Realtime Reinforcement Learning: Towards Rapid Asynchronous Deployment of Large Models

Matthew Riemer*
Mila, UdeM
IBM Research
mdriemer@us.ibm.com

Gopeshh Subbaraj*
Mila, UdeM
gopeshh.subbaraj@mila.quebec

Glen Berseth
Mila, UdeM
glen.berseth@mila.quebec

Irina Rish
Mila, UdeM
irina.rish@mila.quebec

Abstract

Realtime environments change even as agents perform action inference and learning, thus requiring high interaction frequencies to effectively minimize long-term regret. However, recent advances in machine learning involve larger neural networks with longer inference times, raising questions about their applicability in realtime systems where quick reactions are crucial. We present an analysis of lower bounds on regret in realtime environments to show that minimizing long-term regret is generally impossible within the typical sequential interaction and learning paradigm, but often becomes possible when sufficient asynchronous compute is available. We propose novel algorithms for staggering asynchronous inference processes to ensure that actions are taken at consistent time intervals, and demonstrate that use of models with high action inference times is only constrained by the environment's effective stochasticity over the inference horizon, and not by action frequency. Our analysis shows that the number of inference and learning processes needed scales linearly with increasing inference times while enabling use of models that are multiple orders of magnitude larger than existing approaches when learning from a realtime simulation of Game Boy games such as Pokémon and Tetris.

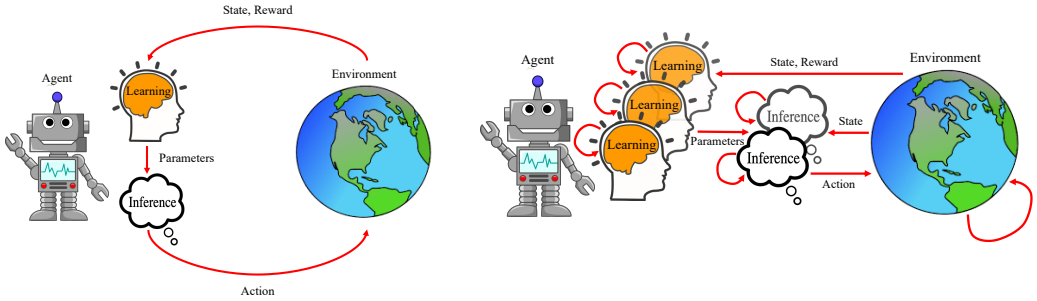
1 Introduction

An often ignored discrepancy between the discrete-time RL framework and the real-world is the fact that the world continues to evolve even while agents are computing their actions. As a result, choosing a particular stochastic or deterministic time discretization rate is fundamental in shaping the agent's understanding of the scope of its impact on the environment in the presence of constant change. Agents that take infrequent actions require some lower-level program to manage behavior between actions, often through simple policies like remaining still or repeating the last action. Ideally, intelligent agents would exert more control over their environment, but this conflicts with the trend of using larger models, which have high action inference and learning times. Consequently, as typically deployed with sequential interaction, large models, which are often found to be essential for complex tasks, increasingly rely on low-level automation, reducing their control over realtime environments. This paper examines this discrepancy and explores alternative asynchronous interaction paradigms, enabling large models to act quickly and maintain greater control in high-frequency environments.

Figure 1a shows the standard sequential interaction paradigm of RL. In this setup, the agent receives a state from the environment, learns from the state transition, and then infers an action. Each process must be completed before the agent can process a new state, limiting the action frequency and

*These authors contributed equally to this work

38th Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2024).



(a) Sequential Interaction and Learning

(b) Asynchronous Multi-process Interaction and Learning

Figure 1: **Frameworks for Environment Interaction in RL**. a) The typical sequential interaction paradigm where both learning and action inference block the environment from moving forward. b) The more realistic setting considered in this work where the environment, the agent's inference process, and agent's learning process all proceed at their own rate and interact asynchronously. Multiple self-loops are depicted for learning and inference to denote multiple asynchronous processes.

increasing reliance on low-level automation as the model size grows. In contrast, Figure 1b illustrates the asynchronous, multi-process interaction paradigm we propose. Our key insight is that even models with high inference times can act at every step using sufficiently many staggered inference processes. Similarly, sufficiently many asynchronous learning processes can maintain rapid interaction without blocking progress, despite high inference and learning times. This work formalizes and empirically tests the benefits and limitations of this approach, making the following contributions:

1. We formalize how the choice of a particular time discretization induces a new learning problem and how that problem relates to the original learning problem in Definition 1.
2. We derive worst-case lower bounds on regret for solving the new problem in terms of the original problem in Theorem 1, leading us to conclude in Remark 1 that the typical sequential interaction framework (Figure 1a) scales poorly with model size.
3. We propose novel approaches for staggering asynchronous inference in Algorithms 1 and 2, addressing the poor scaling properties of sequential interaction (Remark 2).
4. We conduct comprehensive experiments to verify our theory, and demonstrate the application of orders of magnitude larger models to realtime games like Pokémon and Tetris.

2 Formalizing Time Discretization in Realtime Reinforcement Learning

Background - Sequential Interaction: Most RL research focuses on agents interacting sequentially with a Markov Decision Process (MDP) [1; 2] $\mathcal{M}_{\text{seq}} = \langle \mathcal{S}, \mathcal{A}, p, r \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $r(s, a)$ is a reward function with outputs bounded by $r_{\max}$, and $p(s'|s, a)$ is a state transition probability function. Agents take actions based on a policy $\pi_\theta(a|s)$ that maps states to action probabilities parameterized by θ. It is assumed that the time between decisions (which can be denoted $\tau_{\mathcal{M}}$ as it only depends on the MDP in this case) is constant and the environment can be paused while the policy generates an action a from state s. The discrete decision step number t is then given by $t = \lfloor \tau / \tau_{\mathcal{M}} \rfloor$, where τ is the elapsed time in seconds, excluding pauses.

Asynchronous Interaction Environments: The standard MDP formalism lacks a crucial element for realtime settings where the environment cannot be "paused," and the agent interacts with it asynchronously, as described by Travnik et al. [3]. In this case, it is necessary to define the environment's behavior when the agent has not selected an action. We believe the most general solution is to use a preset default behavior if there is no available action a_t by the agent π at time-step t. This behavior follows $a \sim \beta(s)$, where $a \in \mathcal{A}_\beta$ is possibly from a different action space than $\mathcal{A}$, requiring p and r to be defined over $\mathcal{A} \cup \mathcal{A}_\beta$. Now we can define an asynchronous MDP $\mathcal{M}_{\text{async}} = \langle \mathcal{S}, \mathcal{A}, p, r, \beta \rangle$ as an extension of a sequential MDP $\mathcal{M}_{\text{seq}}$ with the addition of the default behavior policy β. Note that β need not be non-Markovian, because the state space should be defined to include any intermediate computations needed to generate the actions of the default behavior policy. Defining the default behavior as a policy is equivalent to the environment following a Markov chain $p^\beta(s'|s)$ when no action is available where $p^\beta(s'|s) := \sum_{a \in \mathcal{A}_\beta} p(s'|s, a)\beta(a|s)$ with expected reward $r^\beta(s) = \sum_{a \in \mathcal{A}_\beta} \beta(a|s) r(s, a)$.

Time Discretization Rates: The real environment evolves in continuous time, so we must define time discretization rates to describe each component of the agent-environment interface in discrete

steps. We treat the environment step time as a random variable $\mathrm{T}_{\mathcal{M}}$ with sampled values $\tau_{\mathcal{M}} \sim \mathrm{T}_{\mathcal{M}}$ and expected value $\bar{\tau}_{\mathcal{M}} := \mathbb{E}[\mathrm{T}_{\mathcal{M}}]$. Similarly, the environment interaction time (a.k.a. the action cycle time or inverse of the interaction frequency) is a random variable $\mathrm{T}_{\mathcal{I}}$ with sampled values $\tau_{\mathcal{I}} \sim \mathrm{T}_{\mathcal{I}}$ and expected value $\bar{\tau}_{\mathcal{I}} := \mathbb{E}[\mathrm{T}_{\mathcal{I}}]$. The action inference time of the policy is another random variable T_θ with sampled values $\tau_\theta \sim \mathrm{T}_\theta$ and expected value $\bar{\tau}_\theta := \mathbb{E}[\mathrm{T}_\theta]$.[1] This sets the stage for defining the decision problem induced by these choices related to the agent-environment boundary.

Definition 1 (Induced Delayed Semi-MDP) *Any configuration of random variables $\mathrm{T}_{\mathcal{M}}$, $\mathrm{T}_{\mathcal{I}}$, and T_θ applied to a asynchronous MDP $\mathcal{M}_{async}$ induces a delayed semi-MDP $\tilde{\mathcal{M}}_{delay} := \langle \mathcal{S}, \mathcal{A}, p, r, \beta, \mathrm{T}_{\mathcal{M}}, \mathrm{T}_{\mathcal{I}}, \mathrm{T}_\theta \rangle$ where the semi-MDP decision making steps $\tilde{t}$ associated with the actual decisions of the agent π happen after $\lceil \tau_{\mathcal{I}}/\tau_{\mathcal{M}} \rceil$ steps t in the ground asynchronous MDP $\mathcal{M}_{async}$. The semi-MDP is delayed with respect to $\mathcal{M}_{async}$ because semi-MDP actions $\tilde{a}_{\tilde{t}} \in \mathcal{A}$ generated by π are equivalent to actions that are delayed by $\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil$ in $\mathcal{M}_{async}$ such that $\pi_\theta(\tilde{a}_{\tilde{t}}|s_{\tilde{t}}) = \pi_\theta(a_{t+\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil}|s_t)$ where $s_{\tilde{t}} = s_t$. If $\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil > 1$ the transition dynamics are p^β and reward dynamics are r^β for $\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil - 1$ steps in $\mathcal{M}_{async}$ until $a_{t+\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil}$ is applied.*

In general, the optimal policy and optimal reward rate will not be the same for $\mathcal{M}_{\text{async}}$ and $\tilde{\mathcal{M}}_{\text{delay}}$, with $\tilde{\mathcal{M}}_{\text{delay}}$ incurring additional sub-optimality because of the coarse nature of the decision problem. That said, we have direct control over $\mathrm{T}_{\mathcal{I}}$ and T_θ, so it is of interest to understand how our design decisions relate to the sub-optimality experienced. Chiefly, we are interested in understanding under what scenarios the optimal reward rate of $\mathcal{M}_{\text{async}}$ can still be achieved even when $\bar{\tau}_\theta >> \bar{\tau}_{\mathcal{M}}$. To do this, we focus on worst case lower bounds on regret i.e. the unavoidable regret incurred because of the interaction defined by $\tilde{\mathcal{M}}_{\text{delay}}$ in the worst case scenario where β is always a suboptimal choice.

Theorem 1 (Realtime Regret Decomposition) *The total accumulated realtime regret $\Delta_{realtime}(\tau)$ as a function of time τ of a delayed semi-MDP $\tilde{\mathcal{M}}_{delay}$ relative to the oracle policy in the underlying asynchronous MDP $\mathcal{M}_{async}$ can be decomposed into three independent terms.*

$$\Delta_{realtime}(\tau) = \Delta_{learn}(\tau) + \Delta_{inaction}(\tau) + \Delta_{delay}(\tau) \tag{1}$$

$\Delta_{learn}(\tau)$ is the regret experienced even in sequential environments as a result of necessary learning and exploration. The lower bound on the regret of this term in the worst case is:[2]

$$\Delta_{learn}(\tau) \in \Omega(\sqrt{\tau(\bar{\tau}_{\mathcal{I}}/\bar{\tau}_{\mathcal{M}})}) \tag{2}$$

$\Delta_{inaction}(\tau)$ expresses the regret as a result of following β rather than optimal actions in $\mathcal{M}_{async}$. The lower bound and upper bound on the regret of this term in the worst case is:

$$\Delta_{inaction}(\tau) \in \Theta(\tau(\bar{\tau}_{\mathcal{I}} - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}}) \tag{3}$$

$\Delta_{delay}(\tau)$ expresses the regret as a result of the delay of actions by π in the underlying asynchronous $\mathcal{M}_{async}$. The lower bound on the regret of this term in the worst case is:

$$\Delta_{delay}(\tau) \in \Omega(\tau \times \mathbb{E}[(1 - p_{minimax})^{\tau_\theta/\tau_{\mathcal{M}}}]) \tag{4}$$

where $p_{minimax} := \min_{s \in \mathcal{S}, a \in \mathcal{A}} \max_{s' \in \mathcal{S}} p(s'|s,a)$ is a measure of environment stochasticity.[3]

See Appendix B for a formal proof of Theorem 1 and our other findings. We believe this work is the first to formally state the regret decomposition in Equation 1. Note that previous studies on real-world RL have highlighted the challenges of learning from limited samples, realtime inference, and managing system delays in scaling methods to realtime settings [5]. Equation 2 extends known lower bounds on learning time [6], using the notation from Definition 1 to explicitly connect with continuous time. Notably, this bound depends on $\bar{\tau}_{\mathcal{I}}$ (not $\bar{\tau}_\theta$) and assumes learning can keep pace with the environment to learn from every interaction. Equation 3 provides a novel regret bound,

[1] While policies in general could have adaptive computation times based on the state being processed, this is relatively uncommon in the RL literature and will be left to future work for simplicity of the discourse.

[2] Known algorithms achieve regret upper bounds within a logarithmic factor of this lower bound [4].

[3] When the environment is deterministic, $p_{\text{minimax}} = 1$ and $\Delta_{\text{delay}}(\tau) = 0$. When the environment is uniformly random, $p_{\text{minimax}} = 1/|\mathcal{S}|$ and $\Delta_{\text{delay}}(\tau)$ is maximized such that as $|\mathcal{S}| \to \infty$, $\Delta_{\text{delay}}(\tau) \in \Omega(\tau)$.

formalizing the known suboptimality of interacting with realtime environments at a slower pace [3; 7; 8; 9; 10; 11]. This result highlights the limitations of the sequential interaction paradigm.

Remark 1 (Realtime Regret of Sequential Interaction) *When π and $\mathcal{M}_{async}$ interact sequentially, $\tau_{\mathcal{I}} \in \Omega(\tau_\theta)$ such that in the worst case $\Delta_{inaction}(\tau) \in \Omega(\tau(\bar{\tau}_\theta - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$. This implies that even as $\tau \to \infty$, in the worst case $\Delta_{realtime}(\tau)/\tau \in \Omega(\Delta_{inaction}(\tau)/\tau) \in \Omega((\bar{\tau}_\theta - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$.*

This means a realtime framework with sequential interaction cannot ensure that regret will eventually dissipate. Thus, we explore asynchronous alternatives in the next section. Finally, Equation 4 highlights the key limitation in minimizing regret using asynchronous compute. Previous work established that suboptimality from delay in MDPs relates to the stochasticity in the underlying undelayed MDP [12; 13], focusing on communication delays inherent to the environment. Our focus, however, is on delays caused by the agent's computations, which we can control. Thus, the emphasis on regret associated with the decision that leads to a particular value of τ_θ is novel. Since this term is the only part of regret that depends on τ_θ, it helps identify which environments are manageable when $\tau_\theta >> \tau_{\mathcal{M}}$. In deterministic environments, there is no regret due to τ_θ as $p_{\text{minimax}} = 1$, but in stochastic environments, the degree and temporal horizon of stochasticity determine what values of τ_θ are tolerable. For simplicity, we present a looser bound here; a tighter bound is available in Appendix B. Stochasticity with respect to actual rewards (not just transitions) is what really matters.

3 Algorithms for Asynchronous Interaction and Learning

Figure 2 highlights key differences between the standard sequential RL framework and the asynchronous multi-process framework we propose. In the sequential framework, interaction and learning delay each other. In contrast, in the multi-process asynchronous framework that we propose, actions and learning can occur at every step with enough processes. However, actions are delayed and reflect past states, which may limit performance in some environments. Note that staggering processes to maintain regular intervals is essential. For example, if inference processes took a deterministic amount of time with no offset, all additional actions in the environment would be overwritten and there would be no benefit of increasing compute. Meanwhile, with staggering we can experience linear speedups.

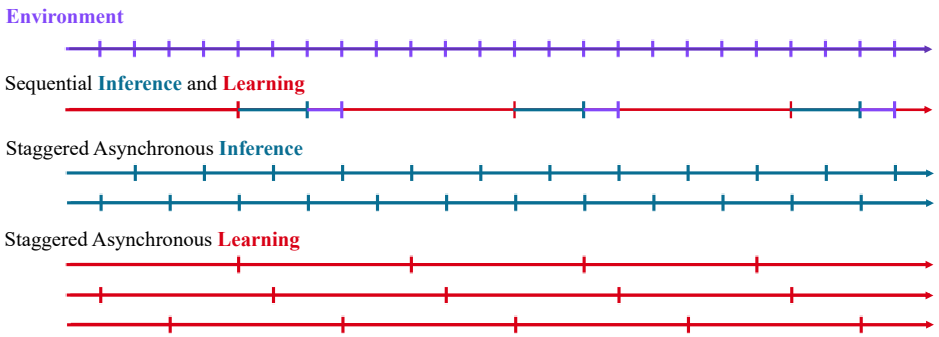


Figure 2: **Realtime Interaction Frequency.** We illustrate the comparative interaction frequency of methods that sequence learning and inference and those that maintain multiple staggered asynchronous processing processes. Even when inference times are greater than the environment step time, it is possible to use asynchronous compute to eliminate inaction and learn from every environment step.

3.1 Background: Staggered Asynchronous Learning

Parallel vs. Asynchronous Updates: Learning from a transition, i.e., computing gradients, usually takes longer than inference. Thus, performing learning in separate processes is crucial to avoid blocking inference [10], especially for models with a large number of parameters. For this use case, one might be tempted to consider parallel learning processes to increase the effective batch size without increasing wall-clock time per batch as this avoids wasted computation. Indeed, parallel updates are better for training large language models when final performance and compute efficiency are most important. In contrast, asynchronous learning can produce updates even faster than learning from a single transition, making the model more responsive to exploration. However, lock-free asynchronous approaches risk overwriting updates, potentially wasting computation that does not contribute to the final performance. Our focus is on maximizing responsiveness in large models, not

necessarily compute efficiency. As such, even overwritten updates are not wasted with respect to total regret.

Round-Robin Asynchronous Learning: Langford et al. [14] laid the foundation for addressing asynchronous update staggering for large neural network models using variants of stochastic gradient descent (SGD). They showed that applying updates in a delayed, orderly fashion avoids wasted compute on overwritten gradients. Their approach demonstrated convergence for delayed SGD, with linear scaling limited only by the time taken to update parameters relative to computing gradients. This method allows significant linear scaling with minimal compute waste for large models and the delay in the updates will not be a significant source of regret in Theorem 1. While not our novel contribution, this strategy is underexplored. We investigate its scaling properties in our experiments.

3.2 Our Contribution: Staggered Asynchronous Inference

In Remark 1 we highlighted that $\bar{\tau}_{\mathcal{I}}$ within the sequential interaction framework is fundamentally limited by $\bar{\tau}_\theta$, which results in persistent regret even as time goes on when $\bar{\tau}_\theta > \bar{\tau}_{\mathcal{M}}$. We will now highlight two novel algorithms for staggering inference processes that can lead to a reduction in $\bar{\tau}_{\mathcal{I}}$ when the number of inference processes $N_{\mathcal{I}}$ are increased. Algorithm 1 is capable of scaling the expected interaction time with the number of processes by $\bar{\tau}_{\mathcal{I}} = \min(\tau_\theta^{\max}/N_{\mathcal{I}}, \bar{\tau}_{\mathcal{M}})$ where $\tau_\theta^{\max}$ is the maximum encountered value of τ_θ. Meanwhile, algorithm 2 is capable of scaling the expected interaction time with the number of processes by $\bar{\tau}_{\mathcal{I}} = \min(\bar{\tau}_\theta/N_{\mathcal{I}}, \bar{\tau}_{\mathcal{M}})$. Both algorithms can eliminate inaction.

Remark 2 (Inaction Regret of Asynchronous Interaction) *For any $\bar{\tau}_\theta$ when π and $\mathcal{M}_{async}$ interact asynchronously with staggering algorithms 1 or 2, there is a value of the number of inference processes $N_{\mathcal{I}}^*$ such that for all $N_{\mathcal{I}} \geq N_{\mathcal{I}}^*$, $\Delta_{inaction}(\tau) \to 0$ as time goes on to $\tau \to \infty$.*

Algorithm 1 always ensures each processes waits for the current estimate of $\tau_\theta^{\max}$ amount of seconds before an action is taken by that process to preserve the spacing between actions. Adjustments are made to the waiting time in each process until the estimate converges to the true $\tau_\theta^{\max}$ value. The benefit of this algorithm is that the spacing between actions stays very consistent with no variance once the maximum value estimate has stabilized. This makes $\tilde{\mathcal{M}}_{\text{delay}}$ easier to learn from. The downside, however, is that the amount of necessary compute to eliminate inaction may be relatively high.

On the other hand, algorithm 2 stops all waiting in all processes as time goes on, so that the expected interaction time of each process is $\bar{\tau}_\theta$. An estimate of $\bar{\tau}_\theta$ is maintained and when the estimate changes after an action is taken, processes wait for an amount of time designed to adjust the average spacing between processes to $\bar{\tau}_\theta/N_{\mathcal{I}}$. The law of large numbers ensures that the estimate converges to $\bar{\tau}_\theta$ in the limit as $\tau \to \infty$ and that the waiting time diminishes to zero. Algorithm 2 has a strictly smaller compute requirements than algorithm 1, but experiences variance in $\mathrm{T}_{\mathcal{I}}$ driven by the variance in T_θ, which makes $\tilde{\mathcal{M}}_{\text{delay}}$ harder to learn from. The compute advantage becomes more significant for distributions that have variance in T_θ such that $\tau_\theta^{\max} - \bar{\tau}_\theta$ is large. In our experiments, we consider Algorithm 1 because we found the variance in τ_θ is extremely small for the models we consider.

4 Related Work

Realtime interaction: Previous work such as Travnik et al. [3] has considered the asynchronous nature of realtime environments. However, we are not aware of any prior paper that has formalized the connection between asynchronous and sequential versions of the same environment as we have. Travnik et al. [3] highlight the reaction time benefit of acting before you learn, and Ramstedt and Pal [8] highlight the reaction time benefit of interacting based on a one-step lag. Meanwhile, the interaction frequency of both of these approaches are limited by sequential interaction and thus the drawback highlighted in Remark 1 also applies to these approaches too.

Designing the interaction rate: Farrahi and Mahmood [11] examined how the choice of $\tau_{\mathcal{I}}$ affects the learning performance of deep RL algorithms in robotics. They found that low $\tau_{\mathcal{I}}$ complicates credit assignment, while high $\tau_{\mathcal{I}}$ complicates learning reactive policies. Karimi et al. [15] proposed a policy that executes multi-step actions with a learned $\tau_{\mathcal{I}}$ within the options framework, which may aid in slow problems where credit assignment is challenging. However, this approach does not address the action delay issue we focus on and may worsen it by committing to multiple actions based on a delayed state. Our policy, defined in the semi-MDP framework (Definition 1), relies on

Algorithm 1 Maximum Time Inference Staggering

Initialize: $\hat{\tau}_\theta^{\max} = 0$ and delay[processnum] $= \epsilon(\text{processnum} - 1)/N_\mathcal{I}$ $\forall$ processnum $\in [1, ..., N_\mathcal{I}]$
Run: INFERENCE[processnum] $\forall$ processnum $\in [1, ..., N_\mathcal{I}]$

function INFERENCE(processnum)
  while alive **do**
    sleep(delay[processnum]) ▷ Sleep for any delays accumulated by other processes
    delay[processnum] $\leftarrow 0$
    $a, \tau_\theta \sim \pi_\theta(s_t)$ ▷ Have the policy sample an action and inference time
    if $\tau_\theta \geq \hat{\tau}_\theta^{\max}$ **then**
      $\delta\tau \leftarrow \tau_\theta - \hat{\tau}_\theta^{\max}$ ▷ Other processes sleep for the difference with the maximum
      for num $\neq$ processnum $\in [1, ..., N_\mathcal{I}]$ **do**
        delay[num] $\leftarrow$ delay[num] + dist(num,processnum) $\times \delta\tau/N_\mathcal{I}$
      $\hat{\tau}_\theta^{\max} \leftarrow \tau_\theta$ ▷ Set new global maximum
    else
      sleep($\hat{\tau}_\theta^{\max} - \tau_\theta$) ▷ Sleep for the remaining time
    $a_{t+\lceil \hat{\tau}_\theta^{\max}/\tau_\mathcal{M} \rceil} \leftarrow a$ ▷ Register action in environment

a low-level policy β, similar to the options framework [16]. The key difference is that β cannot be modified by our policy, preventing intra-option learning and encouraging us to minimize its use.

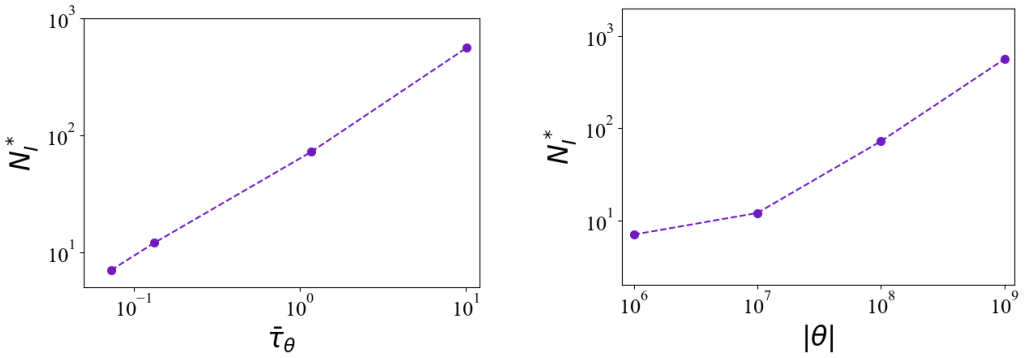
Reinforcement learning with delays: Reinforcement learning in environments with delayed states, observations, and actions is well-studied. Typically, delays are treated as communication delays inherent to the environment [17; 18]. In contrast, we focus on delays resulting from our computations, which are under our control and part of agent design. Our formulation of delay as part of regret is novel due to this unique focus. Common methods address delay by augmenting the state space with all actions taken since the delayed state or observation [19; 20; 21], but this is infeasible for us since these actions are not available when computation begins. Instead, our approach aligns more with methods addressing delay without state augmentation [22; 18; 23; 24; 25]. However, these methods are limited by the environment's stochasticity [12; 13], as highlighted by equation 4 of Theorem 1.

Asynchronous learning: Most work on asynchronous RL involves multiple environment simulators learned from asynchronously or in parallel [26; 27; 28]. We explore a more challenging real-world setup with a single environment, limiting exploration opportunities. Unlike typical asynchronous setups where each process interacts sequentially with the environment and then learn from that interaction [26], our setting benefits from making interaction and learning asynchronous (Remark 2). Similarly to ours, some prior work has considered asynchronous learning to avoid blocking inference [10], focusing on model-based learning [7; 29; 30; 9] and auxiliary value functions [31; 32]. The novelty of our approach is in its use of multiple asynchronous staggered inference processes instead of a single process, a critical contribution for deploying large models (Remark 1 and Remark 2).

5 Empirical Results

To show that our proposed method does indeed provide practical benefits for minimizing regret per second with large neural networks in realtime environments, we perform a suite of experiments to validate the theoretical claims made in Sections 2 and 3. Our experiments include:

- **Question 1:** an evaluation of the scaling properties of Algorithm 1 to demonstrate that the needed number of processes to eliminate inaction $N_\mathcal{I}^*$ does indeed scale linearly with increasing inference times $\bar{\tau}_\theta$ and parameter counts $|\theta|$.
- **Question 2:** an evaluation of the scaling properties of round-robin asynchronous learning [14] to demonstrate that the number of processes needed to learn from every transition $N_\mathcal{L}^*$ also scales linearly with increasing inference times $\bar{\tau}_\theta$ and parameter counts $|\theta|$.
- **Question 3:** an evaluation of the speed of progress through a realtime game with constant novelty to demonstrate that our proposed asynchronous algorithms can not only maintain better throughput of actions, but also maintain learning performance to make faster progress through a game in which agents must demonstrate competent behavior to go on.



(a) $N_{\mathcal{I}}^*$ vs. $\bar{\tau}_\theta$ for Algorithm 1

(b) $N_{\mathcal{I}}^*$ vs. $|\theta|$ for Algorithm 1

Figure 3: **a)** We plot the scaling behavior of the inference compute requirement $N_{\mathcal{I}}^*$ as the expected action inference time $\bar{\tau}_\theta$ increases for ResNet polices across CPUs in the Game Boy environment. **b)** We plot the scaling behavior of $N_{\mathcal{I}}^*$ instead as a function of the number of policy parameters $|\theta|$.

- **Question 4:** an evaluation of episodic reward after training in a game where reaction time is crucial to demonstrate that asynchronous interaction can maintain performance with models that are multiple orders of magnitude larger than those using sequential interaction.

Implementation Details: In all our experiments, we implemented the Deep Q-Network (DQN) learning algorithm [33] within our asynchronous multi-process framework, using a discount factor of $\gamma = 0.99$, a learning rate of 0.001 with the Adam optimizer, and a batch size of 16. A shared experience replay buffer stores the 1 million most recent environment transitions. For preprocessing, we down-sampled monochromatic Game Boy images to 84x84x1, similar to Atari preprocessing [33]. Following the scaling procedure previously established by [34] and [35], we used a 15-layer ResNet model [27] while scaling the number of filters by a factor k to grow the network. The model sizes correspond to: $k = 1$ (1M parameters), $k = 7$ (10M), $k = 29$ (100M), and $k = 98$ (1B). Models were deployed on multi-process CPUs using Pytorch multiprocessing on Intel Gold 6148 Skylake cores at 2.4GHz, with one core per process and multiple machines for models using > 40 processes. See Appendix A for further detailed regarding our experimental setup and a discussion of limitations.

5.1 Computational Scaling Properties for Asynchronous Interaction and Learning

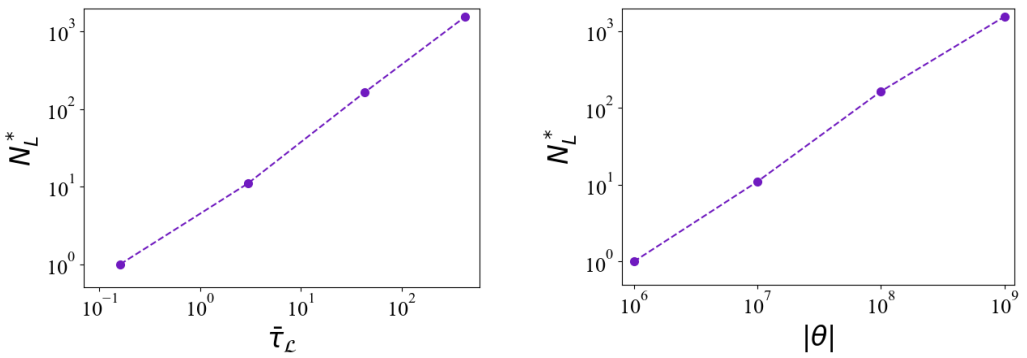
Realtime Game Boy Simulation: To run a comprehensive set of scaling experiments that would not be feasible with real-world deployment, we need a simulation of a realistic realtime scenario. Towards this end, we considered two games from the Game Boy that are made available for simulation as RL environments through the Gymnasium Retro project [36]. We implemented a realtime version of the Game Boy where it is run at 59.7275 frames per second such that $\tau_{\mathcal{M}} = 1/59.7275$ and with "noop" actions executed as the default behavior β. This exactly mimics the way that humans would interact with the Game Boy as a handheld console [37] and matches the setting in which humans compete over speed runs for these games. This is an ideal setting for addressing our core empirical questions.

Question 1: *How does $N_{\mathcal{I}}^*$ from Remark 2 scale with $\bar{\tau}_\theta$ and the number of parameters $|\theta|$?*

Figure 3: We measure $N_{\mathcal{I}}^*$ for the Game Boy when run at the standard frequency using a greedy DQN policy. We implemented maximum inference time staggering from Algorithm 1. Figure 3a shows that $N_{\mathcal{I}}^*$ scales roughly linearly with $\bar{\tau}_\theta$, as expected for effective staggering (Remark 2). Figure 3b also demonstrates that $N_{\mathcal{I}}^*$ scales roughly linearly with $|\theta|$.

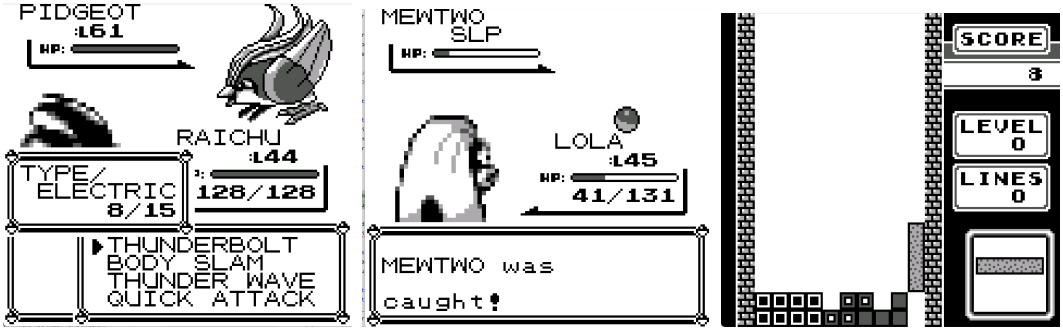
Notation for Asynchronous Learning: We also would like to consider the compute scaling properties of round-robin asynchronous learning [14]. We now assume that the time to learn from an environment transition can be treated as a random variable $\mathrm{T}_{\mathcal{L}}$ with sampled values $\tau_{\mathcal{L}} \sim \mathrm{T}_{\mathcal{L}}$ and expected value $\bar{\tau}_{\mathcal{L}} := \mathbb{E}[\mathrm{T}_{\mathcal{L}}]$. $N_{\mathcal{L}}^*$ will denote the number of learning processes such that all $N_{\mathcal{L}} \geq N_{\mathcal{L}}^*$ include at least one transition learned from for each transition in the environment in the long-run. This quantity is of significant interest because it expresses the number of learning processes needed to learn from each transition in the environment at least once given the frequency of the environment.

Question 2: *How does $N_{\mathcal{L}}^*$ scale with $\bar{\tau}_{\mathcal{L}}$ and the number of parameters $|\theta|$?*



(a) $N_{\mathcal{L}}^*$ vs. $\bar{\tau}_{\mathcal{L}}$ for Algorithm 1 (b) $N_{\mathcal{L}}^*$ vs. $|\theta|$ for Algorithm 1

Figure 4: **a)** We plot the scaling behavior of the learning compute requirement $N_{\mathcal{L}}^*$ as the expected transition learning time $\bar{\tau}_{\mathcal{L}}$ increases for ResNet polices across CPUs in the Game Boy environment. **b)** We plot the scaling behavior of $N_{\mathcal{L}}^*$ instead as a function of the number of policy parameters $|\theta|$.



(a) Pokémon Battling Environment (b) Pokémon Catching Environment (c) Tetris Environment

Figure 5: **a)** A frame from the final battle of Pokémon Blue when the agent is deciding on the next move. **b)** A frame from the final catching encounter of Pokémon Blue when the agent has just successfully caught Mewtwo. **C)** A frame from Tetris right before the agent completes its first line.

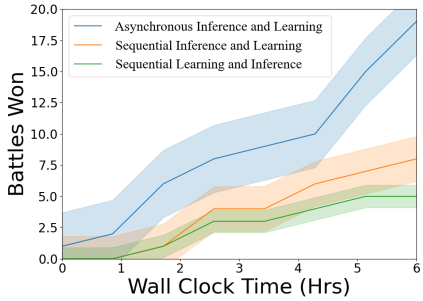
Figure 4: In Figure 4a we demonstrate that $N_{\mathcal{L}}^*$ grows approximately linearly with $\bar{\tau}_{\mathcal{L}}$. This scaling is in line with what we would expect for the Round-Robin algorithm for large and deep neural networks [14]. Additionally, our results in Figure 4b appear to also showcase linear scaling of $N_{\mathcal{L}}^*$ with $|\theta|$.

5.2 Faster Progress Through a Realtime Game with Constant Novelty

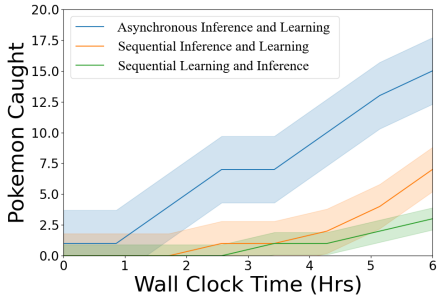
Pokémon Blue: Pokémon Blue is a valuable environment for our study due to its long play through time and constant novelty over many hours of play. Acting quickly is not a necessity to complete this game as it lets the agent dictate the pace of play, but better players are still differentiated based on their speed of completing the game. Indeed, the game has a large community of "speed runners" aiming to complete milestones in record times, with even the fastest milestones taking multiple hours [38]. It is an interesting domain for our study because acting quickly is only beneficial to the extent that the agent displays competent behavior so action throughput alone will not lead to better results when the quality of play correspondingly suffers. Because Pokémon Blue is known as a challenging exploration problem that perhaps even exceeds the scope of previous deep RL achievements [39], we divided the game into two settings based on expert human play: 295 battle encounters (Figure 5a) and 93 catching encounters (Figure 5b). Agents are deployed in these settings and must complete each encounter (by winning a battle or catching a Pokémon) before progressing.

Question 3: *Can asynchronous interaction and learning achieve faster progress through a realtime strategy game where constant learning is necessary to move forward even when models are large?*

Figure 9: For Pokémon Blue we leverage $N_{\mathcal{I}} = N_{\mathcal{I}}^*$ and $N_{\mathcal{L}} = N_{\mathcal{L}}^*/5$ as we did not find benefit from learning at every step given that the underlying game is not responsive to every action taken at the frame level. For all models, exploration rate is annealed from 1.0 to 0.05 over the course of the



(a) 100M: Pokémon Battles Won vs. Time τ



(b) 100M: Wild Pokémon Caught vs. Time τ

Figure 6: **Realtime Pokémon Performance for Staggered Asynchronous Interaction & Learning. a)** Battles won in Pokémon Blue as a function of time for $|\theta| = 100M$. **b)** Wild Pokémon caught in Pokémon Blue as a function of time for $|\theta| = 100M$.

first 100,000 steps of learning. We compare to the standard RL interaction paradigm where inference and learning are performed sequentially [2] and where the order is flipped for realtime settings [3]. Our results in Figures 9a and 9b showcase that asynchronous inference and learning combine for superior scaling of realtime performance as the model size grows. The performance improvement over the sequential interaction framework correspond with our expectations given Remarks 1 and 2.

5.3 Maintaining Performance in a Game that Prioritizes Reaction Time with Large Models

Tetris: We also explore the game Tetris (Figure 5c) that presents a different kind of challenge for our agents where even more of a premium is put on reaction time. In Tetris, the player will lose the game if they wait indefinitely and do not act in time. While a slow policy can eventually win the game in Pokémon, despite taking longer than necessary, a policy that does not act timely cannot progress through Tetris as new pieces must be moved into the correct spots before they fall on existing pieces.

Question 4: *Can asynchronous interaction help for games that prioritize reaction time as $|\theta|$ grows?*

Figure 7: To aid with exploration and jump-start learning, a single episode of human play is provided to each agent to learn from. The agent continues to learn from a total of 2,000 episodes with an exploration rate of 0.05. We see the sequential interaction scales quite poorly for games that prioritize a high frequency of actions and cannot surpass random performance for $|\theta| \geq$1M as we would expect based on Remark 1. Meanwhile, staggered asynchronous inference following Algorithm 1 can achieve a much higher reward rate for $|\theta| >$1B as we would anticipate based on Remark 2.

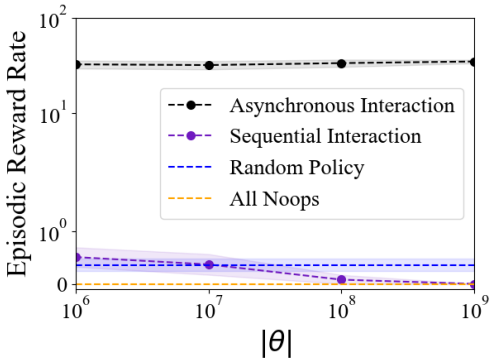


Figure 7: **Realtime Tetris Performance vs. $|\theta|$.** The average episodic return over 2,000 episodes of learning achievable with $N^*_{\mathcal{L}}$ asynchronous learning processes. We compare models with a single inference process with those that perform staggered asynchronous inference following Algorithm 1.

6 Conclusion

In this paper, we have taken a deeper look at RL in realtime settings and the viability of increasing the neural network model size in these environments. Our theoretical analysis of regret bounds has demonstrated the downfall of models that implement a single process of action inference as model sizes grow (Remark 1) and we have proposed staggering algorithms that address this limitation for environments that are sufficiently deterministic (Remark 2). Our empirical results on the realtime Game Boy games Pokémon Blue and Tetris corroborate these findings and demonstrate the ability to perform well in realtime environments with models that are orders of magnitude larger than what is achievable with a single inference process. While conventional wisdom often leads researchers to think that smaller models are necessary for realtime settings, our work indicates that this is not necessarily the case and takes a step towards making realtime deployment of foundation models realistic.

7 Acknowledgements

We would like to acknowledge our support from the Canada CIFAR AI Chair Program and from the Canada Excellence Research Chairs (CERC) Program. We thank the IBM Cognitive Compute Cluster, the Mila cluster, and Compute Canada for providing computational resources. We are grateful to Yoshua Bengio for his support during the initial discussions that helped shape the direction of this paper. We also thank Olexa Bilaniuk for his assistance with questions related to computational resources and multiprocessing.

References

[1] ML Puterman. Markov decision processes. 1994. *Jhon Wiley & Sons, New Jersey*, 1994.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[3] Jaden B Travnik, Kory W Mathewson, Richard S Sutton, and Patrick M Pilarski. Reactive reinforcement learning in asynchronous environments. *Frontiers in Robotics and AI*, 5:79, 2018.

[4] Ronald Ortner. Regret bounds for reinforcement learning via markov chain concentration. *Journal of Artificial Intelligence Research*, 67:115–128, 2020.

[5] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[6] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(4), 2010.

[7] Todd Hester, Michael Quinlan, and Peter Stone. A real-time model-based reinforcement learning architecture for robot control. *arXiv preprint arXiv:1105.1749*, 2011.

[8] Simon Ramstedt and Chris Pal. Real-time reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[9] Yunzhi Zhang, Ignasi Clavera, Boren Tsai, and Pieter Abbeel. Asynchronous methods for model-based reinforcement learning. *arXiv preprint arXiv:1910.12453*, 2019.

[10] Yufeng Yuan and A Rupam Mahmood. Asynchronous reinforcement learning for real-time control of physical robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5546–5552. IEEE, 2022.

[11] Homayoon Farrahi and A Rupam Mahmood. Reducing the cost of cycle-time tuning for real-world policy optimization. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2023.

[12] Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. *arXiv preprint arXiv:2101.11992*, 2021.

[13] Somjit Nath, Mayank Baranwal, and Harshad Khadilkar. Revisiting state augmentation methods for reinforcement learning with stochastic delays. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1346–1355, 2021.

[14] John Langford, Alexander J Smola, and Martin Zinkevich. Slow learners are fast. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 2331–2339, 2009.

[15] Amirmohammad Karimi, Jun Jin, Jun Luo, A Rupam Mahmood, Martin Jagersand, and Samuele Tosatto. Dynamic decision frequency with continuous options. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7545–7552. IEEE, 2023.

[16] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[17] Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18:83–105, 2009.

[18] Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2020.

[19] Dimitri P Bertsekas. Dynamic programming and optimal control. *Journal of the Operational Research Society*, 47(6):833–833, 1996.

[20] Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4): 568–574, 2003.

[21] Somjit Nath, Mayank Baranwal, and Harshad Khadilkar. Revisiting state augmentation methods for reinforcement learning with stochastic delays. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1346–1355, 2021.

[22] Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3226–3231. IEEE, 2010.

[23] Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. In *International Conference on Learning Representations*, 2021.

[24] Mridul Agarwal and Vaneet Aggarwal. Blind decision making: Reinforcement learning with delayed observations. *Pattern Recognition Letters*, 150:176–182, 2021.

[25] Armin Karamzade, Kyungmin Kim, Montek Kalsi, and Roy Fox. Reinforcement learning from delayed observations via world models. *arXiv preprint arXiv:2403.12309*, 2024.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[27] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Todd Hester, Michael Quinlan, and Peter Stone. Rtmba: A real-time model-based reinforcement learning architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pages 85–90. IEEE, 2012.

[30] Todd Hester and Peter Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90:385–429, 2013.

[31] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

[32] Gheorghe Comanici, Amelia Glaese, Anita Gergely, Daniel Toyama, Zafarali Ahmed, Tyler Jackson, Philippe Hamel, and Doina Precup. Learning how to interact with a complex interface using hierarchical reinforcement learning. *arXiv preprint arXiv:2204.10374*, 2022.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[34] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[35] Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

[36] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[37] Wikipedia. Game Boy — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Game%20Boy&oldid=1224667625, 2024. [Online; accessed 20-May-2024].

[38] speedrun.com. Pokemon red/blue leaderboard, 2024. URL https://www.speedrun.com/pkmnredblue.

[39] Shayaan Jagtap. Why can a machine beat mario but not pokemon?, 2018. URL https://towardsdatascience.com/why-can-a-machine-beat-mario-but-not-pokemon-ff61313187e1.

[40] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

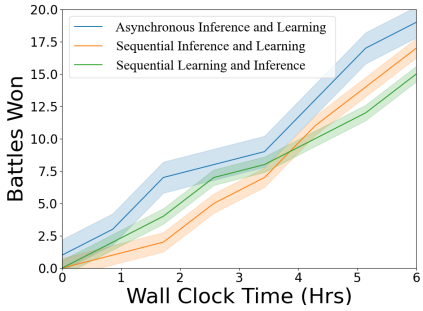
[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

A Further Details Supporting the Main Text

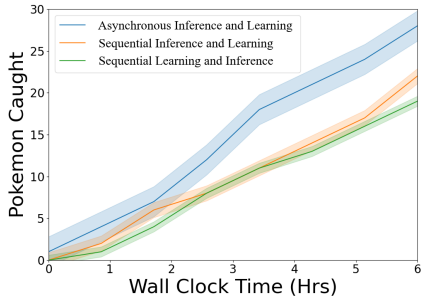
Software Libraries: Our experiments leverage Numpy [40], which is publicly available following a BSD license. Neural network models were developed using Pytorch [41], which is publicly available following a modified BSD license. The Gym Retro project [36] used to simulate the Game Boy in a RL environment is made available following a MIT license. We are not at liberty to distribute the proprietary ROMs associated with Pokémon or Tetris and each person that deploys our provided code must separately obtain their own copy.

Environment Details: We depict the environments considered in our paper in Figure 5. We consider six discrete actions for both Pokémon Blue and Tetris including the A button, the B button, the left directional button, the up directional button, the right directional button, and the down directional button. In the Battling Environment when the opponent Pokémon is knocked out by the agent's Pokémon a reward of 1 is received and a reward of -1 is received when a users Pokémon is knocked out. Battles include 1-6 Pokémon for the agent and 1-6 Pokémon for the opponent AI. In the Catching Environment a reward of 1 is received by the agent when a wild Pokémon is captured and -1 when the encounter is terminated unsuccessfully. In Tetris we provide changes in the in-game score as a reward for the agent to learn from.

Training Procedure for Tetris: The one episode of human play provided for Tetris included 16,000 steps where non-noop actions were taken. We used our experiments from Figure 3 to calculate

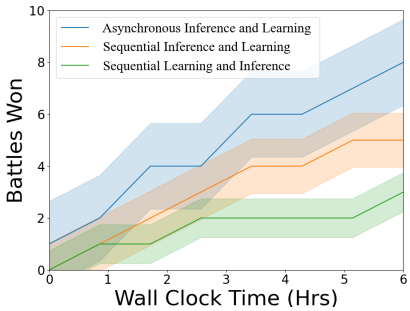


(a) 1M: Pokémon Battles Won vs. Time τ

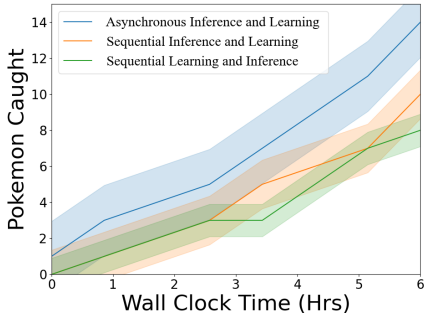


(b) 1M: Wild Pokémon Caught vs. Time τ

Figure 8: **Realtime Pokémon Performance for Staggered Asynchronous Interaction & Learning. a)** Battles won in Pokémon Blue as a function of time for $|\theta| = 1M$. **b)** Wild Pokémon caught in Pokémon Blue as a function of time for $|\theta| = 1M$.



(a) 10M: Pokémon Battles Won vs. Time τ



(b) 10M: Wild Pokémon Caught vs. Time τ

Figure 9: **Realtime Pokémon Performance for Staggered Asynchronous Interaction & Learning. a)** Battles won in Pokémon Blue as a function of time for $|\theta| = 10M$. **b)** Wild Pokémon caught in Pokémon Blue as a function of time for $|\theta| = 10M$.

the amount of action delay per step for each model and populated the replay buffer with 16,000 transitions corresponding to these actions with observations delayed by the expected amount for each model. We the trained the model for 16,000 steps before tuning the model in a simulation of the environment with the corresponding amount of delay 2,000 for episodes. The episodic reward from Figure 7 corresponds to the average episodic reward achieved during that training period.

Statistical Significance: We also note that error bar shading throughout our paper reflects 95% confidence intervals computed with three random seeds: 0, 1, and 2.

Limitations: In both our experiments on Pokémon Blue and Tetris, performance is well below human-level. This is because both of these games pose significant exploration problems and we train our models from scratch for a limited amount of time. We believe that these experiments are more than sufficient to showcase the benefits of staggered asynchronous inference in comparison to the sequential interaction framework by showcasing when the latter framework breaks down in realtime settings. However, we speculate that the results showing that game play does not suffer despite significant action delay will likely not generalize to more intricate human-level policies.

Algorithm Pseudocode: We provide detailed pseudocode for Algorithm 2, which could not be included in the main text due to space constraints.

Algorithm 2 Expected Time Inference Staggering

Initialize: $\hat{\bar{\tau}}_\theta = 0$, $\tau_{\text{tot}} = 0$, and $a_{\text{tot}} = 0$
Initialize: delay[processnum] $= \epsilon(\text{processnum} - 1)/N_{\mathcal{I}}$ $\forall$ processnum $\in [1, ..., N_{\mathcal{I}}]$
Run: INFERENCE[processnum] $\forall$ processnum $\in [1, ..., N_{\mathcal{I}}]$

function INFERENCE(processnum)
  while alive **do**
    sleep(delay[processnum]) ▷ Sleep for any delays accumulated by other processes
    delay[processnum] $\leftarrow 0$
    $a, \tau_\theta \sim \pi_\theta(s_t)$ ▷ Have the policy sample an action and inference time
    $a_{\text{tot}} \leftarrow a_{\text{tot}} + 1$
    $\tau_{\text{tot}} \leftarrow \tau_{\text{tot}} + \tau_\theta$
    $\hat{\bar{\tau}}'_\theta \leftarrow \tau_{\text{tot}}/a_{\text{tot}}$
    $\delta\tau \leftarrow \hat{\bar{\tau}}'_\theta - \hat{\bar{\tau}}_\theta$
    if $\delta\tau \geq 0$ **then** ▷ Wait more further from the current process
      for num $\neq$ processnum $\in [1, ..., N_{\mathcal{I}}]$ **do**
        delay[num] $\leftarrow$ delay[num] + dist(num, processnum) abs$(\delta\tau)/N_{\mathcal{I}}$
    else ▷ Wait more closer to the current process
      for num $\neq$ processnum $\in [1, ..., N_{\mathcal{I}}]$ **do**
        delay[num] $\leftarrow$ delay[num] + $(N_{\mathcal{I}} - 1)$dist(num,processnum) abs$(\delta\tau)/N_{\mathcal{I}}$
    $a_{t+\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil} \leftarrow a$ ▷ Register action in environment

B Proofs for Each Theoretical Statement

Our proofs rely on the following core assumptions, restated from Section 2 in the main text:

1. The environment step time can be treated as an independent random variable $\mathrm{T}_{\mathcal{M}}$ with sampled values $\tau_{\mathcal{M}} \sim \mathrm{T}_{\mathcal{M}}$ and expected value $\bar{\tau}_{\mathcal{M}} := \mathbb{E}[\mathrm{T}_{\mathcal{M}}]$.
2. The environment interaction time can be treated as an independent random variable $\mathrm{T}_{\mathcal{I}}$ with sampled values $\tau_{\mathcal{I}} \sim \mathrm{T}_{\mathcal{I}}$ and expected value $\bar{\tau}_{\mathcal{I}} := \mathbb{E}[\mathrm{T}_{\mathcal{I}}]$.
3. The action inference time of the policy can be treated as an independent random variable T_θ with sampled values $\tau_\theta \sim \mathrm{T}_\theta$ and expected value $\bar{\tau}_\theta := \mathbb{E}[\mathrm{T}_\theta]$.
4. Asynchronous learning can learn from every interaction with $\tilde{\mathcal{M}}_{\text{delay}}$.

B.1 Definition 1

Most of Definition 1 just recaps the dynamics of how the agent interacts with an asynchronous ground MDP following assumptions 1-3 about the nature of that interaction. All that is left to show is that this can be viewed as a delayed MDP and that it can be viewed as a semi-MDP. The interaction process highlighted in Definition 1 matches that of a Random Delay Markov Decision Process (RDMDP) [18] where the action delay distribution is defined by the random variable $\lceil \tau_\theta/\tau_{\mathcal{M}} \rceil$. To show it is a semi-MDP as well, we consider the same proof style of Theorem 1 in Sutton et al. [16]:

A semi-MDP consists of (1) a set of states, (2) a set of actions, (3) for each pair of state and action, an expected cumulative discounted reward, and (4) a well-defined joint distribution of the next state and transit time. We now demonstrate each of these properties. The set of states is S and the set of actions is $\mathcal{A}$. The expected reward and the next-state and transit-time distributions are well defined for every state and delayed action. These expectations and distributions are well defined because $\mathcal{M}_{\text{async}}$ is Markov, thus the next state, reward, and time are dependent only on the delayed action chosen and the state in which it was initiated. Transit times with arbitrary real intervals are permitted in semi-MDPs.

B.2 Theorem 1

To prove Theorem 1 we will demonstrate the validity of each equation of the theorem following the order of presentation in the main text.

Equation 1: By definition $\Delta_{\text{learn}}(\tau)$ and $\Delta_{\text{inaction}}(\tau)$ must be independent contribution to the total regret because learning regret is only incurred when acting in the environment following π and

inaction regret is only incurred when not acting in the environment and thus following the default behavior policy β. The interaction frequency does not depend on the parameter values of π as they change following from the independent random variable assumption. Even when regret from learning is eliminated and regret from inaction is eliminated there is still another independent source of regret that persists $\Delta_{\text{delay}}(\tau)$ reflecting the lower reward rate of the best possible policy in acting over $\tilde{\mathcal{M}}_{\text{delay}}$ in comparison to the best possible policy acting over $\mathcal{M}_{\text{async}}$.

Equation 2: The worst case lower bound for the standard notion of regret arising from the need for learning and exploration has been established as $\Delta_{\text{learn}}(\mathcal{T}) \in \Omega(\sqrt{\mathcal{T}})$ where $\mathcal{T}$ denotes the number of discrete learning steps taken in the environment. Given that we learn from every step in the environment following assumption 4, then the regret as a function of τ scales with the expected number of discrete environment steps as a function of $\mathcal{T}$ i.e. $\mathbb{E}[\mathcal{T}(\tau)] = \tau(\bar{\tau}_{\mathcal{I}}/\bar{\tau}_{\mathcal{M}})$ because $\mathrm{T}_{\mathcal{M}}$ and $\mathrm{T}_{\mathcal{I}}$ are independent. Therefor, $\Delta_{\text{learn}}(\tau) \in \Omega(\sqrt{\tau(\bar{\tau}_{\mathcal{I}}/\bar{\tau}_{\mathcal{M}})})$. As noted in the footnote in the main text, this analysis equally applies to the known $\Delta_{\text{learn}}(\mathcal{T}) \in \tilde{O}(\sqrt{\mathcal{T}})$ minimum upper bound [4].

Equation 3: The worst case lower bound on $\Delta_{\text{inaction}}(\tau)$ is derived by considering a worst case environment with two actions a_1 and a_2 and two states s_1 and s_2 where the default behavior β takes its own action a_3 at every state. The reward provided is 1 at s_1 and 0 at s_2. The next state is s_1 regardless of the state if either a_1 or a_2 is taken and s_2 if a_3 is taken. In this environment the optimal reward rate is 1 and the reward rate when following β is 0.0. The expected number of times a_3 is taken during τ seconds in $\mathcal{M}_{\text{async}}$ is then $\tau(\bar{\tau}_{\mathcal{I}} - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}}$ because $\mathrm{T}_{\mathcal{M}}$ and $\mathrm{T}_{\mathcal{I}}$ are independent. Therefor, $\Delta_{\text{inaction}}(\tau) \in \Omega(\tau(\bar{\tau}_{\mathcal{I}} - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$ for this particular environment. Meanwhile, the expected inaction regret is also upper bounded by $\Delta_{\text{inaction}}(\tau) \leq r_{\max}\tau(\bar{\tau}_{\mathcal{I}} - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$ where $r_{\max}$ is the maximum possible reward per step because by definition the agent cannot incur regret from inaction when it is acting in the environment. Therefor, we have demonstrated that equation 3 holds.

Equation 4: The worst case lower bound on $\Delta_{\text{delay}}(\tau)$ is derived by considering a worst case environment with two actions a_1 and a_2 and two states s_1 and s_2 where the default behavior β takes its own action a_3 at every state. The agent stays in the current state regardless of the action with probability p_{minimax} and goes to the other state with probability p_{minimax} where $p_{\text{minimax}} \leq 0.5$ by definition. The agent receives a reward of 1 for taking a_1 in s_1 or a_2 in s_2 and a reward of 0 otherwise. So the best reward rate as a function of τ that can be ensured with actions delayed by $\tau_\theta/\tau_{\mathcal{M}}$ is $\tau \times \mathbb{E}[(1-p_{\text{minimax}})^{\tau_\theta/\tau_{\mathcal{M}}}]$ meanwhile the best reward rate possible in $\mathcal{M}_{\text{async}}$ is τ, so the lower bound on the regret with respect to that optimal reward rate is $\Delta_{\text{delay}}(\tau) \in \Omega(\tau \times \mathbb{E}[(1 - p_{\text{minimax}})^{\tau_\theta/\tau_{\mathcal{M}}}])$.

B.3 Remark 1

We restate the derivation of the remark from the main text, filling in a bit more detail for clarity. When π and $\mathcal{M}_{\text{async}}$ interact sequentially, we must have $\tau_{\mathcal{I}} \geq \tau_\theta$, so $\Delta_{\text{inaction}}(\tau) \in \Omega(\tau(\bar{\tau}_{\mathcal{I}} - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}}) \in \Omega(\tau(\bar{\tau}_\theta - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$. This implies that even as $\tau \to \infty$, the worst case regret rate $\Delta_{\text{realtime}}(\tau)/\tau \in \Omega(\Delta_{\text{inaction}}(\tau)/\tau) \in \Omega((\bar{\tau}_\theta - \bar{\tau}_{\mathcal{M}})/\bar{\tau}_{\mathcal{M}})$ following from Theorem 1.

B.4 Remark 2

Algorithm 1 is capable of scaling the expected interaction time with the number of processes by $\bar{\tau}_{\mathcal{I}} = \min(\tau_\theta^{\max}/N_{\mathcal{I}}, \bar{\tau}_{\mathcal{M}})$ where $\tau_\theta^{\max}$ is the maximum encountered value of τ_θ as $\tau \to \infty$. This then implies that for $N_{\mathcal{I}} \geq N_{\mathcal{I}}^* = \lceil \tau_\theta^{\max}/\bar{\tau}_{\mathcal{M}} \rceil$, $\bar{\tau}_{\mathcal{I}} = \bar{\tau}_{\mathcal{M}}$. Algorithm 2 is capable of scaling the expected interaction time with the number of processes by $\bar{\tau}_{\mathcal{I}} = \min(\bar{\tau}_\theta/N_{\mathcal{I}}, \bar{\tau}_{\mathcal{M}})$ as $\tau \to \infty$ following the law of large numbers. This then correspondingly implies that for $N_{\mathcal{I}} \geq N_{\mathcal{I}}^* = \lceil \bar{\tau}_\theta/\bar{\tau}_{\mathcal{M}} \rceil$, $\bar{\tau}_{\mathcal{I}} = \bar{\tau}_{\mathcal{M}}$.