H. (1972). Brain function and adaptive systems: a heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA. (cited on page 4)

Kumar, S., Marklund, H., & Van Roy, B. (2023). Maintaining plasticity via regenerative regularization. *arXiv preprint arXiv:2308.11958*. (cited on page 14)

Lan, Q., & Mahmood, A. R. (2023). Elephant neural networks: Born to be a continual learner. *arXiv preprint arXiv:2310.01365*. (cited on pages 5 and 14)

Lee, H., Cho, H., Kim, H., Kim, D., Min, D., Choo, J., & Lyle, C. (2024). Slow and Steady Wins the Race: Maintaining Plasticity with Hare and Tortoise Networks. In *Forty-first International Conference on Machine Learning*. (cited on page 14)

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (2002). Efficient backprop. *Neural networks: Tricks of the trade* (pp. 9-50). (cited on pages 5, 7, and 12)

Lewandowski, A., Kumar, S., Schuurmans, D., György, A., & Machado, M. C. (2024). Learning Continually by Spectral Regularization. *arXiv preprint arXiv:2406.06811*. (cited on page 14)

Lewandowski, A., Tanaka, H., Schuurmans, D., & Machado, M. C. (2023). Curvature Explains Loss of Plasticity. *arXiv preprint arXiv:2312.00246*. (cited on page 14)

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. (cited on pages 26, 26, and **??**)

Lin, J., Zhu, L., Chen, W. M., Wang, W. C., Gan, C., & Han, S. (2022). On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, *35*, 22941-22954. (cited on page 15)

Lin, J., Zhu, L., Chen, W. M., Wang, W. C., & Han, S. (2023). Tiny machine learning: progress and futures. *IEEE Circuits and Systems Magazine*, *23*(3), 8-34. (cited on page 15)

Liu, V., Kumaraswamy, R., Le, L., & White, M. (2019). The utility of sparse representations for control in reinforcement learning. *AAAI Conference on Artificial Intelligence* (Vol. 33, No. 01, pp. 4384-4391). (cited on page 5)

Liu, Z., Du, C., Lee, W. S., & Lin, M. (2024). Locality Sensitive Sparse Encoding for Learning World Models Online. *International Conference on Learning Representations*. (cited on page 15)

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N. M., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*. (cited on page 2)

Lyle, C., Zheng, Z., Khetarpal, K., van Hasselt, H., Pascanu, R., Martens, J., & Dabney, W. (2024a). Disentangling the Causes of Plasticity Loss in Neural Networks. *arXiv preprint arXiv:2402.18762*. (cited on page 14)

Lyle, C., Zheng, Z., Nikishin, E., Pires, B. A., Pascanu, R., & Dabney, W. (2023). Understanding plasticity in neural networks. *International Conference on Machine Learning* (pp. 23190-23211). (cited on pages 2, 2, 4, 5, 7, 7, and 14)

Lyle, C., Zheng, Z., Khetarpal, K., Martens, J., van Hasselt, H., Pascanu, R., & Dabney, W. (2024b). Normalization and effective learning rates in reinforcement learning. *arXiv preprint arXiv:2407.01800*. (cited on page 14)

Lyle, C., Rowland, M., & Dabney, W. (2022). Understanding and Preventing Capacity Loss in Reinforcement Learning. *International Conference on Learning Representations*. (cited on page 2)

Mahmood, A. R. (2017). *Incremental Off-policy Reinforcement Learning Algorithms*. PhD thesis, University of Alberta. (cited on pages 2 and 4)

Mahmood, A. R., & Sutton, R. S. (2015). Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*. (cited on page 2)

Mahmood, A. R., Sutton, R. S., Degris, T., & Pilarski, P. M. (2012). Tuning-free step-size adaptation. *IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 2121-2124). (cited on pages 5, 6, 7, 14, and 14)

Mahmood, A. R. (2010). *Automatic Step-size Adaptation in Incremental Supervised Learning*. Master's thesis, University of Alberta. (cited on pages 5 and 14)

Martens, J. (2010). Deep learning via hessian-free optimization. *International Conference on Machine Learning* (Vol. 27, pp. 735-742). (cited on page 5)

Martens, J., & Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. *International Conference on Machine Learning* (pp. 2408-2417). (cited on page 14)

Ma, G., Li, L., Zhang, S., Liu, Z., Wang, Z., Chen, Y., ... & Tao, D. Revisiting Plasticity in Visual Reinforcement Learning: Data, Modules and Training Stages (2024). *International Conference on Learning Representations*. (cited on page 14)

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. International Conference on Machine Learning (Vol. 30, No. 1, p. 3). (cited on pages 26, 26, and **??**)

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, *24*, 109–165. (cited on page 13)

McLeod, M., Lo, C., Schlegel, M., Jacobsen, A., Kumaraswamy, R., White, M., & White, A. (2021). Continual auxiliary task learning. *Advances in Neural Information Processing Systems*, *34*, 12549-12562. (cited on page 14)

Modayil, J., White, A., & Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, *22*(2), 146-160. (cited on page 13)

Modayil, J., & Abbas, Z. (2023). Towards model-free RL algorithms that scale well with unstructured data. *arXiv preprint arXiv:2311.02215*. (cited on page 14)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533. (cited on pages 1, 2, 2, and 11)

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning* (pp. 1928-1937). (cited on pages 2 and 8)

Nauman, M., Bortkiewicz, M., Miłoś, P., Trzcinski, T., Ostaszewski, M., & Cygan, M. (2024). Overestimation, Overfitting, and Plasticity in Actor-Critic: the Bitter Lesson of Reinforcement Learning. *International Conference on Machine Learning*. (cited on page 7)

Naik, A., Wan, Y., Tomar, M., & Sutton, R. S. (2024). Reward Centering. *Reinforcement Learning Journal, vol. 4, 2024, pp.* . (cited on page 15)

Neuman, S. M., Plancher, B., Duisterhof, B. P., Krishnan, S., Banbury, C., Mazumder, M., ... & Reddi, V. J. (2022). Tiny robot learning: challenges and directions for machine learning in resource-constrained robots. *International Conference on Artificial Intelligence Circuits and Systems* (pp. 296-299). (cited on pages 1 and 15)

Nguyen, T. Q., & Chiang, D. (2017). Improving lexical choice in neural machine translation. *arXiv preprint arXiv:1710.01329*. (cited on page 7)

Nota, C. & Thomas, P. S. (2019). Is the policy gradient a gradient? *arXiv preprint arXiv:1906.07073*. (cited on page 3)

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1-17. (cited on page 4)

Pan, Y., Banman, K., & White, M. (2021). Fuzzy Tiling Activations: A Simple Approach to Learning Sparse Representations Online. *International Conference on Learning Representations*. (cited on page 5)

Patil, S. G., Jain, P., Dutta, P., Stoica, I., & Gonzalez, J. (2022). POET: Training neural networks on tiny devices with integrated rematerialization and paging. *International Conference on Machine Learning* (pp. 17573-17583). (cited on page 15)

Pardo, F., Tavakoli, A., Levdik, V., & Kormushev, P. (2018). Time limits in reinforcement learning. *In International Conference on Machine Learning* (pp. 4045-4054). (cited on pages 27, 27, and **??**)

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... & Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS Workshop Autodiff*. (cited on pages 26, **??**, and **??**)

Profentzas, C., Almgren, M., & Landsiedel, O. (2022). MiniLearn: On-Device Learning for Low-Power IoT Devices. *International Conference on Embedded Wireless Systems and Networks* (pp. 1-11). (cited on page 15)

Rao, N., Aljalbout, E., Sauer, A., & Haddadin, S. (2020). How to make deep RL work in practice. *arXiv preprint arXiv:2010.13083*. (cited on page 8)

Rafiee, B., Abbas, Z., Ghiassian, S., Kumaraswamy, R., Sutton, R. S., Ludvig, E. A., & White, A. (2023). From eye-blinks to state construction: Diagnostic benchmarks for online representation learning. *Adaptive behavior*, *31*(1), 3-19. (cited on page 13)

Riedmiller, M. (2005). Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. *European conference on Machine Learning* (pp. 317-328). (cited on page 2)

Rummery, G. A., & Niranjan, M. (1994). On-line Q-learning using connectionist systems (cited on page 10)

Samsami, M. R., Zholus, A., Rajendran, J., & Chandar, S. (2024). Mastering memory tasks with world models. *arXiv preprint arXiv:2403.04253*. (cited on page 15)

Sharifnassab, A., Salehkaleybar, S., & Sutton, R. (2024). MetaOptimize: A Framework for Optimizing Step Sizes and Other Meta-parameters. *arXiv preprint arXiv:2402.02342*. (cited on page 14)

Schraudolph, N. N. (2002). Centering neural network gradient factors. *In Neural Networks: Tricks of the Trade* (pp. 207-226). (cited on page 7)

Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. (2015). Trust Region Policy Optimization. *International Conference on Machine Learning* (pp. 1889-1897). (cited on page 5)

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. (cited on pages 7 and 8)

Schwarzer, M., Ceron, J. S. O., Courville, A., Bellemare, M. G., Agarwal, R., & Castro, P. S. (2023). Bigger, better, faster: Human-level atari with human-level efficiency. *International Conference on Machine Learning* (pp. 30365-30380). (cited on page 2)

Schaul, T., Ostrovski, G., Kemaev, I., & Borsa, D. (2021). Return-based scaling: Yet another normalisation trick for deep RL. *arXiv preprint arXiv:2105.05347*. (cited on page 8)

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. *International Conference on Artificial Neural Networks*. (cited on page 14)

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354-359. (cited on page 1)

Smith, L., Kostrikov, I., & Levine, S. (2023). Demonstrating a walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *Robotics: Science and Systems (RSS) Demo, 2(3):4*. (cited on page 1)

Sokar, G., Mocanu, E. & Mocanu, D. (2022). Dynamic Sparse Training for Deep Reinforcement Learning. *International Joint Conference on Artificial Intelligence*. (cited on page 5)

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press. (cited on pages 1, 2, 4, 4, and 14)

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9-44. (cited on pages 2, 5, and 13)

Sutton, R. S. (1988). NADALINE: A normalized adaptive linear element that learns efficiently. *GTE TR88-509.4), GTE Laboratories Incorporated*. (cited on page 7)

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, *12*. (cited on page 3)

Sutton, R. S., & Barto, A. G. (1981). Toward a modern theory of adaptive networks: expectation and prediction. *Psychological Review*, *88*(2), 135. (cited on page 4)

Sutton, R. S., Mahmood, A. R., & White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research*, *17*(73), 1-29. (cited on page 15)

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *International Conference on Autonomous Agents and Multiagent Systems*, Volume 2 (pp. 761-768). (cited on pages 13 and 13)

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. AAAI Conference on Artificial Intelligence (Vol. 92, pp. 171-176). (cited on page 14)

Tao, R. Y., White, A., & Machado, M. C. (2023). Agent-State Construction with Auxiliary Inputs. *Transactions on Machine Learning Research*. (cited on page 13)

Thomas, P. (2014). Bias in natural actor-critic algorithms. *International Conference on Machine Learning* (pp. 441–448). (cited on page 3)

Thodoroff, P., Anand, N., Caccia, L., Precup, D., & Pineau, J. (2019). Recurrent value functions. *arXiv preprint arXiv:1905.09562*. (cited on page 2)

Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. (cited on pages 24, **??**, and **??**)

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *IEEE/RSJ international conference on intelligent robots and systems* (pp. 5026-5033). (cited on page 3)

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., ... & Younis, O. G. (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*. (cited on pages 26, **??**, and **??**)

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., ... & Tassa, Y. (2020). dm_control: Software and tasks for continuous control. *Software Impacts*, *6*, 100022. (cited on page 3)

Vasan, G., Elsayed, M., Azimi, S. A., He, J., Shahriar, F., Bellinger, C., White, M., & Mahmood, A. R. (2024). Deep policy gradient methods without batch updates, target networks, or replay buffers. *To appear in Neural Information Processing Systems*. (cited on page 15)

Van de Ven, G. M., Siegelmann, H. T., & Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, *11*(1), 4069. (cited on page 1)

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., & Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research 17*(1):5057-5096. (cited on page 2)

van Hasselt, H., Mahmood, A. R., & Sutton, R. S. (2014). Off-policy TD($\lambda$) with a true online equivalence. *Conference on Uncertainty in Artificial Intelligence*. (cited on page 2)

van Hasselt, H., Madjiheurem, S., Hessel, M., Silver, D., Barreto, A., & Borsa, D. (2021). Expected eligibility traces. *AAAI Conference on Artificial Intelligence* (Vol. 35, No. 11, pp. 9997-10005). (cited on page 2)

van Hasselt, H. P., Guez, A., Hessel, M., Mnih, V., & Silver, D. (2016). Learning values across many orders of magnitude. *Advances in neural information processing systems*, *29*. (cited on page 8)

van Hasselt, H. (2010). Double Q-learning. Advances in neural information processing systems, *23*. (cited on page 15)

Verma, V., Maimone, M. W., Gaines, D. M., Francis, R., Estlin, T. A., Kuhn, S. R., ... & Thiel, E. R. (2023). Autonomous robotics is driving Perseverance rover's progress on Mars. *Science Robotics*, *8*(80). (cited on page 1)

Veeriah, V., van Seijen, H., & Sutton, R. S. (2017). Forward actor-critic for nonlinear function approximation in reinforcement learning. *Conference on Autonomous Agents and MultiAgent Systems* (pp. 556-564). (cited on pages 3, 4, and 5)

Wang, Y., Vasan, G., & Mahmood, A. R. (2023). Real-time reinforcement learning for vision-based robotics utilizing local and remote computers. *International Conference on Robotics and Automation* (pp. 9435-9441). (cited on page 2)

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning* (pp. 1995-2003). (cited on page 15)

Watkins, C. J. C. H. (1989) *Learning from Delayed Rewards*. PhD Thesis, University of Cambridge, England. (cited on page 10)

Welford, B. P. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, *4*(3), 419-420. (cited on page 8)

White, M., & White, A. (2016). A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning. *International Conference on Autonomous Agents* & Multiagent Systems (pp. 557-565). (cited on page 2)

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*, 229-256. (cited on page 10)

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270-280. (cited on page 15)

Xu, J., Sun, X., Zhang, Z., Zhao, G., & Lin, J. (2019). Understanding and improving layer normalization. *Advances in neural information processing systems*, *32*. (cited on pages 7 and 7)

Xu, G., Zheng, R., Liang, Y., Wang, X., Yuan, Z., Ji, T., ... & Xu, H. DrM: Mastering Visual Reinforcement Learning through Dormant Ratio Minimization (2024). *International Conference on Learning Representations*. (cited on page 14)

Young, K., & Tian, T. (2019). Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*. (cited on pages 3, 3, 11, and 14)

Young, K., Wang, B., & Taylor, M. E. (2018). Metatrace actor-critic: Online step-size tuning by meta-gradient descent for reinforcement learning control. *arXiv preprint arXiv:1805.04514*. (cited on page 15)

Yuan, Y., & Mahmood, A. R. (2022). Asynchronous reinforcement learning for real-time control of physical robots. *International Conference on Robotics and Automation*. (cited on page 2)

Zhang, S., Laroche, R., van Seijen, H., Whiteson, S., & Tachet des Combes, R. (2022). A deeper look at discounting mismatch in actor-critic algorithms. *International Conference on Autonomous Agents and Multiagent Systems* (pp. 1491–1499). (cited on page 3)

Zhang, B., & Sennrich, R. (2019). Root mean square layer normalization. *Advances in Neural Information Processing Systems*, *32*. (cited on page 7)

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *AAAI Conference on Artificial Intelligence* (Vol. 35, No. 12, pp. 11106-11115). (cited on pages 3 and 13)

Zhu, S., Voigt, T., Ko, J., & Rahimian, F. (2022). On-device training: A first overview on existing systems. *arXiv preprint arXiv:2212.00824*. (cited on page 15)

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S. C., Dvornek, N., Papademetris, X., & Duncan, J. (2020). Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in Neural Information Processing Systems*, *33*, 18795-18806 (cited on pages 24, **??**, and **??**)

Zucchet, N., Meier, R., Schug, S., Mujika, A., & Sacramento, J. (2024). Online learning of long-range dependencies. *Advances in Neural Information Processing Systems*, *36*. (cited on page 15)

## A The stream SARSA($\lambda$) algorithm

**Algorithm 10** Stream SARSA($\lambda$)

**Given** LayerNorm action-value network $\hat{q}(s, a; \boldsymbol{w})$ with vectorized weights vector and initialized with SparseInit
**Initialize** discount factor $\gamma$ (e.g. 0.99) and eligibility traces parameter $\lambda$ (e.g. 0.9)
**Initialize** step size $\alpha$ (e.g., 1), and scaling factor $\kappa_{\hat{q}}$ (e.g., 2)
**Initialize** $p_r, p_S$ to zero and $\mu_S, t$ to one
**for** each episode **do**
  $\boldsymbol{z_w} \leftarrow \mathbf{0}$
  Initialize $S$ (first state of the episode)
  $S, \mu_S, p_S, \leftarrow$ NormalizeObservation($S, \mu_S, p_S, t$)
  Choose $A$ from $S$ using policy derived from $\hat{q}$ (e.g., $\epsilon$-greedy)
  **for** each time step in the episode **do**
    $t \leftarrow t + 1$
    Take action $A$, observe $S'$, $R$, $T$ ▷ $T$ indicates whether $S'$ is a terminal state
    $S', \mu_S, p_S \leftarrow$ NormalizeObservation($S', \mu_S, p_S, t$)
    $R, p_r, \leftarrow$ ScaleReward($R, \gamma, p_r, T, t$)
    Choose $A'$ from $S'$ using policy derived from $\hat{q}$ (e.g., $\epsilon$-greedy)
    $\delta \leftarrow R + \gamma\hat{q}(S', A', \boldsymbol{w}) - \hat{q}(S, A, \boldsymbol{w})$ ▷ if $S'$ is terminal, then $\hat{q}(S', ., \boldsymbol{w}) \doteq 0$
    $\boldsymbol{z_w} \leftarrow \gamma\lambda\boldsymbol{z_w} + \nabla_{\boldsymbol{w}}\hat{v}(S, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow$ ObGD($\boldsymbol{z_w}, \boldsymbol{w}, \delta, \alpha_{\hat{q}}, \kappa_{\hat{q}}$)
    $S \leftarrow S'$

## B Adaptive Overshooting-bounded Gradient Descent

Here, we create an adaptive version of our method. Let us consider an RMSProp-based optimizer (Tieleman & Hinton 2012). The update vector would be $\frac{\alpha\delta\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}$, where $\boldsymbol{v}$ is a vector containing the second moments of $\delta\boldsymbol{z}$ and $\epsilon$ is a small number for numerical stability. Repeating the analysis we have in Section 3.2, we can end up with the following:

$$
\begin{aligned}
\xi &= \alpha\left(\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right)^\top (\gamma\nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x}') - \nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x})) \qquad \text{(under local linearity)} \\
&\leq \alpha\left|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right|^\top |\gamma\nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x}') - \nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x})| \\
&\leq \alpha\left|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right|^\top \mathbf{1} \\
&= \alpha\left\|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right\|_1 \leq \kappa\alpha\left\|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right\|_1, \qquad \text{where } \kappa > 1 \\
&\leq \kappa\alpha\bar{\delta}\left\|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right\|_1, \qquad \text{where } \bar{\delta} = \max(|\delta|, 1)
\end{aligned} \tag{5}
$$

using the same assumption that all entries of $|\gamma\nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x}') - \nabla_{\boldsymbol{w}}v(\boldsymbol{w};\boldsymbol{x})| \leq 1$ and performing the division element-wise in $\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}$. The derivation also works if we replace the second-moment estimator with a variance estimator (cf. Zhuang et al. 2020). In Algorithm 11, we show the Adaptive ObGD optimizer pseudocode.

**Algorithm 11** Adaptive ObGD

**Require**: Eligibility trace $\boldsymbol{z_w}$, weight vector $\boldsymbol{w}$, error $\delta$, step size $\alpha$, scaling factor $\kappa$, second moment estimate $\boldsymbol{v}$, decay factor $\beta$, small number $\epsilon$
$\boldsymbol{v} \leftarrow \beta\boldsymbol{v} + (1-\beta)\delta\boldsymbol{z_w}$ ▷ Update the second moment of semi-gradients
$\bar{\delta} = \max(|\delta|, 1)$
$M \leftarrow \alpha\kappa\bar{\delta}\left\|\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}\right\|_1$ ▷ Note that $\boldsymbol{z_w} = \nabla_{\boldsymbol{w}}f$ for supervised learning
$\alpha \leftarrow \min\left(\frac{\alpha}{M}, \alpha\right)$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\frac{\delta\boldsymbol{z}}{\sqrt{\boldsymbol{v}+\epsilon}}$
**return** $\boldsymbol{w}$

## C Overshooting bounds for supervised regression

The update vector in supervised regression is given by $\boldsymbol{u} = -\alpha \nabla_{\boldsymbol{w}} \mathcal{L}$ for the input-target pair $(\boldsymbol{x}, y)$. We write the effective step size for supervised regression with the following:

$$
\begin{aligned}
\xi &= \frac{(f(\boldsymbol{x};\boldsymbol{w}) - y) - (f(\boldsymbol{x};\boldsymbol{w}_+) - y)}{\delta} \\
&= \frac{(f(\boldsymbol{x};\boldsymbol{w}) - y) - (f(\boldsymbol{x};\boldsymbol{w}) - \alpha \nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w})^\top \nabla_{\boldsymbol{w}} \mathcal{L} - y)}{\delta} \quad \text{(under local linearity)} \\
&= \frac{\delta \alpha \nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w})^\top f(\boldsymbol{x};\boldsymbol{w})}{\delta} \\
&= \alpha \nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w})^\top \nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w}) \\
&\leq \alpha \kappa \|\nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w})\|_1, \qquad \text{where } \kappa > 1 \\
&\leq \kappa \alpha \bar{\delta} \|\nabla_{\boldsymbol{w}} f(\boldsymbol{x};\boldsymbol{w})\|_1, \qquad \text{where } \bar{\delta} = \max(|\delta|, 1)
\end{aligned} \tag{6}
$$

## D Bounding effective step size for linear functions

Here, we provide two simple examples to show how to bound the effective step size in supervised regression and temporal difference learning with linear function approximation. First, we consider the problem of supervised regression with squared error loss. Let us consider function $f$ to be linear where $y$ is the target and $\boldsymbol{w}^T\boldsymbol{x}$ is the prediction. The loss is given by $\mathcal{L} = \frac{1}{2}(\boldsymbol{w}^\top \boldsymbol{x} - y)^2$. The effective step size for a given input-output pair $(\boldsymbol{x}, y)$ is:

$$
\begin{aligned}
\xi &= \frac{(\boldsymbol{w}^\top \boldsymbol{x} - y) - (\boldsymbol{w}_+^\top \boldsymbol{x} - y)}{\delta} = \frac{(\boldsymbol{w}^\top \boldsymbol{x} - y) - ((\boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} \mathcal{L})^\top \boldsymbol{x} - y)}{\delta} \\
&= \frac{(\boldsymbol{w}^\top \boldsymbol{x} - y) - ((\boldsymbol{w} - \alpha \delta \boldsymbol{x})^\top \boldsymbol{x} - y)}{\delta} = \frac{\delta \alpha \boldsymbol{x}^\top \boldsymbol{x}}{\delta} = \alpha \boldsymbol{x}^\top \boldsymbol{x},
\end{aligned}
$$

where $\boldsymbol{w}_+$ is the weight vector after a gradient descent update is performed. Overshooting happens whenever $\alpha \boldsymbol{x}^\top \boldsymbol{x} > 1$, which leads to the same condition used in AutoStep algorithm (Mahmood et al. 2012).

Next, we find the overshooting condition for TD($\lambda$) with linear function approximation. The TD error is given by $\delta = r + \gamma \boldsymbol{w}^T \boldsymbol{x}' - \boldsymbol{w}^T \boldsymbol{x}$, where $r$ is the reward at the current time step, $\boldsymbol{x}$ is the feature vector at the current time step, and $\boldsymbol{x}'$ is the feature vector at the next time step. Remember that for semi-gradient TD($\lambda$), we have $\boldsymbol{w}_+ = \boldsymbol{w} + \delta \boldsymbol{z}$, where $\boldsymbol{z}$ is the eligibility trace vector and $\boldsymbol{x}$ is the feature vector. The effective step size for semi-gradient TD($\lambda$) is given by

$$
\begin{aligned}
\xi &= \frac{(r + \gamma \boldsymbol{w}^\top \boldsymbol{x}' - \boldsymbol{w}^\top \boldsymbol{x}) - (r + \gamma \boldsymbol{w}_+^T \boldsymbol{x}' - \boldsymbol{w}_+^\top \boldsymbol{x})}{\delta} \\
&= \frac{-\alpha \delta \boldsymbol{z}^\top \boldsymbol{x} + \gamma \alpha \delta \boldsymbol{z}^\top \boldsymbol{x}'}{\delta} = \alpha \boldsymbol{z}^\top (\gamma \boldsymbol{x}' - \boldsymbol{x}).
\end{aligned}
$$

The overshooting happens when $\alpha \boldsymbol{z}^\top (\gamma \boldsymbol{x}' - \boldsymbol{x}) > 1$, which is similar to the condition given by Dabney & Barto (2012) and in the Auto algorithm (McLeod et al. 2021). In these two examples, the condition we ended up detecting overshooting without any additional forward passes, which is only the case for linear function approximation.

## E Entropy regularization and eligibility traces

The entropy regularized gradient $\delta_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}_t) + \tau \nabla_{\boldsymbol{\theta}} H(\cdot|S_t, \boldsymbol{\theta}_t)$, where $\tau \in [0, \infty)$, has been shown to promote exploration (Mnih et al. 2016). We use an adaptive entropy regularization: $\delta_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}_t) + |\delta_t| \tau \nabla_{\boldsymbol{\theta}} H(\cdot|S_t, \boldsymbol{\theta}_t)$, which makes the entropy contribution proportional to the magnitude of the TD error. We can rewrite this gradient as $\delta_t(\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}_t) + \tau \text{sign}(\delta_t) \nabla_{\boldsymbol{\theta}} H(\cdot|S_t, \boldsymbol{\theta}_t))$, for which eligibility trace vector is defined as $\boldsymbol{z}_t = \gamma \lambda \boldsymbol{z}_{t-1} + (\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}_t) + \tau \text{sign}(\delta_t) \nabla_{\boldsymbol{\theta}} H(\cdot|S_t, \boldsymbol{\theta}_t))$. This definition of the entropy-regularized gradient can be easily implemented with our ObGD optimizer since it is in the form of an error multiplied by a vector (see Section 3.2).

# F Experimental Details

We implemented the algorithms in Python and used PyTorch (Paszke et al. 2017) for automatic differentiation to backpropagate gradients in neural networks. In addition, we use the Gymnasium (Towers et al. 2024) framework for environment implementations.

## F.1 Electricity transformer temperature prediction

We used a $128 \times 128$ fully connected network with LeakyReLU activations (Maas et al. 2013) where LayerNorm (Ba et al. 2016) is added before each activation layer. Stream TD($\lambda$) uses $\lambda = 0.8$ and $\gamma = 0.99$. We used $\kappa = 2$ and a step size $\alpha$ of 1 for ObGD. The agent experiences 6700 time steps in total. Lastly, we used sparse initialization with a sparsity ratio $s$ of $90\%$.

For classic TD, we use Adam optimizer (Kingma & Ba 2015) with a step size of $3 \times 10^{-4}$ using the default $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$.

## F.2 MinAtar

We used a network composed of a convolutional layer (LeCun et al. 1998) with 16 filters of size $3 \times 3$ and a stride of 1 followed by a fully connected layer with 1024 hidden units and another one with 128 hidden units. We used LeakyReLU activations (Maas et al. 2013) where LayerNorm (Ba et al. 2016) is added before each activation layer. Both stream Q($\lambda$) and stream SARSA($\lambda$) use $\lambda = 0.8$ and $\gamma = 0.99$. We used $\kappa = 2$ and a step size $\alpha$ of 1 for ObGD. The agent experiences 5M time steps in total and uses an $\epsilon$-greedy policy where $\epsilon$ starts with 1 and decreases to 0.01 with a linear schedule that reaches $\epsilon = 0.01$ at $5\%$ of the total time steps used. Lastly, we used sparse initialization with a sparsity ratio $s$ of $90\%$.

We used the implementation of CleanRL (Huang et al. 2022b) for DQN with the same hyperparameter set and changed the batch size and replay buffer sizes to 1 to obtain DQN1. Additionally, we make learning start from the first time step with a train frequency of 1 (updating each time step). The DQN at 5M data points used in our MinAtar plots are taken from Figure 3 in the work by Young & Tian (2019).

For classic Q($\lambda$) and SARSA($\lambda$), we use Adam optimizer (Kingma & Ba 2015) with a step size of $10^{-5}$ using the default $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$. We kept reducing Adam's step size from its default value of $3 \times 10^{-4}$ until we found a step size that did not cause divergence.

## F.3 Atari

We used a network composed of a convolutional layer (LeCun et al. 1998) with 32 filters of size $8 \times 8$ and using a stride of 5 followed by another convolutional layer with 64 filters of size $4 \times 4$ and using a stride of 3 followed by another convolutional layer with 64 filters of size $3 \times 3$ and using a stride of 2. The output of the last convolutional layer is flattened and is fed to a fully connected layer with 256 hidden units and another one with 128 hidden units. We used LeakyReLU activations (Maas et al. 2013) where LayerNorm (Ba et al. 2016) is added before each activation layer. Stream Q($\lambda$) and stream SARSA($\lambda$) use $\lambda = 0.8$ and $\gamma = 0.99$. We used $\kappa = 2$ and a step size $\alpha$ of 1 for ObGD. The agent experiences 50M time steps or 200M frames in total and uses an $\epsilon$-greedy policy where $\epsilon$ starts with 1 and decreases to 0.01 with a linear schedule that reaches $\epsilon = 0.01$ at $5\%$ of the total time steps used. Each action taken by the agent is repeated 4 times. Lastly, we used sparse initialization with a sparsity ratio $s$ of $90\%$.

The Atari environments are prepossessed, the same way as other works (e.g., Rainbow, Hessel et al. 2018). We downsample the frames to $84 \times 84$ then we convert each frame from RGB to grayscale. To address partial observability, we stack 4 frames. No clipping for the rewards is performed, and no division by 255 is done for the frames. We made the agent take a random action at the start of the episode for environments that are fixed until firing. The episode is terminated on loss of life. Finally, we make the agent take a random number of no-operation (no-op) actions (up to 30) at the beginning of each episode

We used the implementation of CleanRL (Huang et al. 2022b) for DQN with the same hyperparameter set and changed the batch size and replay buffer sizes to 1 to obtain DQN1. Additionally, we made

learning start from the first time step with a train frequency of 1 (updating each time step). The DQN at 200M data points used in our Atari plots were taken from Table 6 in the work by Hessel et al. (2018).

For classic Q($\lambda$) and classic SARSA($\lambda$), we use Adam optimizer (Kingma & Ba 2015) with a step size of $10^{-5}$ using $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$. We kept reducing Adam's step size from its default value of $3 \times 10^{-4}$ until we found a step size that did not cause divergence.

### F.4 MuJoCo Gym and DM Control

We used a $128 \times 128$ fully connected network with LeakyReLU activations (Maas et al. 2013) where LayerNorm (Ba et al. 2016) is added before each activation layer. In the last layer of the policy network, we used two heads: one for the actions mean and the other for actions standard deviation. We used separate networks for the policy and value functions. In continuous control, the standard deviation is parameterized by the SoftPlus function: $f(x) = \log(1 + e^x)$. For numerical stability, when the input to the function exceeds a threshold of 20, we used a linear mapping of $y = x$. The actions are clamped to be in the range $[-1, 1]$. In discrete control, we used softmax policy parameterization. Stream AC($\lambda$) uses $\lambda = 0.8$ and $\gamma = 0.99$. We used $\kappa = 3$ in the policy network and $\kappa = 2$ in the value network for ObGD using a step size $\alpha$ of 1. The agent experiences 20M time steps in total. Lastly, we used sparse initialization with a sparsity ratio $s$ of $90\%$. Since MuJoCo and DM control environments use time limits to have episodes with bounded lengths, this practice introduces partial observability and forces the agent to make conflicting updates at states where truncation happens, potentially creating learning instability (Pardo et al. 2018). We followed the recommendation by Pardo et al. (2018) to include the time step as a part of the agent observation, distinguishing between terminations due to timeouts or the environment itself. The remaining time is normalized to be in the range $[-1/2, 1/2]$, where $1/2$ marks the end of the episode.

We used the implementation of CleanRL (Huang et al. 2022b) for PPO with the same hyperparameter set. For PPO1, we changed the mini-batch size, replay buffer size, and number of epochs to 1.

For classic AC($\lambda$), we use Adam optimizer (Kingma & Ba 2015) with a step size of $10^{-7}$ using the default $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-4}$ in the ablation study in Figure 7 and Figure 8. We kept reducing Adam's step size from its default value of $3 \times 10^{-4}$ until we found a step size that did not cause divergence during 2M time steps for HalfCheetah-v4 and Ant-v4. However, we could not find a step size that did not cause divergence for Hooper-v4 and Walker-2d even when using a step size of $10^{-11}$, so we used the same step size used in HalfCheetah-v4 and Ant-v4 for simplicity. On the other hand, we could not find any step size that did not cause divergence in most MuJoCo and DM control environments, so we decided to drop them from our Figure 3 and Figure 5.

# G Additional Results

## G.1 SARSA($\lambda$) in MinAtar environments

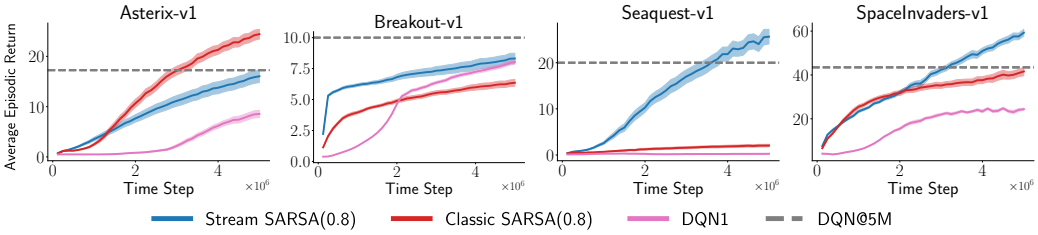



Figure 11: Performance of stream SARSA($\lambda$) on MinAtar environments. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.

## G.2 Q($\lambda$) and SARSA($\lambda$) in Atari environments

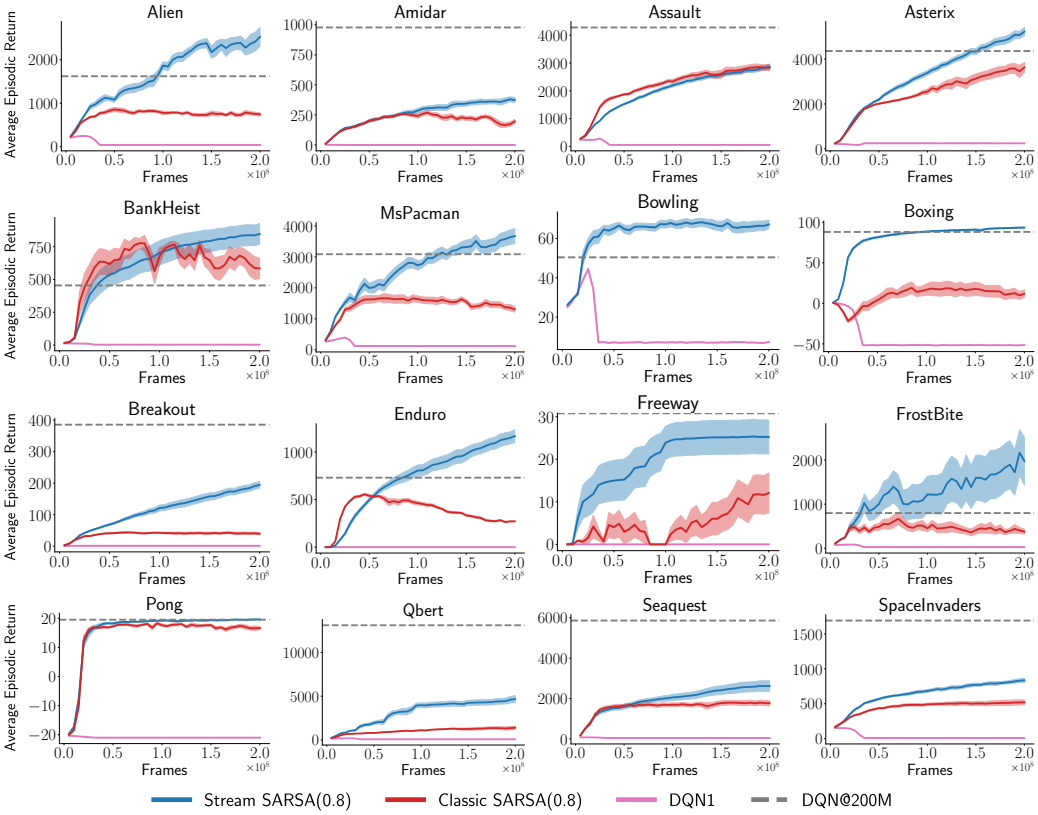



Figure 12: Performance of stream SARSA($\lambda$) on Atari environments. The results are averaged over 10 independent runs. The shaded area represents a 90% confidence interval.

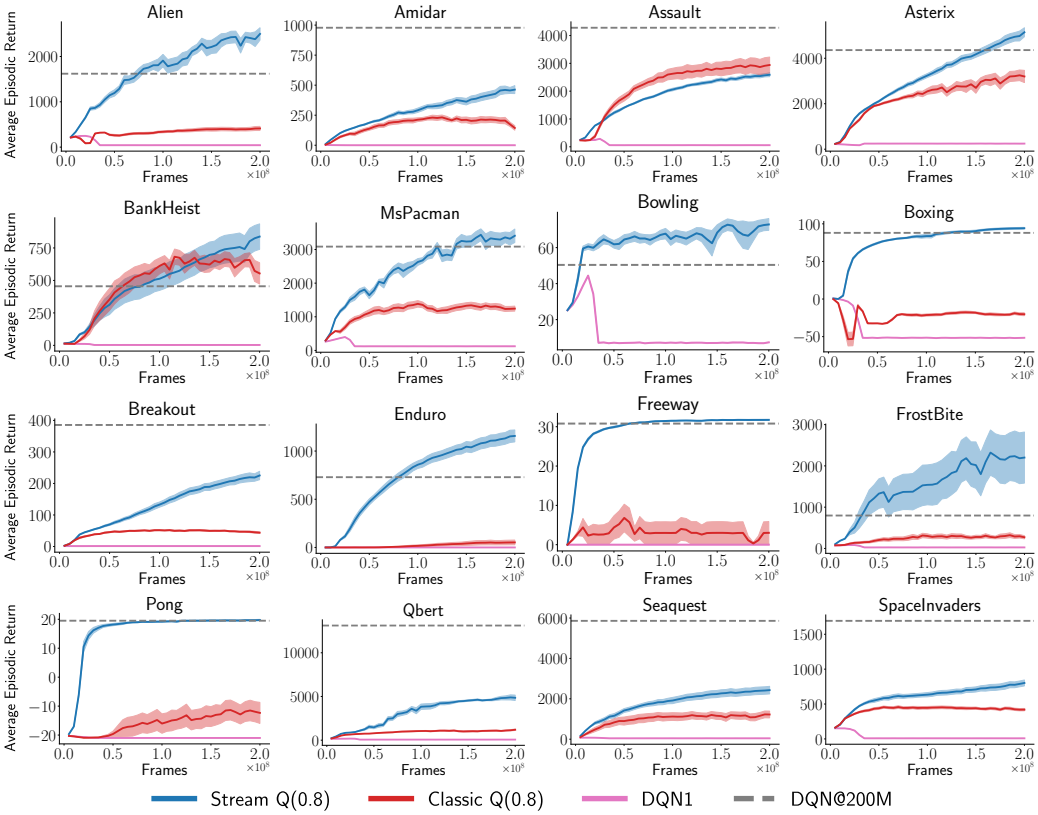



Figure 13: Performance of stream Q($\lambda$) on Atari environments. The results are averaged over 10 independent runs. The shaded area represents a 90% confidence interval.

## G.3 AC($\lambda$) on MuJoCo and DM Control environments

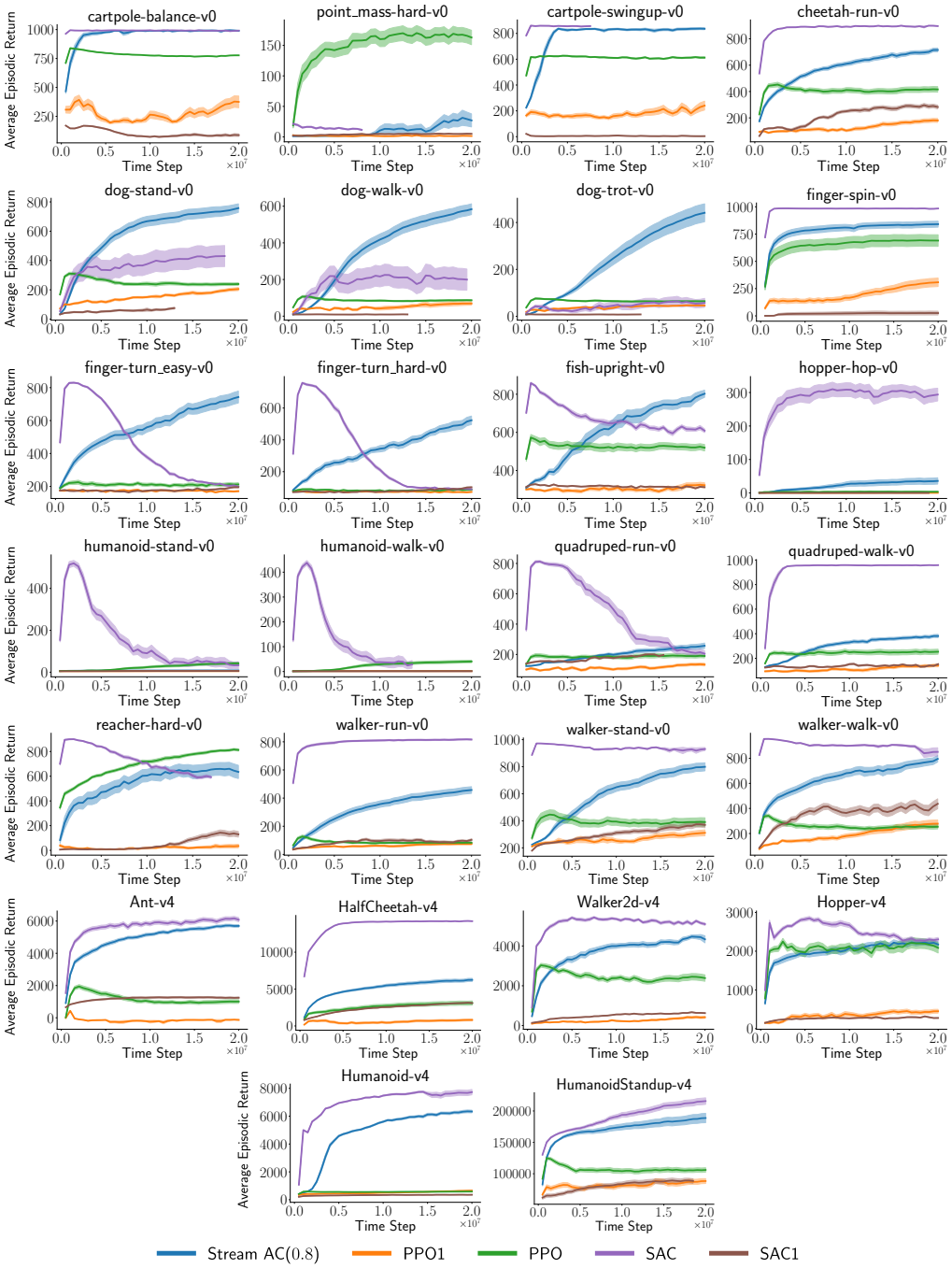



Figure 14: Performance of stream AC in DMC and MuJoCo environments. The results are averaged over 30 independent runs. The shaded area represents a 90% confidence interval.