Streaming Reinforcement Learning under Partial Observability with Real-Time Recurrent Learning

Noah Farr[1,5,*], **Aryaman Reddi**[1,3,*], **Carlo D'Eramo**[2], **Jan Peters**[1,3,4]
noah.farr@stud.tu-darmstadt.de aryaman.reddi@tu-darmstadt.de

[1]**Technical University of Darmstadt**
[2]**Center for Artificial Intelligence and Data Science, University of Würzburg**
[3]**Hessian.AI**
[4]**German Research Center for AI (DFKI)**
[5]**Zuse School ELIZA**

[*] Equal contribution.

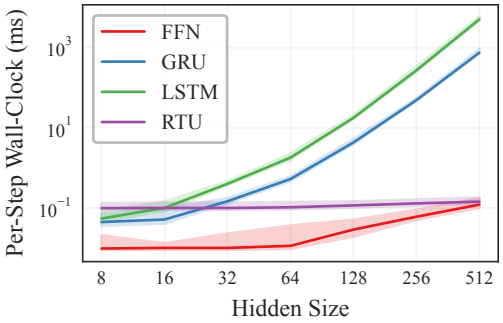
Abstract

Streaming reinforcement learning has emerged as an online learning paradigm that conforms to the restrictions of natural learning agents that process data incrementally, i.e. with a batch size of 1 and no replay buffer. While streaming RL has recently been shown to scale with deep function approximation with full observability, partially observable settings have remained out of reach. Truncated backpropagation through time collapses to a one-step gradient horizon under the streaming setting, and exact real-time recurrent learning is prohibitively expensive. We close this gap using recurrent trace units, a diagonal recurrent architecture that enables exact RTRL with linear time and memory complexity in the parameter count, and show that they integrate cleanly into existing streaming algorithms across both discrete and continuous control. On a MemoryChain diagnostic with chain lengths from 2 to 128, our method sustains performance where streaming TBPTT(1) baselines using feedforward, GRU, and RTU networks collapse. On five POPGym tasks and on partially observable MuJoCo continuous control, the streaming approach is competitive with batched PPO on POPGym and recovers a substantial fraction of batched performance on masked MuJoCo, despite using no replay buffer or batched updates.

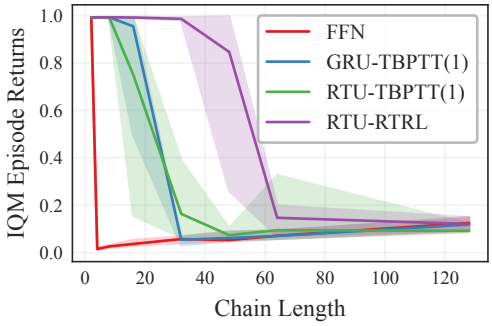
1 Introduction

Streaming reinforcement learning, where an agent updates from each sample as it arrives without a replay buffer, has recently been shown to scale to deep function approximation in fully observable environments using eligibility traces (Elsayed et al., 2024; Vasan et al., 2024). Extending streaming methods to partially observable settings, which are more representative of realistic deployment scenarios, has remained out of reach. Under single-sample update settings, the only feasible form of recurrent learning with backpropagation through time is with one-step truncation, denoted TBPTT(1), which provides only a one-step gradient horizon.

Real-Time Recurrent Learning (RTRL) (Williams & Zipser, 1989) maintains recurrent forward-mode gradients incrementally and is the natural fit for streaming, but for general recurrent architectures it has prohibitive computational cost that scales quartically in the hidden dimension. Recurrent Trace Units (RTU) (Elelimy et al., 2024), a recently proposed diagonal complex-valued recurrent architecture, admit exact RTRL with linear time and memory complexity in the parameter



(a) Per-step wall-clock time (ms) of a single gradient update as a function of hidden width, comparing feedforward backpropagation (FFN) against exact RTRL for Gated Recurrent Unit (GRU), LSTM, and RTU recurrent layers.



(b) Episodic return as a function of chain length on the MemoryChain diagnostic, averaged over 5 seeds. We compare a feedforward network, GRU and RTU with TBPTT over 1 step and RTU with RTRL.

count, but have so far only been used with batched algorithms such as Proximal Policy Optimization (PPO) (Schulman et al., 2017). Bringing these two threads together, we show that recurrent RTRL traces with RTUs compose naturally with the eligibility traces of the streaming update, yielding a single-pass procedure that propagates credit through both time and the recurrent state without truncation.

We evaluate the approach on four benchmarks chosen to isolate distinct properties. A MemoryChain diagnostic with chain lengths from 2 to 128 isolates the challenge of credit assignment over time, where baselines using feedforward, GRU (Cho et al., 2014), and RTU architectures all collapse past 16 steps while streaming RTRL with RTUs sustains performance until 64 steps (Figure 1b). On five tasks from POPGym (Morad et al., 2023), the streaming methods match batched PPO, with the two streaming variants jointly solving all five. On partially observable MuJoCo with masked positions or velocities (Ni et al., 2022), the streaming methods extend to continuous control but trail batched PPO, recovering a substantial fraction of its performance. On a KMemoryChain task we measure and quantify the staleness of RTRL sensitivities. We do not claim state-of-the-art performance on any single benchmark, as our contribution is closing the partial observability gap in streaming deep RL with a general method.

Contributions.

- A streaming deep RL algorithm for partially observable environments using RTUs with RTRL, requiring no replay buffer or batched updates.
- A MemoryChain length sweep showing that this combination sustains credit assignment past streaming TBPTT(1) baselines using feedforward, GRU, and RTU recurrence.
- Empirical evidence on five POPGym memory tasks and partially observable MuJoCo continuous control that the streaming approach is competitive with batched PPO.
- A staleness analysis of the RTRL sensitivity matrix under streaming updates, bounding the error in steady state and identifying a first-order Taylor correction that tightens the bound.

2 Background

Streaming reinforcement learning with eligibility traces. A streaming agent receives an observation and reward at each time step, takes an action, performs a learning update from that single sample, and then discards it. No replay buffer is maintained. To propagate credit across time without storing past data, streaming algorithms maintain eligibility traces (Sutton & Barto, 1998). For a per-step gradient $\boldsymbol{g}_t$ of a value function or policy with respect to parameters $\boldsymbol{w}_t$ at time t, these

methods iterate as follows:

$$\boldsymbol{z}_0 \doteq \mathbf{0} \tag{1}$$

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \boldsymbol{g}_t, \tag{2}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\delta_t\boldsymbol{z}_t, \tag{3}$$

where δ_t is the temporal difference (TD) error, α is the step size, and $\lambda \in [0, 1]$ is the trace-decay parameter. The total parameter change generated by these updates is equivalent to that of the λ-return in the offline case, without storing a multi-step rollout. Existing streaming deep RL methods (Elsayed et al., 2024; Vasan et al., 2024) use this method with feedforward function approximation, where $\boldsymbol{g}_t$ is the output of standard backpropagation through a single sample.

Real-time recurrent learning. For a recurrent network $\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\psi})$ with parameters $\boldsymbol{\psi}$, computing $\boldsymbol{g}_t$ requires the gradient $\partial\boldsymbol{h}_t/\partial\boldsymbol{\psi}$ of the current hidden state with respect to the recurrent parameters. Backpropagation through time (BPTT) obtains this gradient by unrolling the recurrence backward over a stored trajectory of length T, which reduces to a one-step gradient horizon when $T = 1$. RTRL (Williams & Zipser, 1989) avoids this unrolling by maintaining the gradient forward through time, applying the chain rule to the recurrence,

$$\frac{\partial\boldsymbol{h}_t}{\partial\boldsymbol{\psi}} = \frac{\partial\boldsymbol{f}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\psi})}{\partial\boldsymbol{\psi}} + \frac{\partial\boldsymbol{f}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\psi})}{\partial\boldsymbol{h}_{t-1}}\,\frac{\partial\boldsymbol{h}_{t-1}}{\partial\boldsymbol{\psi}}. \tag{4}$$

The first term is the direct dependence of $\boldsymbol{h}_t$ on $\boldsymbol{\psi}$ through the current step. The second propagates accumulated dependence through $\boldsymbol{h}_{t-1}$. For a general recurrent network with hidden dimension n and parameter count $O(n^2)$, this gradient is an $n \times O(n^2)$ matrix and the recursion costs $O(n^4)$ per step, prohibitive even for moderate n.

Recurrent trace units. RTUs (Elelimy et al., 2024) sidestep this barrier with a diagonal complex-valued recurrence,

$$\boldsymbol{h}_t = \boldsymbol{\Lambda}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t, \tag{5}$$

where $\boldsymbol{\Lambda} \in \mathbb{C}^{n\times n}$ is diagonal with entries $\lambda_k = r_k(\cos\Omega_k + i\sin\Omega_k)$, where r_k and Ω_k define the polar complex representation of the k'th parameter. Because each hidden unit has only one recurrent connection to itself, exact RTRL has linear time and memory complexity in the number of parameters. Figure 1a confirms this empirically. Per-step wall-clock time for RTU under exact RTRL tracks feedforward backpropagation as hidden width grows, while GRU and LSTM under RTRL exhibit the prohibitive $O(n^4)$ scaling.

3 Methodology

Our method makes a single change to existing streaming algorithms. Where an algorithm typically uses a feedforward function approximator mapping observations to values or actions, we insert an RTU layer between the observation and the feedforward head. The streaming update machinery, including eligibility traces and step-size adaptation, is unchanged. The only modification is how the per-step gradient that feeds into the eligibility trace is computed. We describe this gradient computation for a single network below.

Per-step gradient. The function approximator has three parameter groups: an encoder $\phi(\,\cdot\,;\boldsymbol{\theta})$ producing features $\boldsymbol{x}_t$, a recurrent RTU layer with parameters $\boldsymbol{\psi}$ producing the state $\boldsymbol{h}_t$, and a feedforward head with parameters $\boldsymbol{w}$ (e.g. a value head $v(\boldsymbol{h}_t; \boldsymbol{w}_v)$). We compute the gradient of the head output with respect to each group in a single backward pass, combined with the RTU's forward-mode RTRL trace $\boldsymbol{S}_t = \partial\boldsymbol{h}_t/\partial\boldsymbol{\psi}$.

The head parameters have no temporal dependency and receive an ordinary backward pass. The recurrent parameters enter only through $\boldsymbol{h}_t$, so combining the backward pass through the head with

$\boldsymbol{S}_t$ yields their gradient across the entire history at $\mathcal{O}(|\boldsymbol{\psi}|)$ per step. The encoder parameters are more complicated. They influence the state both instantaneously through $\boldsymbol{x}_t$ and recursively through every past state. Maintaining the exact trace would cost $\mathcal{O}(|\boldsymbol{\theta}| \cdot \dim \boldsymbol{h})$ per step, prohibitive for a deep encoder. We therefore use a one-step approximation, backpropagating through the head, a single step of the recurrence, and the encoder. This credits the encoder for its immediate effect on $\boldsymbol{h}_t$ while ignoring its influence through past states.

The same decomposition applies to any other head (policy, auxiliary). Each network carries its own RTU layer and RTRL trace, so recurrent parameters are updated independently across heads.

Composition with eligibility traces. The RTRL trace and the eligibility trace coexist without interacting. The RTRL trace is a forward-mode object that maintains $\partial \boldsymbol{h}_t / \partial \boldsymbol{\psi}$ at each step, with no notion of credit assignment over time. The eligibility trace is a temporal accumulator of whatever per-step gradient the streaming algorithm uses, and accumulates the recurrent contribution as it accumulates the head contribution. Operationally, each RTU layer supplies its $\partial \boldsymbol{h}_t / \partial \boldsymbol{\psi}$ on demand whenever the streaming update needs the gradient with respect to its recurrent parameters, and the eligibility trace then folds this into its usual recursion.

Instantiations. We evaluate this method on two streaming algorithms covering both value-based and policy-based control. We use QRC(λ) (Elelimy et al., 2025) for value-based control and stream AC(λ) (Elsayed et al., 2024) for policy-based control, applying QRC(λ) to MemoryChain, QRC(λ) and stream AC(λ) to POPGym as well as KMemoryChain, and stream AC(λ) to masked MuJoCo. The method itself is independent of this choice and applies to any streaming algorithm whose update is driven by per-step gradients. In both cases each network has its own RTU layer with its own RTRL trace, so the recurrent parameters of each network are updated independently. Complete pseudocode for both instantiations is given in Appendix B.

4 Experiments

We evaluate the method on four benchmarks chosen to isolate distinct properties. MemoryChain (Section 4.1) sweeps a memory horizon to measure how far credit propagates through a streaming update. POPGym (Section 4.2) tests whether the streaming method is competitive with a batched recurrent baseline on long-memory discrete control. Masked MuJoCo (Section 4.3) extends the evaluation to continuous control under partial observability. KMemoryChain (Section 4.4) sweeps a memory length to measure the staleness of sensitivities in RTRL. All curves report the interquartile mean (IQM) over 5 seeds with shaded standard error. All experiments are implemented in Memorax (Farr et al., 2025), an open-source framework for memory-augmented RL. Hyperparameters, network architectures, and per-task details are listed in Appendix A.

4.1 MemoryChain

Task and design. The MemoryChain (Osband et al., 2020) task presents an informative binary cue at $t{=}0$. The agent then receives uninformative observations for $L{-}1$ steps and is rewarded for reproducing the cue at $t{=}L$. The chain length L controls how far back the agent must propagate credit, isolating the temporal credit-assignment problem from representation learning. We sweep $L \in \{2, 4, 8, 16, 32, 48, 64, 128\}$. The streaming algorithm is QRC(λ) (Elelimy et al., 2025), and we compare four agents. The first is a feedforward network without recurrence (FFN). The second is GRU with TBPTT(1). The third is RTU with TBPTT(1). The fourth is RTU with exact RTRL. All non-recurrence hyperparameters are matched across agents.

Result. Figure 1b reports IQM episodic return against L. The three TBPTT(1) curves drop to near-zero return past $L{=}16$ and stay there for the remainder of the sweep. A one-step gradient horizon struggles to carry credit through even moderately long chains, and substituting RTU for GRU under TBPTT(1) does not change this. The RTU-RTRL curve holds near the maximum return

through L=48 and degrades through L=64. The gap between the dotted and solid RTU curves attributes this improvement to the gradient method rather than the architecture, since both use the same recurrence and the same streaming update with only the computation of $\partial \boldsymbol{h}_t / \partial \boldsymbol{\psi}$ differing. RTRL is therefore the operative ingredient under streaming, and a diagonal recurrence alone is not sufficient.

4.2 POPGym

Task and design. We evaluate on five memory tasks from POPGym (Morad et al., 2023). These are Autoencode, Concentration, CountRecall, HigherLower, and RepeatFirst. We compare stream AC(λ)-RTU and QRC(λ)-RTU against a batched PPO-RTU baseline and an FFN baseline. The cell of interest is RTU with QRC(λ) and stream AC(λ). RTU with batched PPO is the comparable baseline that uses the same architecture without the streaming constraint. Each agent is run for 5M environment frames.

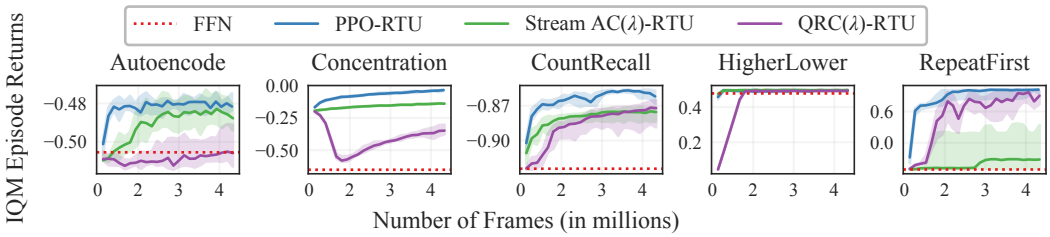


Figure 2: IQM episodic return over 5 seeds with shaded standard error on five POPGym memory tasks.

Result. Across five POPGym tasks, QRC(λ)-RTU and stream AC(λ)-RTU each outperform the FFN baseline on four of five tasks, isolating the contribution of recurrent state under the streaming constraint. The two methods are complementary across the task set: stream AC(λ)-RTU wins on Autoencode, Concentration, and CountRecall, while QRC(λ)-RTU wins by a wide margin on RepeatFirst, the one task where stream AC(λ)-RTU fails to learn. Compared against batched PPO-RTU, which relaxes the streaming constraint by maintaining a replay buffer, QRC(λ)-RTU matches PPO on HigherLower and RepeatFirst, trails it by a small margin on CountRecall, and fails to learn on Autoencode and Concentration—precisely the tasks where stream AC(λ)-RTU succeeds. Taken together, the two streaming methods solve all five tasks, recovering batched performance without storing past trajectories.

4.3 Masked MuJoCo

Task and design. Following Ni et al. (2022), we evaluate on four MuJoCo locomotion tasks of Ant, HalfCheetah, Hopper, Walker2d and under two observation regimes masked velocities (**P**) and masked positions (**V**). Both settings have partially-observable observations, requiring the agent to infer the missing component from history. We compare a stream AC(λ) agent with RTU against PPO with RTU. Each run uses 2M environment frames.

Result. Across the eight task-regime cells, stream AC(λ)-RTU matches batched PPO-RTU on HalfCheetah-P, overtakes it on Walker2d-V, fails to learn on Walker2d-P, and trails it by varying margins on the remaining five. HalfCheetah-P is essentially tied, both curves rise together and finish indistinguishably near 2,000. Walker2d-V is the reversal, where stream AC(λ) rises sharply within the first 0.5M frames to roughly 800 and holds that level while PPO climbs more slowly and finishes below it. Walker2d-P is the only failure case where stream AC(λ) remains near its initial return throughout training while PPO reaches roughly 1,000. On the remaining five cells both methods learn, with stream AC(λ) trailing PPO by a modest margin on Hopper in both regimes, and by a

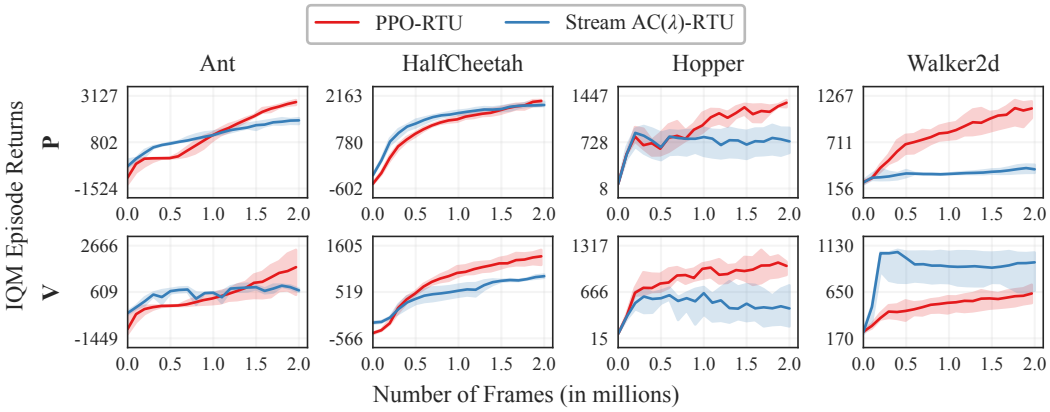


Figure 3: IQM episodic return over 5 seeds with shaded standard error on four MuJoCo tasks under two observation regimes.

wider margin on Ant in both regimes and on HalfCheetah-V, where stream AC(λ) plateaus while PPO continues to improve. Partially observable continuous control is therefore feasible under the streaming constraint, but the gap to batched PPO is wider than on POPGym, indicating that the streaming method recovers a meaningful fraction, though not all, of batched performance in this regime.

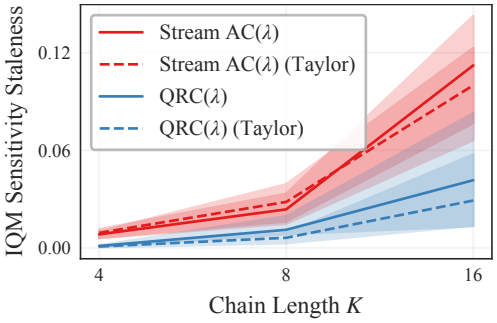
4.4 RTRL Staleness

The RTRL sensitivity matrix $\boldsymbol{S}_t = \partial \boldsymbol{h}_t / \partial \boldsymbol{\psi}$ is built up by accumulating immediate Jacobians $\boldsymbol{I}_k(\boldsymbol{\psi}_k)$ injected at past steps and propagated forward by the state-to-state Jacobian $\boldsymbol{J}_t$. Under a streaming update the parameters $\boldsymbol{\psi}$ move every step, so each $\boldsymbol{I}_k(\boldsymbol{\psi}_k)$ in this sum was evaluated at parameters the optimizer has since left behind. The maintained sensitivity is therefore a biased estimate of the ideal sensitivity $\boldsymbol{S}_t^*$, which re-evaluates every historical immediate Jacobian at the current parameters $\boldsymbol{\psi}_t$.

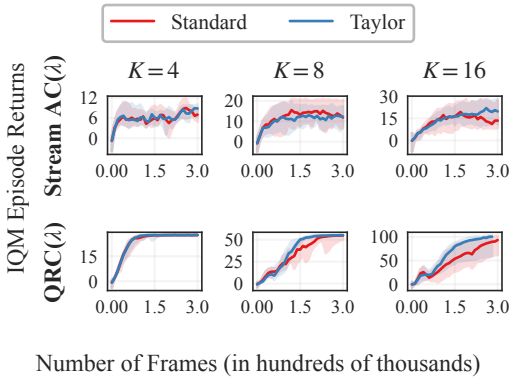
RTRL staleness has been observed qualitatively in prior work (Williams & Zipser, 1989; Menick et al., 2021; Irie et al., 2024; Elelimy et al., 2024), with smaller learning rates (Menick et al., 2021) or periodic recomputation of the trace over the trajectory (Elelimy et al., 2024) suggested as mitigations, but it has not been quantitatively bounded.

Appendix C bounds this staleness in steady state and shows that a first-order Taylor correction in $\boldsymbol{\psi}$ reduces the error from $\mathcal{O}(\eta)$ to $\mathcal{O}(\eta^2)$. This section probes the bias empirically and tests whether the correction tracks the bound under sustained credit-assignment load.

Task and design. We use KMemoryChain, an every-step variant of MemoryChain designed to exercise the RTRL trace continuously. At each step the agent observes a fresh ± 1 bit together with a time-remaining feature, and is rewarded ± 1 for predicting the bit it saw K steps ago. Rewards activate after a K-step warmup. The classic MemoryChain rewards a single recall at episode end and leaves the immediate Jacobian inactive for most of the rollout, which is precisely the regime in which staleness has the least opportunity to accumulate. KMemoryChain instead injects a new bit and emits a reward-relevant target at every step, keeping the immediate Jacobian non-zero throughout the rollout and making K a direct dial on the credit-assignment horizon over which the sensitivity must remain accurate. We sweep $K \in \{4, 8, 16\}$ and run both streaming algorithms, QRC(λ) and stream AC(λ). Each algorithm is run with the standard RTRL sensitivity and with the first-order Taylor correction of Appendix C enabled, giving four configurations per K.



(a) RTRL sensitivity staleness on KMemoryChain. Relative ℓ_2 distance between the online sensitivity matrix $\tilde{\boldsymbol{S}}_t$ and its recomputed reference $\boldsymbol{S}_t^*$, $\|\tilde{\boldsymbol{S}}_t - \boldsymbol{S}_t^*\|_2 / \|\boldsymbol{S}_t^*\|_2$, over different chain lengths.



(b) IQM episodic return over 5 seeds with shaded standard error on different chain lengths of KMemoryChain.

Staleness metric. At every training step we maintain a buffer of the observations from the current episode. To form a reference, we replay the buffer forward from the saved initial state using the *current* parameters $\boldsymbol{\psi}_t$, recomputing what the sensitivity would be if every immediate Jacobian in the episode had been evaluated at $\boldsymbol{\psi}_t$ rather than at the parameters in force when it was originally injected. The staleness reported is the ℓ_2 distance between the live sensitivity carried by the algorithm and this recomputed reference.

Result. Figure 4a reports the IQM sensitivity staleness on KMemoryChain as a function of the memory credit-assignment horizon K. Staleness rises with K across all four configurations, remaining near zero at $K{=}4$ and growing through $K{=}8$ to its peak at $K{=}16$. This K-dependence is consistent with the $(1-\rho)^{-2}$ scaling of the steady-state bound in Equation (11): longer credit-assignment horizons require the recurrence to retain memory over more steps, raising the effective contraction constant and amplifying the bound.

The two algorithms differ in absolute staleness level: stream AC(λ) staleness is consistently higher than QRC(λ) staleness and grows more steeply with K. We attribute this gap to the optimizer rather than the algorithm. Stream AC(λ) uses ObGD (Elsayed et al., 2024) with base step size $\alpha = 1$, which is scaled down only when the effective-step-size threshold is violated and otherwise drives large per-step parameter updates. QRC(λ), by contrast, uses plain SGD with fixed learning rates of 10^{-4} for the Q-network and 10^{-5} for the auxiliary h-network, two to three orders of magnitude smaller than the nominal ObGD step. Because the steady-state bound in Equation (11) scales linearly with the per-step parameter update magnitude ηG, the larger ObGD updates translate directly into proportionally larger steady-state staleness. The Taylor correction reduces staleness across both algorithms, with the largest absolute reduction at $K{=}16$ where the uncorrected staleness is largest and the leading-order term predicted by Equation (10) dominates. At $K{=}4$ the corrected and uncorrected variants are essentially indistinguishable, leaving little headroom for a first-order correction to act on.

Figure 4b reports the corresponding IQM episodic returns across training for each $(\text{algorithm}, K)$ cell. The Taylor correction does not degrade learning at any K, and at $K{=}4$ and $K{=}8$ it tracks the standard sensitivity throughout training. At $K{=}16$, where the standard sensitivity incurs the largest staleness, the Taylor correction appears to learn faster and produce a more stable training curve, providing downstream evidence that the staleness reductions in Figure 4a translate into improved credit assignment in the regime where the bound is tightest. Taken together, streaming RTRL staleness stays bounded under sustained credit-assignment load, scales with K in the direction the

bound predicts, and tightens under the Taylor correction in the regime where the leading-order term dominates.

5 Related Work

Streaming deep reinforcement learning. Stream-x (Elsayed et al., 2024) stabilized value-based and actor-critic streaming methods in deep RL through a combination of step-size adaptation, layer normalization, and observation and reward scaling, removing the long-standing argument that streaming sacrifices sample efficiency. AVG (Vasan et al., 2024) obtained a similar result for policy gradient methods using a reparameterized policy. Both works restrict to fully observable environments. Our method extends this line to partial observability while preserving the no-replay, constant-memory streaming property that motivates it.

Practical real-time recurrent learning. RTRL was introduced as the natural online counterpart to backpropagation through time (Williams & Zipser, 1989), but its quartic cost in the hidden dimension has long limited its use. Recent work makes RTRL efficient by restricting the recurrence. Menick et al. (2021) sparsify the influence matrix to make RTRL tractable for general recurrent cells and observe that online weight updates introduce a staleness bias in the maintained sensitivity. Linear recurrent units (Orvieto et al., 2023) use a diagonal complex-valued recurrence trained by parallel scan, and RTUs (Elelimy et al., 2024) build on this foundation with a real-valued cosine parameterization that fits cleanly into deep RL pipelines. Irie et al. (2024) apply exact RTRL with element-wise recurrent networks to actor-critic RL and discuss staleness as a practical limitation. RTUs have so far been combined only with batched algorithms such as PPO. We use them as the recurrent component of a streaming algorithm, leveraging the linear-time RTRL trace they admit and quantify the staleness that streaming updates induce in Appendix C.

Recurrent RL under partial observability. Recurrent model-free RL with GRU or Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) backbones trained via truncated BPTT is a strong baseline for POMDPs when a replay buffer is available (Ni et al., 2022). POPGym (Morad et al., 2023) provides systematic benchmarking across recurrent architectures and supplies the long-memory tasks we use. Our method matches the structural assumption of these recurrent baselines (model-free, recurrent state for memory) but removes the dependence on stored trajectories.

6 Discussion and Limitations

Restricted recurrent architecture. We use a single diagonal RTU layer per network, because RTUs enable exact RTRL in linear time and because multi-layer RTRL remains an open challenge. Even with diagonal recurrence at each layer, the gradient trace between layers requires either approximation or persisting cross-layer Jacobians. Elelimy et al. (2024) show that diagonal recurrence does not hurt on partially observable tasks, which we also observe. Partially observable tasks requiring richer representations remain to be tested.

Scope of the empirical claim. We do not claim state-of-the-art absolute performance on any single benchmark. The contribution is closing the streaming-plus-partial-observability gap, and we compare against methods that respect the same resource constraints. On benchmarks where unconstrained batched methods are available, those methods may still produce better sample efficiency per environment step. The case for the streaming method is the regime where the constraints are real, such as on-device learning, real-time control, and lifelong learning settings where storing past trajectories is impractical.

7 Conclusion

We present a streaming deep reinforcement learning method for partially observable environments. An RTU layer trained by exact real-time recurrent learning is inserted between the observation and the feedforward heads of an existing streaming algorithm, leaving the streaming update machinery unchanged. The RTRL trace and the streaming eligibility trace compose without modification, yielding a single-pass procedure that propagates credit through both the recurrent state and time. Empirically, the method sustains credit assignment on MemoryChain chain lengths well past streaming TBPTT(1) baselines, jointly solves all five POPGym memory tasks at parity with batched PPO across the two streaming variants, and extends to continuous control on masked MuJoCo at a meaningful performance level. An analysis of the RTRL sensitivity matrix bounds its staleness under streaming updates and identifies a first-order Taylor correction that tightens the bound.

A Hyperparameters

We used fully connected networks of width 64 for MemoryChain, POPGym and Masked MuJoCo, with LeakyReLU activations on the discrete-action benchmarks and $\tanh$ on Masked MuJoCo, and LayerNorm (Ba et al., 2016) before each activation. A single RTU layer of hidden dimension 192 is inserted between the input and output blocks. We used separate networks for the policy and value functions. For stream AC(λ) we used softmax policy parameterization in discrete control and a state-independent Gaussian with actions clipped to $[-1, 1]$ in continuous control. For QRC(λ) we used an ε-greedy policy. We used sparse initialization with a sparsity ratio of 90% (Elsayed et al., 2024) and normalized observations and rewards online with running statistics. stream AC(λ) uses ObGD with step size $\alpha = 1$; the per-benchmark κ values for the policy and value networks are listed in the experiment-specific tables below. QRC(λ) uses SGD with global-norm gradient clipping at 1.0 and learning rates 10^{-4} for the Q-network and 10^{-5} for the auxiliary h-network. For the batched PPO baseline we used the hyperparameters reported by Elelimy et al. (2024).

Table 1: Hyperparameters for QRC(λ) on the MemoryChain experiment (Section 4.1).

Name	Value
Chain length, L	$\{2, 4, 8, 16, 32, 48, 64, 128\}$
Frames per run	500,000
Number of seeds	5
Discount factor, γ	0.99
Trace decay, λ	0.95
Q-network step size, α_Q	10^{-4}
h-network step size, α_h	10^{-5}
Regularization coefficient, β	1.0
MLP width	64
RTU hidden dimension	192

Table 2: Hyperparameters for QRC(λ) on POPGym (Section 4.2).

Name	Value
Frames per run	5,000,000
Number of seeds	5
Discount factor, γ	0.99
Trace decay, λ	0.95
Q-network step size, α_Q	10^{-4}
h-network step size, α_h	10^{-5}
Regularization coefficient, β	1.0
ε schedule	$1.0 \rightarrow 0.01$ over 20% of training
MLP width	64
RTU hidden dimension	192

Table 3: Hyperparameters for stream AC(λ) on POPGym (Section 4.2).

Name	Value
Frames per run	5,000,000
Number of seeds	5
Discount factor, γ	0.99
Trace decay, λ	0.8
Entropy coefficient	0.095
Optimizer	ObGD
Actor step size, α_π	1.0
Critic step size, α_V	1.0
Actor κ_π	3.0
Critic κ_V	2.0
MLP width	64
RTU hidden dimension	192

Table 4: Hyperparameters for stream AC(λ) on the Masked MuJoCo experiment (Section 4.3).

Name	Value
Frames per run	2,000,000
Number of seeds	5
Observation regimes	masked velocities (**P**), masked positions (**V**)
Discount factor, γ	0.99
Trace decay, λ	0.8
Entropy coefficient	0.01
Optimizer	ObGD
Actor step size, α_π	1.0
Critic step size, α_V	1.0
Actor κ_π	1.5
Critic κ_V	1.0
MLP width	64
RTU hidden dimension	192

Table 5: Hyperparameters for stream AC(λ) on the KMemoryChain experiment (Section 4.4).

Name	Value
Memory length, K	$\{4, 8, 16\}$
Frames per run	300,000
Number of seeds	5
Discount factor, γ	0.99
Trace decay, λ	0.95
Actor step size, α_π	1.0
Critic step size, α_v	1.0
Actor ObGD κ_π	3
Critic ObGD κ_v	2.0
Entropy coefficient	0.01
RTU feature dimension	64
RTU hidden dimension	192

Table 6: Hyperparameters for QRC(λ) on the KMemoryChain experiment (Section 4.4).

Name	Value
Memory length, K	$\{4, 8, 16\}$
Frames per run	300,000
Number of seeds	5
Discount factor, γ	0.99
Trace decay, λ	0.95
Q-network step size, α_Q	10^{-4}
h-network step size, α_h	10^{-5}
Regularization coefficient, β	1.0
ϵ-greedy start	1.0
ϵ-greedy end	0.01
ϵ-greedy decay fraction	0.1
RTU feature dimension	64
RTU hidden dimension	192

B Pseudocode

The contribution to either parent algorithm is the per-step gradient computation: each network maintains an RTU hidden state $\boldsymbol{h}$ and a forward-mode RTRL sensitivity matrix $S = \partial \boldsymbol{h} / \partial \boldsymbol{\psi}$, both advanced once per environment step, and gradients of any scalar depending on $\boldsymbol{h}$ are computed from S. Following the parent algorithms, $\boldsymbol{\theta}$, $\boldsymbol{\psi}$ and $\boldsymbol{w}$ denote each network's full parameter set (encoder, RTU, head). Trace recursions and update rules are unchanged from the parents.

Algorithm 1 QRC(λ) with RTU-RTRL

Input: a differentiable action-value function parametrization $\hat{q}_{\boldsymbol{w}}$ with an RTU layer
Input: a differentiable auxiliary function parametrization $\hat{h}_{\boldsymbol{\theta}}$ with an RTU layer
Algorithm parameters: learning rate α_q, h learning rate α_h, exploration parameter ϵ.
Initialize $\boldsymbol{z}_t^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$
Initialize $\boldsymbol{z}_t^{\boldsymbol{w}} \leftarrow \mathbf{0}$
Initialize $z_t^h \leftarrow 0$
Initialize RTU carries $\boldsymbol{h}_t^q, \boldsymbol{h}_t^h \leftarrow \mathbf{0}$ and RTRL traces $\boldsymbol{S}_t^q, \boldsymbol{S}_t^h \leftarrow \mathbf{0}$
Observe initial state $\boldsymbol{S}_0$
for iteration $t = 1, 2, \cdots$ **do**
  Advance $(\boldsymbol{h}_t^q, \boldsymbol{S}_t^q)$ and $(\boldsymbol{h}_t^h, \boldsymbol{S}_t^h)$ one RTU step on $\boldsymbol{S}_t$.
  Sample an action $A_t \sim \pi$. ▷ We use an ϵ-greedy policy.
  Take action A_t, observe R_{t+1} and S_{t+1}.
  Compute δ_t and $\nabla_{\boldsymbol{w}_t} \delta_t$ as in QRC(λ) (Elelimy et al., 2025), with parameter gradients of $\hat{q}_{\boldsymbol{w}}$ obtained from the RTRL trace $\boldsymbol{S}_t^q$.
  Update the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$, $\boldsymbol{z}_t^{\boldsymbol{w}}$, and z_t^h as in QRC(λ) (Elelimy et al., 2025), with parameter gradients of $\hat{h}_{\boldsymbol{\theta}}$ obtained from the RTRL trace $\boldsymbol{S}_t^h$.
  Compute $\Delta \boldsymbol{w}_t$ and $\Delta \boldsymbol{\theta}_t$ as in QRC(λ) (Elelimy et al., 2025).
  Update the parameters $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha_q \Delta \boldsymbol{w}_t$.
  Update the parameters $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_h \Delta \boldsymbol{\theta}_t$.
  if episode terminated or A_t is non-greedy **then**
    Reset the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$, $\boldsymbol{z}_t^{\boldsymbol{w}}$, and z_t^h to zeros.
  end if
  if episode terminated **then**
    Reset the RTU carries $\boldsymbol{h}_t^q, \boldsymbol{h}_t^h$ and RTRL traces $\boldsymbol{S}_t^q, \boldsymbol{S}_t^h$ to zeros.
  end if
end for

Algorithm 2 stream AC(λ) with RTU-RTRL

Input: a differentiable policy parametrization $\pi(a \mid s; \boldsymbol{\theta})$ with an RTU layer
Input: a differentiable state-value function parametrization $\hat{v}(s; \boldsymbol{w})$ with an RTU layer
Algorithm parameters: discount γ, trace decay λ, policy step size α_π, value step size $\alpha_{\hat{v}}$, policy and value scaling factors $\kappa_\pi, \kappa_{\hat{v}}$, entropy coefficient τ.
Initialize $\boldsymbol{z}_t^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$
Initialize $\boldsymbol{z}_t^{\boldsymbol{w}} \leftarrow \mathbf{0}$
Initialize RTU carries $\boldsymbol{h}_t^\pi, \boldsymbol{h}_t^v \leftarrow \mathbf{0}$ and RTRL traces $\boldsymbol{S}_t^\pi, \boldsymbol{S}_t^v \leftarrow \mathbf{0}$
Initialize observation- and reward-normalization statistics as in stream AC(λ) (Elsayed et al., 2024)
Observe initial state $\boldsymbol{S}_0$
for iteration $t = 1, 2, \cdots$ **do**
  Advance $(\boldsymbol{h}_t^\pi, \boldsymbol{S}_t^\pi)$ and $(\boldsymbol{h}_t^v, \boldsymbol{S}_t^v)$ one RTU step on $\boldsymbol{S}_t$.
  Sample an action $A_t \sim \pi(\cdot \mid \boldsymbol{S}_t, \boldsymbol{\theta}_t)$.
  Take action A_t, observe R_{t+1} and S_{t+1}.
  Compute δ_t and the per-step gradients $\nabla_{\boldsymbol{w}} \hat{v}(\boldsymbol{S}_t, \boldsymbol{w}_t)$ and $\nabla_{\boldsymbol{\theta}}\big(\log \pi(A_t \mid \boldsymbol{S}_t, \boldsymbol{\theta}_t) + \tau \operatorname{sign}(\delta_t)\, \mathcal{H}(\cdot \mid \boldsymbol{S}_t, \boldsymbol{\theta}_t)\big)$ as in stream AC(λ) (Elsayed et al., 2024), with parameter gradients of $\hat{v}$ and π obtained from the RTRL traces $\boldsymbol{S}_t^v$ and $\boldsymbol{S}_t^\pi$.
  Update the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$ and $\boldsymbol{z}_t^{\boldsymbol{w}}$ as in stream AC(λ) (Elsayed et al., 2024).
  Compute $\Delta\boldsymbol{\theta}_t$ and $\Delta\boldsymbol{w}_t$ via ObGD as in stream AC(λ) (Elsayed et al., 2024).
  Update the parameters $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t$.
  Update the parameters $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \Delta\boldsymbol{w}_t$.
  if episode terminated **then**
    Reset the traces $\boldsymbol{z}_t^{\boldsymbol{\theta}}$ and $\boldsymbol{z}_t^{\boldsymbol{w}}$ to zeros.
    Reset the RTU carries $\boldsymbol{h}_t^\pi, \boldsymbol{h}_t^v$ and RTRL traces $\boldsymbol{S}_t^\pi, \boldsymbol{S}_t^v$ to zeros.
  end if
end for

C Staleness error bound for the RTRL sensitivity matrix

In the nonlinear setting, the RTRL sensitivity matrix maintained by an RTU layer incurs a *staleness error*: immediate Jacobians accumulated at earlier steps were evaluated at parameters that the optimizer has since moved away from. We bound this error and give a second-order correction that reduces it from $\mathcal{O}(\eta)$ to $\mathcal{O}(\eta^2)$.

C.1 Setup and assumptions

For a recurrent network $\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\psi})$ with state-to-state Jacobian $\boldsymbol{J}_t = \partial \boldsymbol{h}_t / \partial \boldsymbol{h}_{t-1}$ and immediate Jacobian $\boldsymbol{I}_t(\boldsymbol{\psi}) = \partial \boldsymbol{h}_t / \partial \boldsymbol{\psi}$, RTRL maintains the sensitivity matrix

$$\boldsymbol{S}_t \;=\; \boldsymbol{J}_t\, \boldsymbol{S}_{t-1} \;+\; \boldsymbol{I}_t. \tag{6}$$

Let $\boldsymbol{\psi}_t \in \mathbb{R}^p$ denote the recurrent parameters in force at step t. The *stale sensitivity matrix* maintained by the online algorithm is

$$\tilde{\boldsymbol{S}}_t \;=\; \boldsymbol{J}_t\, \tilde{\boldsymbol{S}}_{t-1} \;+\; \boldsymbol{I}_t(\boldsymbol{\psi}_t) \;=\; \sum_{k=0}^{t} \left(\prod_{j=k+1}^{t} \boldsymbol{J}_j \right) \boldsymbol{I}_k(\boldsymbol{\psi}_k), \tag{7}$$

where the immediate Jacobian injected at step k was evaluated at the parameters $\boldsymbol{\psi}_k$ in force at that step. The *ideal sensitivity matrix*, which re-evaluates each historical immediate Jacobian at the current parameters, is

$$\boldsymbol{S}_t^* \;=\; \sum_{k=0}^{t} \left(\prod_{j=k+1}^{t} \boldsymbol{J}_j \right) \boldsymbol{I}_k(\boldsymbol{\psi}_t). \tag{8}$$

The staleness error is $E_t = \|\boldsymbol{S}_t^* - \tilde{\boldsymbol{S}}_t\|$.

Assumptions.

(A1) **Contraction.** $\|\boldsymbol{J}_t\| \leq \rho < 1$ uniformly in t.[1]

(A2) **Bounded updates.** $\|\boldsymbol{\psi}_t - \boldsymbol{\psi}_{t-1}\| \leq \eta G$ for all t, where η is a step size and G bounds the per-step update magnitude.

(A3) **Lipschitz immediate Jacobian.** The map $\boldsymbol{\psi} \mapsto \boldsymbol{I}_t(\boldsymbol{\psi})$ is Lipschitz with constant L_I, uniformly in t: $\|\boldsymbol{I}_t(\boldsymbol{\psi}) - \boldsymbol{I}_t(\boldsymbol{\psi'})\| \leq L_I \|\boldsymbol{\psi} - \boldsymbol{\psi'}\|$.

(A4) **Bounded magnitude.** $\|\boldsymbol{I}_t(\boldsymbol{\psi})\| \leq M_I$ uniformly.

Bound on the sensitivity matrix. Taking norms in the stale recursion and applying the triangle inequality, $\|\tilde{\boldsymbol{S}}_t\| \leq \rho \|\tilde{\boldsymbol{S}}_{t-1}\| + M_I$, which unrolls to the geometric steady-state bound $\|\tilde{\boldsymbol{S}}_t\| \leq M_I/(1-\rho)$.

Error recurrence. Expanding $\boldsymbol{S}_{t+1}^*$ and $\tilde{\boldsymbol{S}}_{t+1}$, the injection at step $t+1$ cancels because both contain $\boldsymbol{I}_{t+1}(\boldsymbol{\psi}_{t+1})$. Adding and subtracting $\boldsymbol{I}_k(\boldsymbol{\psi}_t)$ inside the bracket of the resulting difference and factoring out the leading Jacobian $\boldsymbol{J}_{t+1}$,

$$\boldsymbol{S}_{t+1}^* - \tilde{\boldsymbol{S}}_{t+1} \;=\; \boldsymbol{J}_{t+1} \sum_{k=0}^{t} \left(\prod_{j=k+1}^{t} \boldsymbol{J}_j \right) \big[\boldsymbol{I}_k(\boldsymbol{\psi}_{t+1}) - \boldsymbol{I}_k(\boldsymbol{\psi}_t) \big] \;+\; \boldsymbol{J}_{t+1} \big(\boldsymbol{S}_t^* - \tilde{\boldsymbol{S}}_t \big). \tag{9}$$

Taking norms, applying (A1) to each product of Jacobians, applying (A2) and (A3) to the freshly injected term, and bounding the geometric sum by $1/(1-\rho)$,

$$E_{t+1} \;\leq\; \rho \cdot \frac{L_I \, \eta \, G}{1-\rho} \;+\; \rho \, E_t. \tag{10}$$

This is a first-order linear recurrence with contraction factor $\rho < 1$ and constant driving term. The transient $\rho^t E_0$ vanishes and the geometric series converges, yielding the steady-state bound

$$E_\infty \;\leq\; \frac{\rho \, L_I \, G \, \eta}{(1-\rho)^2}. \tag{11}$$

Periodic updates. If $\boldsymbol{\psi}$ is held fixed for m steps between updates, the staleness injected at each update decays by ρ^m before the next, sharpening the bound to

$$E_\infty \;\leq\; \frac{\rho \, L_I \, G \, \eta}{(1-\rho)(1-\rho^m)}. \tag{12}$$

Since $1 - \rho^m \geq 1 - \rho$ for $m \geq 1$, periodic updates strictly reduce the steady-state staleness.

C.2 A first-order Taylor-trace correction

The leading-order term in the error recurrence (10) is the freshly injected staleness $\boldsymbol{I}_k(\boldsymbol{\psi}_{t+1}) - \boldsymbol{I}_k(\boldsymbol{\psi}_t)$. A first-order Taylor expansion in $\boldsymbol{\psi}$ gives

$$\boldsymbol{I}_k(\boldsymbol{\psi}_t) \;\approx\; \boldsymbol{I}_k(\boldsymbol{\psi}_{t-1}) \;+\; \left. \frac{\partial \boldsymbol{I}_k}{\partial \boldsymbol{\psi}} \right|_{\boldsymbol{\psi}_{t-1}} \Delta \boldsymbol{\psi}_{t-1}, \tag{13}$$

where $\Delta \boldsymbol{\psi}_{t-1} = \boldsymbol{\psi}_t - \boldsymbol{\psi}_{t-1}$. Substituting into $\boldsymbol{S}_t^*$ and grouping the second-order factors into an auxiliary accumulator

$$\boldsymbol{\Omega}_t \;=\; \boldsymbol{J}_t \, \boldsymbol{\Omega}_{t-1} \;+\; \left. \frac{\partial \boldsymbol{I}_t}{\partial \boldsymbol{\psi}} \right|_{\boldsymbol{\psi}_t} \tag{14}$$

[1] For an RTU layer with $\boldsymbol{\Lambda} = \mathrm{diag}(r_k e^{i\theta_k})$ the contraction constant is $\rho = \max_k r_k$, kept below one by the standard RTU parameterization.

yields a corrected update

$$\boldsymbol{S}_t \;=\; \boldsymbol{J}_t \left(\boldsymbol{S}_{t-1} + \boldsymbol{\Omega}_{t-1}\, \Delta\boldsymbol{\psi}_{t-1}\right) \;+\; \boldsymbol{I}_t(\boldsymbol{\psi}_t). \tag{15}$$

The additive term $\boldsymbol{J}_t\, \boldsymbol{\Omega}_{t-1}\, \Delta\boldsymbol{\psi}_{t-1}$ neutralizes the leading-order staleness injection, reducing the residual error from $\mathcal{O}(\|\Delta\boldsymbol{\psi}\|)$ to $\mathcal{O}(\|\Delta\boldsymbol{\psi}\|^2)$.

Storing $\boldsymbol{\Omega}_t$ exactly is intractable: since $\boldsymbol{I}_t$ has shape $|h| \times p$, $\boldsymbol{\Omega}_t$ is a rank-three tensor of shape $|h| \times p \times p$ requiring $\mathcal{O}(|h|\, p^2)$ memory. A parameter-wise diagonal approximation,

$$\boldsymbol{\omega}_t \;=\; \boldsymbol{J}_t\, \boldsymbol{\omega}_{t-1} \;+\; \mathrm{diag}_{\boldsymbol{\psi}}\left(\frac{\partial \boldsymbol{I}_t}{\partial \boldsymbol{\psi}}\right), \quad \boldsymbol{S}_t \;=\; \boldsymbol{J}_t \left(\boldsymbol{S}_{t-1} + \boldsymbol{\omega}_{t-1} \odot \Delta\boldsymbol{\psi}_{t-1}\right) \;+\; \boldsymbol{I}_t(\boldsymbol{\psi}_t), \tag{16}$$

recovers the original space complexity of the RTRL sensitivity, where $\odot$ denotes the elementwise product along the parameter axis.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy (EXC-3057/1 "Reasonable Artificial Intelligence", Project No. 533677015).
Noah Farr is supported by the Konrad Zuse School of Excellence in Learning and Intelligent Systems (**ELIZA**) through the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014. URL https://arxiv.org/abs/1409.1259.

Esraa Elelimy, Adam White, Michael Bowling, and Martha White. Real-time recurrent learning using trace units in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2024.

Esraa Elelimy, Brett Daley, Andrew Patterson, Marlos C. Machado, Adam White, and Martha White. Deep reinforcement learning with gradient eligibility traces, 2025. URL https://arxiv.org/abs/2507.09087.

Mohamed Elsayed, Gautham Vasan, and A. Rupam Mahmood. Streaming deep reinforcement learning finally works. *arXiv preprint arXiv:2410.14606*, 2024.

Noah Farr, Arne Backstein, Mahdi Kallel, and Aryaman Reddi. Memorax: A unified framework for memory-augmented reinforcement learning, 2025. URL https://github.com/noahfarr/memorax.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

Kazuki Irie, Anand Gopalakrishnan, and Jürgen Schmidhuber. Exploring the promise and limits of real-time recurrent learning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=V2cBKtdC3a.

Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation to the Jacobian. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=q3KSThy2GwB.

Steven Morad, Ryan Kortvelesy, Matteo Bettini, Stephan Liwicki, and Amanda Prorok. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *International Conference on Machine Learning*, pp. 16691–16723. PMLR, 2022.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences, 2023. URL https://arxiv.org/abs/2303.06349.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

Gautham Vasan, Mohamed Elsayed, Alireza Azimi, Jiamin He, Fahim Shahriar, Colin Bellinger, Martha White, and A. Rupam Mahmood. Deep policy gradient methods without batch updates, target networks, or replay buffers. In *Advances in Neural Information Processing Systems*, 2024.

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