TREE SEARCH-BASED POLICY OPTIMIZATION UNDER STOCHASTIC EXECUTION DELAY

David Valensi
Technion
davidvalensi@campus.technion.ac.il

Esther Derman
Technion
estherderman@campus.technion.ac.il

Shie Mannor
Technion & Nvidia Research
shie@ee.technion.ac.il

Gal Dalal
Nvidia Research
gdalal@nvidia.com

ABSTRACT

The standard formulation of Markov decision processes (MDPs) assumes that the agent's decisions are executed immediately. However, in numerous realistic applications such as robotics or healthcare, actions are performed with a delay whose value can even be stochastic. In this work, we introduce stochastic delayed execution MDPs, a new formalism addressing random delays without resorting to state augmentation. We show that given observed delay values, it is sufficient to perform a policy search in the class of Markov policies in order to reach optimal performance, thus extending the deterministic fixed delay case. Armed with this insight, we devise DEZ, a model-based algorithm that optimizes over the class of Markov policies. DEZ leverages Monte-Carlo tree search similar to its non-delayed variant EfficientZero to accurately infer future states from the action queue. Thus, it handles delayed execution while preserving the sample efficiency of EfficientZero. Through empirical analysis, we stress that none of the prior benchmarks consistently outperforms others across different delays. We demonstrate that our algorithm surpasses all benchmark methods in Atari games when dealing with constant or stochastic delays. The code is available at https://github.com/davidva1/Delayed-EZ.

1 INTRODUCTION

The conventional Markov decision process (MDP) framework commonly assumes that all of the information necessary for the next decision step is available in real-time: the agent's current state is immediately observed, its chosen action instantly actuated, and the corresponding reward feedback concurrently perceived (Puterman, 2014). However, these input signals are often delayed in real-world applications such as robotics (Mahmood et al., 2018), healthcare (Politi et al., 2022), or autonomous systems, where they can manifest in different ways. Perhaps the most prominent example where delay can be stochastic is in systems that rely on data transmission. Oftentimes, there is some interference in transmission that stems from internal or external sources. Internal sources may be due to temperature or pressure conditions that influence the sensing hardware, whereas external sources of interference occur when a policy infers actions remotely (e.g., from a cloud). For example, an autonomous vehicle may experience delay from its perception module to recognize the environment around it. This initial recognition delay is known as *observation delay*. Additional delays can occur when taking action according to a decision previously made. This delay in response is termed *execution delay*. As highlighted in (Katsikopoulos & Engelbrecht, 2003) and despite their distinct manifestations, both types of delay are functionally equivalent and can be addressed using similar methodologies.

In addition to its existence, the delay's nature is often overlooked its nature. In complex systems, delays exhibit stochastic properties that add layers of complexity to the decision-making process (Dulac-Arnold et al., 2019). This not only mirrors their unpredictability but also warrants a novel approach to reinforcement learning (RL) that does not rely on state augmentation. This popular

method consists of augmenting the last observed state with the sequence of actions that have been selected by the agent, but which result has not yet been observed (Bertsekas, 2012; Altman & Nain, 1992; Katsikopoulos & Engelbrecht, 2003). Although it presents the advantage of recovering partial observability, this approach has two major limitations: (i) its computational complexity inevitably grows exponentially with the delay value (Derman et al., 2021); (ii) it structurally depends on that value, which prevents it from generalizing to random delays. As a matter of fact, the augmentation method introduced in (Bouteiller et al., 2020) to tackle random delays was practically tested on small delay values and/or small support.

In this work, we tackle the question: How does one effectively engage in an environment where action repercussions are subject to random delays? We introduce the paradigm of stochastic-execution-delay MDPs (SED-MDPs). We then establish a significant finding: To address stochastic delays in RL, it is sufficient to optimize within the set of Markov policies, which is exponentially smaller than that of history-dependent policies. Our result improves upon the one in (Derman et al., 2021) that tackled the narrower scope of deterministic delays.

Based on the observation above, we devise *Delayed EfficientZero* (DEZ). This model-based algorithm builds on the strengths of its predecessor, EfficientZero, by using Monte Carlo tree search to predict future actions (Ye et al., 2021). In practice, DEZ keeps track of past actions and their delays using two separate queues. It utilizes these queues to infer future states and make decisions accordingly. We also improve the way the algorithm stores and uses data from previous experience, enhancing its overall accuracy. In essence, DEZ offers a streamlined approach to managing stochastic delays in decision-making. We accordingly modify the Atari suite to incorporate both deterministic and stochastic delays, where the delay value follows a random walk. In both cases, our algorithm surpasses 'oblivious' EfficientZero that does not explicitly account for delay, and 'Delayed-Q' from Derman et al. (2021).

In summary, our contributions are as follows.

1. We formulate the framework of MDPs with stochastic delay and address the principled approach of state augmentation.
2. We prove that if the realizations of the delay process are observed by the agent, then it suffices to restrict policy search to the set of Markov policies to attain optimal performance.
3. We devise DEZ, a model-based algorithm that builds upon the prominent EfficientZero. DEZ yields non-stationary Markov policies, as expected by our theoretical findings. It is agnostic to the delay distribution, so that no assumption is made concerning the delay values themselves.
4. We thoroughly test DEZ on the Atari suite under both deterministic and stochastic delay schemes. In both cases, our method achieves significantly higher reward than the original EfficientZero and 'Delayed-Q' from Derman et al. (2021).

2 Related work

Although it is critical for efficient policy deployment, the notion of delayed execution remains scarce in the RL literature. One way to address this is through state-space augmentation, which consists of concatenating all pending actions to the original state. This brute-force method presents the advantage of recovering the Markov property, but its computational cost increases exponentially with the delay value (Walsh et al., 2009; Derman et al., 2021).

Previous works that addressed random delay using state-embedding include Katsikopoulos & Engelbrecht (2003); Bouteiller et al. (2020). The work Katsikopoulos & Engelbrecht (2003) simply augments the MDP with the maximal delay value to recover all missing information. In (Bouteiller et al., 2020), such augmented MDP is approximated by a neural network, and trajectories are resampled to compensate for the actual realizations of the delay process during training. The proposed method is tested on a maximal delay of 6 with a high likelihood near 2. This questions the viability of their approach to (i) higher delay values; and (ii) delay distributions that are more evenly distributed over the support. Differently, by assuming the delay process to be observable, our method augments the state space by one dimension only, regardless of the delay value. It also shows effi-

ciency for delay values of up to 25, which we believe comes from the agnosticism of our network structure to the delay.

To avoid augmentation, Walsh et al. (2009) alternatively propose to infer the most likely present state and deterministically transition to it. Their model-based simulation (MBS) algorithm necessitates the original transition kernel to be almost deterministic for tractability. Additionally, MBS proceeds offline and requires a finite or discretized state space, which raises the curse of dimensionality. (Derman et al., 2021) address the scalability issue through their Delayed DQN algorithm, which presents two main features: (i) In the same spirit as MBS, Delayed DQN learns a forward model to estimate the state induced by the current action queue; (ii) This estimate is stored as a next-state observation in the replay buffer, thus resulting in a time shift for the Bellman update. Although (Walsh et al., 2009; Derman et al., 2021) avoid augmentation, these two works focus on a fixed and constant delay value, whereas DEZ allows it to be random.

DEZ leverages the promising results obtained from tree-search-based learning and planning algorithms (Schrittwieser et al., 2020; Ye et al., 2021). Based on a Monte-Carlo tree search (MCTS), MuZero (Schrittwieser et al., 2020) and EfficientZero (Ye et al., 2021) perform multi-step lookahead and in-depth exploration of pertinent states. This is performed alongside latent space representations to effectively reduce the state dimensions. MuZero employs MCTS as a policy improvement operator by simulating multiple hypothetical trajectories within the latent space of a world model. On the other hand, EfficientZero builds upon MuZero's foundation by introducing several enhancements. These enhancements include a self-supervised consistency loss, the ability to predict returns over short horizons in a single step, and the capacity to rectify off-policy trajectories using its world model. Yet, none of these methods account for the delayed execution of prescribed decisions whereas DEZ takes advantage of the world model to infer future states and accordingly adapt decision-making.

3 Preliminaries

A discounted infinite-horizon MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$ where $\mathcal{S}, \mathcal{A}$ are finite state and action spaces, respectively, P is a transition kernel $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ mapping each state-action pair to a distribution over the state space, $r : S \times A \to \mathbb{R}$ is a reward function, $\mu \in \Delta_{\mathcal{S}}$ an initial state distribution, and $\gamma \in [0, 1)$ a discount factor diminishing the weight of long-term rewards.

At each decision step t, an agent observes a state $\tilde{s}_t$, draws an action $\tilde{a}_t$ that generates a reward $r(\tilde{s}_t, \tilde{a}_t)$, then progresses to $\tilde{s}_{t+1}$ according to $P(\cdot|\tilde{s}_t, \tilde{a}_t)$. Action $\tilde{a}_t$ follows some prescribed decision rule π_t, itself having different possible properties. More precisely, a decision rule can be Markovian (M) or history-dependent (H) depending on whether it takes the current state or the entire history as input. It can be randomized (R) or deterministic (D) depending on whether its output distribution supports multiple or a single action. This yields four possible classes of decision rules: HR, MR, HD, MD. A policy $\pi := (\pi_t)_{t\in\mathbb{N}}$ is a sequence of decision rules. The different classes of policies are denoted by $\Pi^{\text{HR}}, \Pi^{\text{MR}}, \Pi^{\text{HD}}, \Pi^{\text{MD}}$ and determined by that of their decision rules. When $\pi_t = \pi_0, \forall t \in \mathbb{N}$, the resulting policy is said to be stationary. We simply denote by Π the set of stationary policies.

The value function under a given policy $\pi \in \Pi^{\text{HR}}$ is defined as

$$v^{\pi}(s) := \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\tilde{s}_t, \tilde{a}_t)|\tilde{s}_0 = s\right],$$

where the expectation is taken with respect to the process distribution induced by the policy. Ultimately, our objective is to find $\pi^* \in \arg\max_{\pi\in\Pi^{\text{HR}}} v^{\pi}(s) =: v^*(s), \quad \forall s \in \mathcal{S}$. It is well-known that an optimal stationary deterministic policy exists in this standard MDP setting (Puterman, 2014).

4 Stochastic Execution-Delay MDP

In this section, we introduce stochastic execution-delay MDPs (SED-MDPs), which formalize stochastic delays and MDP dynamics as two distinct processes. We adopt the ED-MDP formulation of (Derman et al., 2021) that sidesteps state augmentation, and extend it to the random delay

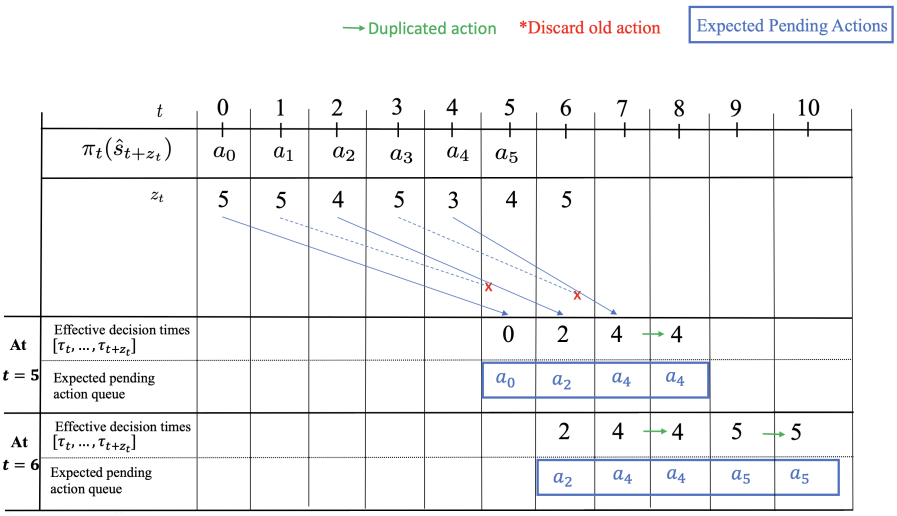


Figure 1: Pending queue resolution in a SED-MDP. The policy input $\hat{s}_{t+z_t}$ corresponds to the state inferred at t by a forward model (see Section 5). For clarity, effective decision times are shown for $t \in \{5, 6\}$ only.

case. With this perspective, we proceed in steps: (1) we define the SED-MDP framework with a suitable probability space; (2) we introduce the concept of *effective decision time*; (3) we prove the sufficiency of Markov policies to achieve optimal return in a SED-MDP.

A SED-MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma, \zeta, \bar{a})$ such that $(\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$ is an infinite-horizon MDP and $\zeta \in \Delta_{[M]}$ a distribution over possible delay values, where $[M] := \{0, 1, \cdots, M\}$. The last element $\bar{a} \in \mathcal{A}^M$ models a default queue of actions to be used in case of null assignment.[1] At each step t, a delay value $\tilde{z}_t$ is generated according to ζ; the agent observes $\tilde{z}_t$ and a state $\tilde{s}_t$, prescribes an action $\tilde{a}_t$, receives a reward $r(\tilde{s}_t, \tilde{a}_t)$ and transits to $\tilde{s}_{t+1}$ according to $P(\cdot|\tilde{s}_t, \tilde{a}_t)$. Unlike the constant delay case, the action $\tilde{a}_t$ executed at time t is sampled according to a policy that had been prescribed at a previous random time. This requires us to introduce the notion of effective decision time, which we shall describe later in Section 4.1.

For any $\pi \in \Pi^{\text{HR}}$, the underlying probability space is $\Omega = (\mathcal{S} \times [M] \times \mathcal{A})^\infty$ which we assume to be equipped with a σ-algebra and a probability measure $\mathbb{P}^\pi$. Its elements are of the form $\omega = (s_0, z_0, a_0, s_1, z_1, a_1, \cdots)$. For all $t \in \mathbb{N}$, let the random variables $\tilde{s}_t : \Omega \to \mathcal{S}$, $\tilde{z}_t : \Omega \to [M]$ and $\tilde{a}_t : \Omega \to \mathcal{A}$ be respectively given by $\tilde{s}_t(\omega) = s_t$, $\tilde{z}_t(\omega) = z_t$ and $\tilde{a}_t(\omega) = a_t$. Clearly, $\tilde{a} := (\tilde{a}_t)_{t\in\mathbb{N}}$ and $\tilde{z} := (\tilde{z}_t)_{t\in\mathbb{N}}$ are dependent, as the distribution of $\tilde{a}_t$ depends on past realizations of $\tilde{z}_{:t}$ for all $t \in \mathbb{N}$. On the other hand, in this framework, the delay process $\tilde{z}$ is independent of the MDP dynamics while it is observed in real-time by the agent. The latter assumption is justified in applications such as communication networks, where delay can be effectively managed and controlled through adjustments in transmission power. We finally define history random variables $(\tilde{h}_t)_{t\in\mathbb{N}}$ according to $\tilde{h}_0(\omega) = (s_0, z_0)$ and $\tilde{h}_t(\omega) = (s_0, z_0, a_0, \cdots, s_t, z_t), \quad \forall t \geq 1$.

4.1 Effective decision time

To deal with random delays, we follow (Bouteiller et al., 2020) and assume that the SED-MDP actuates the most recent action being available at that step. This implies that previously decided actions can be overwritten or duplicated. Fig. 1 depicts the queue dynamics resulting from successive realizations of the delay process. Concretely, at each step t, the delayed environment stores two distinct queues: one queue of M past actions $(a_{t-M}, a_{t-M+1}, \cdots, a_{t-1})$ and one queue of M past

[1] One may use a sequence of M distributions instead of deterministically prescribed actions, but we restrict to actions for notation brevity.

execution delays $(z_{t-M}, z_{t-M+1}, \cdots, z_{t-1})$. Then, we define the *effective decision time* at t as:

$$\tau_t := \max_{t' \in [t-M:t]} \{t' + z_{t'} \leq t\}. \tag{1}$$

It is a stopping time in the sense that τ_t is $\sigma(\tilde{h}_t)$-measurable. Notably, it also determines the distribution that generated each executed action: given a policy $\pi = (\pi_t)_{t\geq 0}$, for any step t, the action at time t is generated through the decision rule $\tilde{a}_t \sim \pi_{\tau_t}$. Intuitively, τ_t answers the question "What is the most recent action available for execution, given past and current delays"? Alternatively, we may define $e_t := t + z_t$ as the *earliest execution time* resulting from a decision taken at time t.

Example. Let $M = 5$ and assume that the following table gives the first realizations of decision rules and delays:

$\boldsymbol{t}$	0	1	2	3	4
$\boldsymbol{z_t}$	5	4	4	4	3
$\boldsymbol{e_t}$	5	5	6	7	7
$\boldsymbol{a_t}$	a_0	a_1	a_2	a_3	a_4

As a result, the effective decision time at $t = 5$ is given by $\tau_5 = 1 \cdot \mathbb{1}_{\{z_5>0\}} + 5 \cdot \mathbb{1}_{\{z_5=0\}}$. From the agent's perspective, the pending queue at time $t = 5$ depends on the delay value z_5:

- If $z_5 = 5$, then the pending queue is $[a_1, a_2, a_4, a_4, a_4]$.
- If $z_5 = 4$, then the pending queue is $[a_1, a_2, a_4, a_4]$.
- If $z_5 = 3$, then the pending queue is $[a_1, a_2, a_4]$.
- If $z_5 = 2$, then the pending queue is $[a_1, a_2]$.
- If $z_5 = 1$, then the pending queue is $[a_1]$.
- If $z_5 = 0$, then the pending queue is $[]$.

4.2 SED-MDP PROCESS DISTRIBUTION

We now study the process distribution generated from any policy, which will lead us to establish the sufficiency of Markov policies for optimality. To begin with, the earliest time at which *some* action prescribed by the agent's policy is executed corresponds to $\min\{z_0, z_1 + 1, \ldots, M - 1 + z_{M-1}\}$ or equivalently, to $\min\{e_0, e_1, \ldots, e_{M-1}\}$. From there on, the SED-MDP performs a switch and uses the agent's policy instead of the fixed default queue. Denoting $t_z := \min\{z_0, z_1 + 1, \ldots, M - 1 + z_{M-1}\}$, for any policy $\pi \in \Pi^{\text{HR}}$, we have:

$$\mathbb{P}^\pi\left(\tilde{a}_t = a | \tilde{h}_t = (h_{t-1}, a_{t-1}, s_t, z_t)\right) = \begin{cases} \mathbb{1}_{\bar{a}_t}(a) & \text{if } t < \min\{z_0, z_1 + 1, \ldots, z_t + t\}, \\ \pi_{\tau_t}(h_t)(a) & \text{otherwise.} \end{cases} \tag{2}$$

We can now formulate the probability of a sampled trajectory in the following theorem. A proof is given in Appendix A.1.

Theorem 4.1. *For any policy $\pi := (\pi_t)_{t\in\mathbb{N}} \in \Pi^{\text{HR}}$, the probability of observing history $h_t := (s_0, z_0, a_0, \cdots, a_{t-1}, s_t, z_t)$ is given by:*

$$\mathbb{P}^\pi(\tilde{s}_0 = s_0, \tilde{z}_0 = z_0, \tilde{a}_0 = a_0, \cdots, \tilde{a}_{t-1} = a_{t-1}, \tilde{s}_t = s_t, \tilde{z}_t = z_t)$$
$$= \mu(s_0) \left(\prod_{k=0}^{t_z-1} \mathbb{1}_{\bar{a}_k}(a_k) P(s_{k+1}|s_k, a_k)\right) \left(\prod_{l=t_z}^{t-1} \pi_{\tau_l}(h_l)(a_l) P(s_{l+1}|s_l, a_l)\right) \left(\prod_{m=0}^{t} \zeta(z_m)\right).$$

Extending the result of (Derman et al., 2021), we establish that the process distribution generated from any history-dependent policy can equivalently be generated from a Markov policy. The proof is given in Appendix A.2.

Theorem 4.2. *Let $\pi \in \Pi^{\text{HR}}$ a history dependent policy. For all $s_0 \in \mathcal{S}$, there exists a Markov policy $\pi' \in \Pi^{\text{MR}}$ that yields the same process distribution as π, i.e., $\mathbb{P}^{\pi'}(\tilde{s}_{\tau_t} = s', \tilde{a}_t = a | \tilde{s}_0 = s_0, \tilde{z} = z) = \mathbb{P}^\pi(\tilde{s}_{\tau_t} = s', \tilde{a}_t = a | \tilde{s}_0 = s_0, \tilde{z} = z)$, for all $a \in \mathcal{A}, s' \in \mathcal{S}, t \geq 0$, and $\tilde{z} := (\tilde{z}_t)_{t\in\mathbb{N}}$ the whole delay process.*

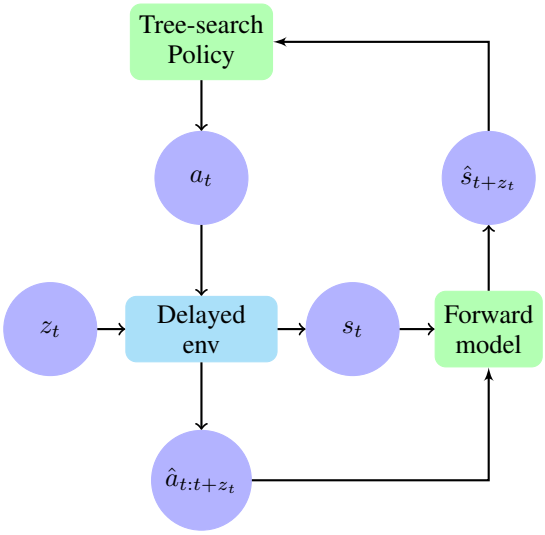
We can now devise policy training in the smaller class Π^{MR} without impairing the agent's return. The algorithm we propose next precisely does so.

5 STOCHASTIC-DELAY EFFICIENTZERO

We introduce a novel algorithm designed to address the challenges posed by stochastic execution delay. Drawing inspiration from the recent achievements of EfficientZero on the Atari 100k benchmark (Ye et al., 2021), we leverage its architectural framework to infer future states with high accuracy. Its notable sample efficiency enables us to quickly train the forward model. We note that recent advancements in model-based approaches extend beyond lookahead search methods such as MuZero and EfficientZero. Depending on the task in hand, alternatives such as IRIS (Micheli et al., 2022), SimPLE (Kaiser et al., 2019), or DreamerV3 (Hafner et al., 2023) exhibit distinct characteristics that may also be considered. Our approach is adaptable and can be seamlessly integrated with any of these algorithms.

Algorithm description. DEZ is depicted in Fig. 2. It adapts EfficientZero (Ye et al., 2021) to act and learn in environments with stochastic delays. At any time t, we maintain two queues of length M. One is the action queue of M previous realizations $[a_{t-M}, \ldots, a_{t-1}]$. The second one is the delay value queue observed for these M actions, $[z_{t-M}, \ldots, z_{t-1}]$. At inference, we observe state s_t and delay z_t, and aim to estimate the future state $\hat{s}_{t+z_t}$. To do so, we take the expected pending actions denoted by $[\hat{a}_t, \ldots, \hat{a}_{t+z_t-1}]$ and successively apply the learned forward model: $\hat{s}_{t+1} = \mathcal{G}(s_t, \hat{a}_t), \ldots, \hat{s}_{t+z_t} = \mathcal{G}(\hat{s}_{t+z_t-1}, \hat{a}_{t+z_t-1})$, using notations from Ye et al. (2021). We finally perform an MCTS search to output $a_t := \pi_t(\hat{s}_{t+z_t})$ and add a_t, z_t to the respective queues. On the other hand, the action executed in the delayed environment is a_{τ_t}. Since no future action can overwrite the first expected action $\hat{a}_t$, we note that $a_{\tau_t} = \hat{a}_t$. The system's reward is then stored to form transition (s_t, a_{τ_t}, r_t).

Figure 2: Interaction diagram between DEZ and the delayed environment



Pending actions resolution. Fig. 1 depicts the action and delay queues, together with the relation between effective decision times and pending actions. At each time step, the actual pending actions are calculated using the effective decision times from Eq. (1).

Future state prediction. To make accurate state predictions, we embrace the state representation network $s_t = \mathcal{H}(o_t)$ and the dynamics network $\hat{s}_{t+1} = \mathcal{G}(s_t, a_t)$ from EfficientZero (Ye et al., 2021). For simplicity, we do not distinguish between the observation o_t and the state representation s_t as do Ye et al. (2021). However, all network parts—forward, dynamics, value, and policy—operate in the latent space. Moreover, the replay buffer contains transition samples with states s_t, s_{t+1} in that same space. The pseudo-code in Algo. 1 depicts how Self-play samples episodes in the stochastic delayed environment.

Corrections in the replay buffer. In the original version of EfficientZero, episode transitions (s_t, a_t, r_t) are stored in game histories containing state, action, and reward per step. However, to insert the right transitions into the replay buffer, we post-process the game histories using all execution delays $\{z_t\}_{t=0}^{t=L}$ **ED:** what is L? , enabling us to compute effective decision times and store (s_t, a_{τ_t}, r_t) instead. More details about the episode processing can be found in Appx. B.

Learning a policy in a delayed environment. Action selection is carried out using the same MCTS approach as in EfficientZero. The output of this search denoted by $\pi_t(\hat{s}_{t+z_t})$ also serves as the policy target in the loss function. The primary alteration occurs in the policy loss, which takes the form:

$$\mathcal{L}_p = \mathcal{L}(\pi_t(\hat{s}_{t+z_t}), p_t(s_{t+z_t})).$$

ED: What is p_t? Although the MCTS input relies on a predicted version of s_{t+z_t}, this loss remains justified due to the exceptional precision exhibited by the dynamics model within the latent space. Other loss components remain unchanged.

6 EXPERIMENTS

The comprehensive framework we construct around EfficientZero accommodates both constant and stochastic execution delays. We consider the delay values $\{5, 15, 25\}$ as in (Derman et al., 2021) **ED:** is the full support in $\{5, 15, 25\}$ or are these max delay values? . For both constant and stochastic delays, we refrain from random initialization of the initial action queue as in (Derman et al., 2021). Instead, our model determines the first M actions. This is achieved through the iterative application of the forward and the policy networks. In practice, the agent observes the initial state s_0, infers the policy through $\bar{a}_0$, and predicts subsequent state $\hat{s}_1 = \mathcal{G}(s_0, \bar{a}_0)$. It similarly infers $\hat{s}_2, \ldots, \hat{s}_M$ and selects actions $\bar{a}_2, \ldots, \bar{a}_{M-1}$ through this iterative process.

EfficientZero sampled 100K transitions, aligning with the Atari 100K benchmark. Although our approach significantly benefits from EfficientZero's sample efficiency, the presence of delay adds complexity to the learning process requiring mildly more interactions – 130K in our case. Successfully tackling delays with such limited data is a non-trivial task.

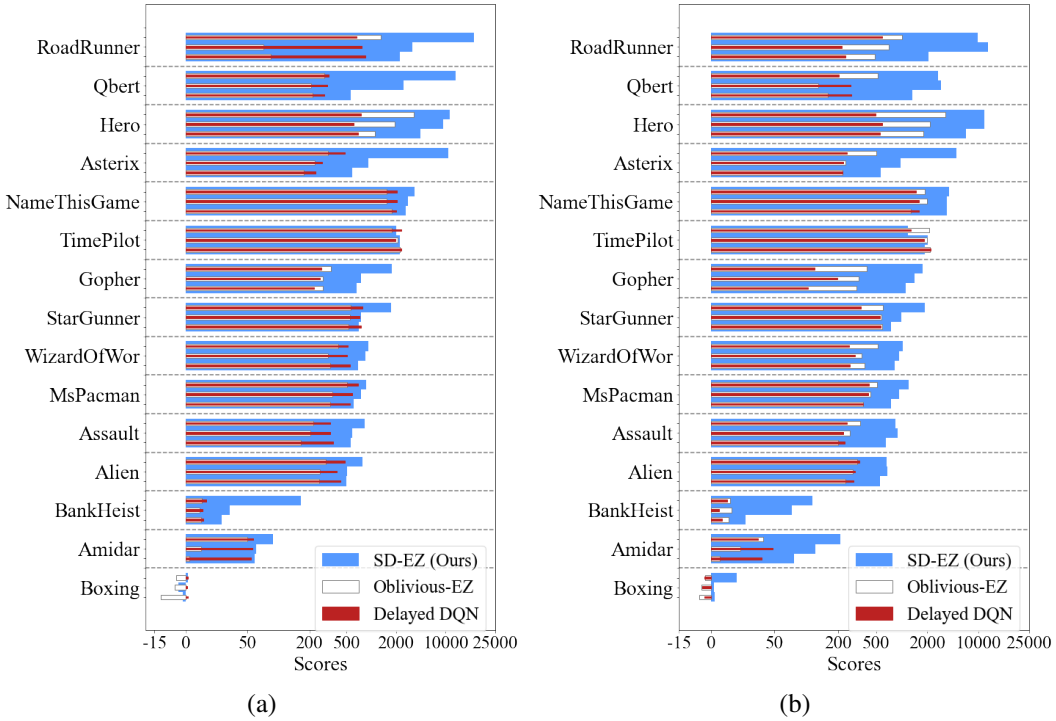


Figure 3: Average score on 15 Atari games and delays $M \in \{5, 15, 25\}$ over 32 test episodes per trained seed. Delays appear from low to high values for each game. (a) Constant delay value; (b) Stochastic delay value within $\{0, \cdots, M\}$.

To evaluate our theoretical findings, we subject DEZ to testing across 15 Atari games, each under both constant and stochastic delays. In either case, the delay value is revealed to the agent. We stress that the baseline scores are not consistent for the two types of delays we simulate. Indeed, on constant delays, Delayed-Q (Derman et al., 2021) tends to achieve higher scores than the oblivious version of EfficientZero, whereas Delayed-Q suffers more from stochastic delays than EfficientZero. As explained next, DEZ consistently achieves higher scores on both types of delays. A full summary of scores and standard deviations for our algorithm and baseline methods is presented in Appx. C.4, Tabs. 1 and 2.

6.1 CONSTANT DELAYS

We first analyze the scores produced by DEZ on constant delays and refer to Appx. C.1 for the convergence plots of all tested games.

Figure 3(a) shows that our method achieves the best average score in 39 of the 45 experiments. The oblivious version of EfficientZero confirms previously established theoretical findings (Derman et al., 2021, Prop. 5.2.): stationary policies alone are insufficient to achieve the optimal value. Given that Oblivious EfficientZero lacks adaptability and exclusively generates stationary policies, it is not surprising that it struggles to learn effectively in most games.

Our second baseline reference is Delayed-Q (Derman et al., 2021), acknowledged as a state-of-the-art algorithm to address constant delays in Atari games. However, it is worth noting that Delayed-Q is a DQN-based algorithm, and its original implementation entailed training with one million samples. Our observations reveal that within the constraints of the allotted number of samples, Delayed-Q struggles to achieve efficient learning. On the other hand, Delayed-Q benefits from its perfect forward model, provided by the environment itself. We observe that in 21 out of 45 experiments, Delayed-Q achieves an average score that is at least 85% of our method's result, even from the initial training iterations.

Importance of the forward model. Despite the reliance of Delayed-Q on the Atari environment as its forward model, this baseline exhibits significant vulnerabilities when faced with limited data. To address this challenge, we harness the representation learning capabilities of EfficientZero, allowing us to acquire state representations and dynamics through the utilization of its self-supervised rich reward signal, among other features. Our findings illustrate that when we effectively learn a precise prediction model, delays become more manageable. Yet, it is worth noting that increased randomness in the environment can result in larger compounding errors in the subsequent forward models.

6.2 Stochastic delays

For a more realistic scenario, we test our method on stochastic delays as well. For each Atari game, we use three stochastic delay settings. For each value $M \in \{5, 15, 25\}$, we use the following formulation of the delay z_t at time t:

$$z_0 = M$$

$$z_{t>0} = \begin{cases} \min(z_t + 1, M) & \text{with probability } 0.2, \\ \max(z_t - 1, 0) & \text{with probability } 0.2, \\ z_t & \text{otherwise.} \end{cases}$$

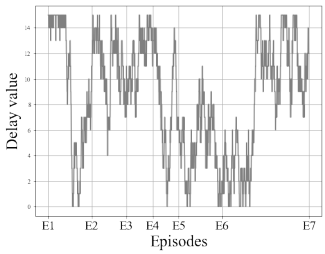


Figure 4: Random walk behavior of the stochastic delay across multiple episodes. No initial delay value is set at episode start. Here, the maximal delay is 15.

Note that the delay is never reinitialized to M when the episode terminates. Instead, the delays remain the same for the beginning of the next episode. By doing that, we do not assume an initial delay value and cover a broader range of applications.

We opt to avoid the augmentation approach in the baselines due to its inherent complexity. Incorporating the pending action queue into the state space was deemed infeasible due to its exponential growth, even in simple environments such as Maze (Derman et al., 2021). In our scenario, the length of the pending queue ranges from zero to M in cases involving stochastic delays. Consequently, the size of the augmented state space becomes $|\mathcal{S}| \times |\mathcal{A}|^M$. As the value of M increases, the order of magnitude of this quantity becomes overwhelming.

We report the average scores of DEZ and baselines for 15 Atari games in Figure 3(b), and refer to Appx. C.2 for convergence plots. Similarly to the constant case, we observe an impressive achievement, even slightly better, highlighting the added resilience of our method to stochastic delays. In 42 of 45 experiments, our method prevails with the highest average score. DEZ is additionally trained to incorporate very large delay values for evaluation in extreme scenarios. The performance in such cases is predominantly influenced by the specific characteristics of the environment. For instance, in Asterix, we observe that DEZ struggles to learn efficiently with delays of $M = 35$ and $M = 45$. Conversely, in Hero, our algorithm demonstrates comparable learning performance even with the largest delay value.

7 DISCUSSION

In this work, we formalized stochastic execution delay in MDPs without resorting to state augmentation. Although its analysis is more engaged, this setting leads to an intriguing insight: as in the constant delay case, we can restrict the policy search to the class of Markov policies and still reach optimal performance in the stochastic delay case. We introduced DEZ, a model-based approach that achieves state-of-the-art performance for delayed environments, thanks to an accurately learned world model. A natural extension of our approach would consider predicting next-state distribution instead of point prediction $\hat{s}_{t+1}$ to train a more robust world model and mitigate uncertainty in more stochastic environments.

DEZ heavily relies on a learned world model to infer future states. Devising a model-free method that addresses delayed execution while avoiding embedding remains an open question. DEZ also assumes delay values are observed in real-time, which enabled us to backpropagate the forward model the corresponding number of times. Ridding of this constraint would add another layer of difficulty because the decision time that generated each action would be ignored then. For deterministic execution delay, we may estimate its value. For stochastic delays, we could adopt a robust approach to roll out multiple realizations of delays and act according to a worst-case criterion.

Useful improvements may also be made to support continuous delays and get rid of drop and duplication of actions. In such a case, we may plan and derive continuous actions. Another modeling assumption we made was that the delay process is independent of the MDP environment. Alternatively, one may study dependent delay processes, e.g., delay values that depend on the agent's current state or decision rule. Such extension is particularly meaningful in autonomous driving where the agent state must be interpreted from pictures of various information complexities.

8 REPRODUCIBILITY

To further advance the cause of reproducibility in the field of Reinforcement Learning (RL), we are committed to transparency and rigor throughout our research process. We outline the steps undertaken to facilitate reproducibility: (1) Detailed Methodology: Our commitment to transparency begins with a description of our methodology. We provide clear explanations of the experimental setup, algorithmic choices, and the concepts introduced to ensure that fellow researchers can replicate our work. (2) Formal proofs are presented in the Appendix. (3) Open Implementation: To foster exploration and research into delay-related challenges in RL, we include the code as part of the supplementary material. We include a README file with instructions on how to run both training and evaluation. Our repository is a fork of EfficientZero (Ye et al., 2021) with the default parameters taken from the original paper. (4) Acknowledging Distributed Complexity: It is important to note that while we strive for reproducibility, the inherent complexity of distributed architectures, such as EfficientZero, presents certain challenges. Controlling the order of execution through code can be intricate, and we acknowledge that achieving exact-result replication in these systems may pose difficulties. In summary, our dedication to reproducibility is manifested through transparent methodologies, rigorous formal proofs, and an openly accessible implementation. While we recognize the complexities of distributed systems, we believe that our contributions provide valuable insight into delay-related issues in RL, inviting further collaboration within the research community.

REFERENCES

Eitan Altman and Philippe Nain. Closed-loop control with delayed information. *ACM sigmetrics performance evaluation review*, 20(1):193–204, 1992.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

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

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. January 2021.

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

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

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.

A Rupam Mahmood, Dmytro Korenkevych, Brent J Komer, and James Bergstra. Setting up a reinforcement learning task with a real-world robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4635–4640. IEEE, 2018.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

Ruth E Politi, Peter D Mills, Lisa Zubkoff, and Julia Neily. Delays in diagnosis, treatment, and surgery: Root causes, actions taken, and recommendations for healthcare improvement. *Journal of Patient Safety*, 18(7):e1061–e1066, 2022.

Martin L Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. October 2021.