Benchmarking Real-Time Reinforcement Learning

Pierre Thodoroff[*] PT440@CAM.AC.UK
Wenyu Li[*] WL414@CAM.AC.UK
Neil D. Lawrence NDL21@CAM.AC.UK

Abstract

Decision-making algorithms can require fast response time in applications as diverse as self-driving cars and minimizing load times of webpages. Yet, modern algorithms (deep reinforcement learning) are usually developed in scenarios where inference and training computational costs are ignored. This proposal aims to study reinforcement learning and control algorithms for real-time continuous control. In this scenario, the environment continuously evolves while actions are being computed by the agent (either in training or inference). The first goal is to provide a clear picture of the performance of modern algorithms modulated by their computational costs. The second goal is to identify the major challenges that arise when considering real-time environments to guide further research.

Keywords: Pre-registration, Machine Learning, Reinforcement Learning

1. Introduction

Recent advances in machine learning (deep learning) have been driven by models with significant computational costs (Goodfellow et al., 2016; Brown et al., 2020). Furthermore, the majority of the research community is focusing on maximizing accuracy, as reflected by the benchmarks (Deng et al., 2009; Mnih et al., 2013) which ignore computational costs. This trend leads to research improvements that are driven by increasing training data.[1]

However, in the real world, slow computations can lead to negative outcomes. For a self-driving car, the computation time of deciding when to brake can have a life-changing impact. In a real-time setting, the environment will keep evolving while the next action is being selected. This can create complications such as: 1) the action selected might not be relevant anymore, 2) the agent might enter an irreversible part of the environment (a car crash for example).

Starting in the 1960s, the control theory community considered mathematical frameworks to satisfy real-time constraints (Nelson, 1965; Rabin, 1963) that arise when doing control in a time-sensitive environment. The theory was considered for controllers in embedded devices with safety-critical applications such as space navigation (Hamlin, 1964). Formally, real-time control focuses on finding a controller guaranteeing the stability of the policy if inference is performed under a time threshold T. The need for real-time machine learning has also been identified in different areas of the field such as computer vision (Kiani Galoogahi et al., 2017; Li et al., 2020) or speech translation (Bangalore et al., 2012).

[*] Equal contribution
1. It has been suggested that the majority of the improvements of machine learning come from increased computational power and not algorithms (Sutton, 2019; Radford et al., 2018). This supports the need to consider the performance of an algorithm modulated by its energy use as suggested by (Welling, 2018).

In recent years, reinforcement learning algorithms combined with flexible non-linear models (e.g. deep neural networks) have demonstrated an ability to solve complex environments (Mnih et al., 2013; Schulman et al., 2017) at the costs of high sample and computational complexity. Deep reinforcement learning algorithms usually interact with a simulator without time constraints on inference or training. This creates a gap between these solutions and many real-world applications where action selection can be temporally and computationally constrained. The majority of existing benchmarks ignore this discrepancy. The motivation of this work is to analyze the performance of modern algorithms in the context of real-time control.

The aim is to address the following two goals:

Goal 1: Analyze the performance trade-off between computationally complex and simple algorithms in the context of real-time decision-making. This analysis can be used to define which algorithm to use depending on the time requirements as well as the complexity of the environment.

Goal 2: Exploratory analyses of the algorithmic issues arising when working in real-time. This can help guide the research community in the future on the most pressing issues to be addressed before modern algorithms can be deployed in time-sensitive applications.

We consider a continuous control benchmark to compare the performance of control and RL algorithms on robotic environments with varying difficulty. The real-time aspect is emulated by running the agent's algorithm and the environment together in parallel (illustrated in Figure 1). The difficulty of the tasks can be scaled by increasing or slowing down the speed of time in the environment's simulation. We consider several tracks with varying amounts of computational power available to underline the sensitivity of RL algorithm to computational considerations. The goal is to compare a wide array of methods from control and reinforcement learning using simple (linear) and flexible models (non-linear). This real-time benchmark enables a new way to compare algorithmic decisions such as model-free versus model-based RL which have different computational profiles for training and inference.

2. Related work

The two leading frameworks for sequential decision-making environment are control theory (Lee and Markus, 1967) and reinforcement learning (Sutton and Barto, 2018). Both attempt to model sequential decision-making, however, control theory has a strong emphasis on models, sample efficiency, and theoretical guarantees. In contrast, reinforcement learning is more focused on learning from experience and data.

Real-time control considers the problem of creating a policy with guaranteed performance if the output is produced within a given fixed time window T (Kwon and Pearson, 1980). Several works have expanded the scope of those algorithms to accommodate for uncertainty over the time horizon T and the dynamics (Cucu-Grosjean, 2013; Santinelli and Cucu-Grosjean, 2015). Theoretically, the performance is guaranteed by modelling the control problem as applying the action with a deterministic delay. Mathematically, this is often done using functional ordinary differential equations (Hale, 1971), which are a generalization of ODE where the inputs can be several timesteps before. A similar strategy was developed in Reinforcement learning to model for the delay of applying an action (Chen et al., 2021; Bouteiller et al., 2020; Derman et al., 2021; Ramstedt and Pal, 2019).

Practically, the delay in all the works mentioned above is implicitly modelled by looking for a policy that will perform well even if the action is applied with a specific delay. For example, this can be done using policy gradient where the rewards correspond to the evaluation of the delayed policy. This in contrast to explicit modelling of the delay that could predict the state in the future using a dynamical model and optimize with respect to this prediction.

The key aspect in the RL works mentioned above is that the delay applied is independent of the computation performed. It is usually a fixed quantity used to demonstrate the property of the algorithms. The proposed research pushes this one step further by experimentally making the delay proportional to the computational costs of the algorithm used. Practically, this means that any computation spent by an algorithm will result in a delay for the action to be taken. This enables a new way of comparing algorithms where performance is modulated by computation costs.

3. Methodology

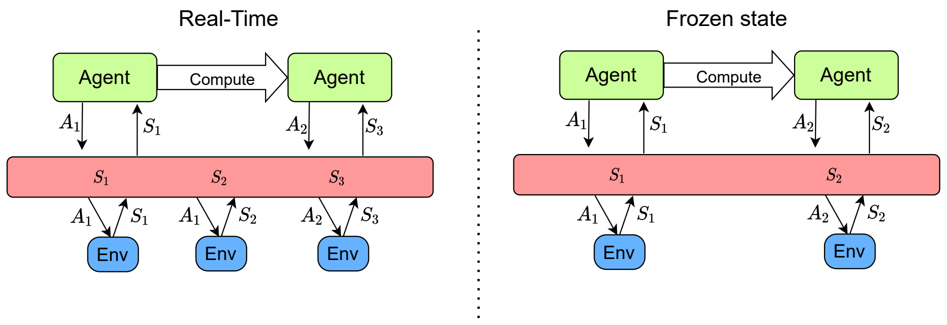

Figure 1: Diagram of the interactions between the environment and the agent. S represent the state and A the action. The left diagram represent the real-time interaction with an environment. As you can see the action A_1 is being repeatedly applied until the new action A_2 is computed. The right diagram represents how RL algorithms are trained when the state is frozen during action selection.

We consider a Markov Decision Process defined as a tuple $\{S, A, T, R\}$ where S represent the set of states, A the set of actions, $T(s'|s, a) : S \times A \to S$ the transition function and $R(s, a) : S \times A \to \mathbb{R}$ the reward function.

In continuous control, both the state ($s \in S$) and action ($a \in A$) are usually continuous and belongs to $\mathbb{R}^N$. The goal is to accumulate rewards over N time steps and maximize the following quantity:

$$\mathbb{E}_a[\sum_{i=0}^{i=N} \gamma^t R(s, a)|s_0] \quad a \in A, s \in S. \tag{1}$$

where s_0 is the starting state. Theoretically, the appropriate definition of this process should consider continuous time due to the desire to benchmark algorithms in *real-time*. We present the discrete version due to the fact that almost all control algorithms are concretely solving discrete version of the process and the mathematical complexity introduced by the continuous time version is not necessary for this paper. Experimentally, by taking a time discretization steps small enough, it is possible to emulate continuous time. Theoretically, an equivalence between discrete and continuous time MPD can developed (Serfozo, 1979). The real-time aspect of the environment is emulated using sticky actions. This means that until the agent decides on a new action, the previous one will be applied, a natural concept in continuous control (illustrated in Fig 1).

3.1. Control models

From a reinforcement learning and control perspective, the challenge is to accumulate as many rewards as possible with the caveat that any computation performed by the algorithm will translate into time lost in the execution of this action. We consider several axes of comparison between decision-making algorithms. The methods considered and their characteristics are detailed in Table 1.

Table 1: Table describing the characteristics of the methods considered. MF model-free, MB model-based, H hybrid model-based and policy, LM linear model, NLM non-linear model, RR real-time robustness.

	MF	MB	H	LM	NLM	RR
Random policy (Mania et al., 2018)	x			x		
SAC (Haarnoja et al., 2018)	x				x	
Real-time RL (Ramstedt and Pal, 2019)	x				x	x
MPC linear dynamics		x		x		
Muzero (Schrittwieser et al., 2020)			x		x	

The first axis considers the computational costs of running the decision-making algorithm. There exists a long-standing debate between methods that include an explicit model of the dynamics of the environment (model-based RL, control algorithm) and those that don't (model-free). While it is easy to argue that a model of the environment should improve generalization and performance, it often comes at higher computational costs. As an extreme, performing Monte-Carlo roll-out in chess may lead to an optimal answer, however, computing the full tree is intractable. In those cases, algorithms often cut the tree by using a policy (model-free algorithm). In this paper, we call those methods hybrid. Practically, model-based methods display sample efficient training but slow inference time due to the necessary roll-out. Lower policy training time can lead to more frequent policy updates and better data collections. In contrast, model-free methods will display quick inference time which can lead to more timely response and improved stability.

The second axis concerns the computational costs of the learning algorithm involved. While training a deep neural network could enable higher performance on complex locomotion tasks, it incurs significant computational costs (inference and training) potentially hindering performance. It will be interesting to compare its performance to simpler models (linear regression) that are computationally efficient but have less flexibility.

Finally, we also include two methods developed to adapt to the real-time setting, one from RL (Ramstedt and Pal, 2019) and one from control (Zeilinger et al., 2014).

3.2. Software considerations

While the algorithmic side of things is essential, as illustrated earlier in this proposal, computation and algorithm are intricately connected. It is essential to study the software specifications of the problem.

The first one concerns the simulation of the real-time environment. One option is to run an environment in a separate thread (CPU) which will result in a simple interface, however, the step frequency will be varying based on the computational speed of the machine considered (this may be controllable). This could render benchmark between different research groups complicated. The second option would be to use internal clocks to measure the time spend on computations and pass this information onto the environment. The issue with this approach is that it creates a synchronous paradigm where state and actions can not be easily shared asynchronously. The decision on how to implement the environment is not fixed yet, but the second option seems easier for reproducibility purposes.

4. Experimental protocol

This proposal focuses on continuous control environments due to their direct applicability for real-world problems. Furthermore, continuous control is strictly a more general problem than turn-based environments (Mnih et al., 2013) with a time limit because the environment keeps evolving while one runs the computations.

The first step is to benchmark 4 popular control environments based on the Mujoco simulator (Todorov et al., 2012): inverted pendulum, half-cheetah, hopper, and humanoid (Ellenberger, 2018–2019). The environments considered range in difficulty from the inverted pendulum (1 degree of freedom) to the humanoid (24 degrees of freedom). The range of difficulty considered should be wide enough to allow deep reinforcement learning algorithms to outperform simpler models at the expense of higher computational costs.

The second set of environments attempts to solve manipulation tasks such as fetch-pickandplace and handmanipulategg. Finally, we also consider a vision-based environments with high-dimensional input, where the goal is to drive a car on a track (Ramstedt and Pal, 2019). This will enable to get a better sense of when deep learning architecture are superior to linear approximators even though the computational costs is greater.

4.1. Metrics

The main metric of interest is the cumulative reward described in Equation 1. However, as illustrated in (Henderson et al., 2018), the variance of the return can vary greatly depending on the seed used. This underlines the brittleness of deep RL methods and the need for a thorough evaluation of the variance profile of the reward before drawing any conclusion. The main strategy is to run the algorithm on multiple seeds (10+) and analyze the performance profile of the return (Henderson et al., 2017; Jordan et al., 2020). Hypothesis testing can be used to compare the relative performance of different algorithms. Based on each environment, it can be interesting to display a 2 dimensional scatter graph displaying the performance of each algorithm based on its computational cost. This could be used to guide a decision on which algorithm to use in which context.

4.2. Discretization factor

There exists two ways to simulate real-time interaction in a discrete system. The first one consists of using dynamic time steps in the physics simulator. Most simulators use sampling-based solutions to solve the complex ODE underlying the environment considered and varying time steps could be passed. The issue with this method is that it involves re-compiling the environment every time the time step is changed. The other method is to set the timestep in the environment to be small and run the environment quickly. We will implement the latter one due to its simplicity.

The difficulty of the environment can be scaled by increasing or slowing down the environment's loop. Practically, 1 second of compute can be equal to 0.1 or 0.01 seconds spent in the environment.

4.3. Compute resources available

Practical applications of decision-making algorithms can be resource-constrained. For example, in embedded devices, the decision algorithm may only have access to one CPU. We assume one thread (CPU) is reserved for simulating the environment. Based on the amount of processing power available, we consider three experimental tracks.

Track 1: We first train a policy in a simulator without time constraints and then evaluate its performance in real-time. This track focuses on the delay induced by the computational costs of inference. There are no computational constraints on the inference mechanism, any number of GPU's or CPU's can be used in this track. This represents an important benchmark for deploying controllers in the real-world.

Track 2: The setup is similar to track 1, except that the controller is restricted to 1 CPU for inference. This is meant to represent the challenge of restricted available compute in embedded device. In this scenario, it will be interesting to study the performance of deep learning architectures due to the potentially slow inference on CPUs.

Track 3: The agent has access to CPUs and GPUs but both training and inference result in delay for action selection. The challenging aspect of this track is that most algorithms are not tuned or developed for this setting. For most deep reinforcement learning algorithms, how often to alternate between training and inference is not clear. Furthermore, in DRL, the training time is so large that it will fail many episodes before ending training. However, this is the expected behaviour. The goal is to display how some of those algorithms fail in this regime and how some may succeed (real-time MPC). The goal is to analyze performance as the algorithm are currently being used. While further tuning of each algorithm could improve the performance it is not the focus of this benchmark.

4.4. Reproducibility

One of the challenging aspects of this benchmark is that reproducibility may be difficult to achieve. We discuss below some of the issues that will be encountered and the steps towards mitigating reproducibility concerns.

- Different hardware (CPU, GPU) will lead to different inference times and performance. The results reported will only be valid for a specific compute engine and experiments will need to be re-run in a different environment. Docker will be used to make sure the software environment remains the same and improve cross-platform portability.
- Due to the priority process of commonly used OS (Linux), background processes can interrupt and delay computations of our scripts. We take a statistical approach where the interruptions should be averaged out over multiple runs. We will also perform an analysis of the variance of the computation time of each step to confirm that the results are coherent. Finally, we will attempt to minimize any other CPU work happening on the machine to minimize preemption. This can also be done by increasing the priority of the python script.
- We will record the computation time of the various steps and report all the relevant metrics in a publicly accessible wandb dashboard.

5. Discussion and Conclusion

To summarize, the goal of this work is to give an objective assessment of the performance of real-time decision-making algorithms in continuous control. The second motivation is to analyze the challenging aspect of real-time decisions such as to underline interesting research questions for the community. One interesting aspect of this benchmark is that it raises other research questions such as:

- When computing the next action, how does an agent process a new incoming state or reward? Should the agent be able to interrupt the currently running computation and implement a priority system?
- How much time should an agent spend on inference (state or environment-dependent)?
- How to allocate time between training and inference?

6. Documented Modifications

Algorithm: For some algorithms (Muzero and MPC), a tuned working open-source version of the algorithms were not available. An updated list of the algorithms considered can be found in Table 2. Muzero is replaced with MBPO (Schrittwieser et al., 2020) which features similar computational characteristics due to the Hybrid model-based model-free policy. PETS (Chua et al., 2018) is replacing MPC as a simulation-based method.

Table 2: Updated table describing the characteristics of the methods considered. MF model-free, MB model-based, H hybrid model-based and policy, LM linear model, NLM non-linear model, RR real-time robustness.

	MF	MB	H	LM	NLM	RR
ARS (Mania et al., 2018)	x			x		
SAC (Haarnoja et al., 2018)	x				x	
PPO (Schulman et al., 2017)	x				x	
Real-time RL (Ramstedt and Pal, 2019)	x				x	x
PETS (Chua et al., 2018)		x		x		
MBPO (Schrittwieser et al., 2020)			x		x	

Environments: The benchmark is performed on Mujoco as opposed to the proposed PyBullet. Mujoco became open-source after its acquisition by DeepMind addressing the criticism of reproducibility. Furthermore, Mujoco is more widely used than Pybullet which makes it easier to find working implementations and tuned hyper-parameters. The second change considers the environments considered for visual control. The performance of several algorithms in the proposed self-driving environment was unstable. We replaced the self-driving environments with pixel-based extensions of Mujoco robotics tasks using the DMcontrol library (Tunyasuvunakool et al., 2020) and benchmark a model-based (planet (Hafner et al., 2019b)) and hybrid model (dreamer (Hafner et al., 2019a)).

Real-time implementation: During the pre-registration, we considered two ways of emulating real-time, one based on asynchronously running the policy and the environment in parallel. The second involved measuring the inference time and applying the corresponding delay in the environment. The first proposed solution suffered from high variance due to other computational processes running in the background and instead we implemented the second one.

Compute resource available: We removed the third track that considers training a model online while performing inference. The performance displayed in this category was deemed uninteresting as the training time for all those algorithms is considerably too long for any interesting behaviour to emerge. However, we add several experiments in the resource-constrained setting by emulating embedded devices using CPU throttling (see section 7.1).

7. Results

In this section, we present the results of RL algorithms in real-time continuous control. First, we discuss the inference time of modern algorithms on a variety of hardware. Then, we analyze the robustness of RL algorithms to inference delays due to computation time. Finally, we present a computation-adjusted performance analysis that details the best performing algorithms based on available hardware in real-time control.

7.1. Inference time

To ensure stable reproducible results, the inference time of each algorithm is calculated in a docker container. For each algorithm, we perform the inference sequentially 1000 times and take the median time to avoid outliers skewing the results. We also made sure that no other significant process was running on the server at the time.

To simulate the computational challenge of an embedded device, we throttle the number of CPU cycles used by the Docker container to varying degrees. As a reference point, modern CPUs (and the one used here) run around 2.5 billion cycles per second in contrast to an Arduino that will only run 20 million. Throttling cycles per second is not a perfect tool to simulate an embedded device due to other important components such as caching, ram and efficiency of CPU instructions. However, it provides some preliminary insights into the degradation of performance that happens on embedded devices. In Figure 2, 0.1 CPU refers to a maximum of 10% of the original CPU cycle per second (250M)[2].

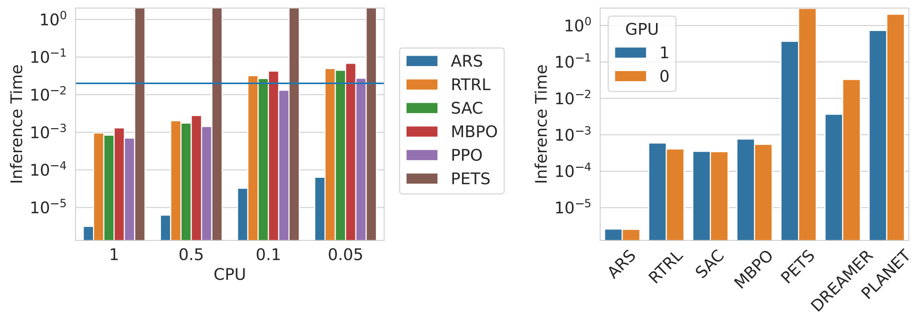

Figure 2: Inference time using varying compute infrastructure (CPU and GPU). The blue line represents a usual discretization factor in Mujoco environment. We can observer linear models (ARS) have low inference time compared to deep learning model and simulation based algorithm (PETS). Using a GPU is beneficial on image-based environment (DREAMER, PLANET) but provides negligible benefits on low-dimensional environments.

As displayed in the barplot 2, the computation time of linear models such as ARS is significantly faster than any other algorithms. In contrast, the inference time of a simulation-based algorithm (PETS) is significant and impractical for any real-time problem. The use of

2. More information on how this throttling is performed through the CFS scheduler can be found https://docs.docker.com/config/containers/resource_constraints/.

a GPU for low-dimensional problems (standard Mujoco tasks) does not provide significant speedup over the use of CPUs. However, for image-based versions of the control tasks, algorithms such as Planet and Dreamer benefit from a substantial speedup using GPUs. PLaNet is also simulation-based algorithm (in contrast to DREAMER) which explains the significant inference time.

7.2. Robustness to real-time

In this section, we study the robustness of RL algorithms to the delay introduced by computations. Practically, we first train algorithms in the standard-setting where the inference delay is ignored except for RTRL where a delay of 1 timestep is applied as proposed by the authors (Ramstedt and Pal, 2019). We evaluate the performance of the algorithms by progressively delaying the action decision which is implemented as a repeat of the previous action. Each environment has a standard time discretization which we lowered by a factor of 5 to give more fine-grained results. The discretization for RTRL remained the same due to its non-markovian structure (the previous action is passed as an input). Lowering the discretization factor severely affected performance.

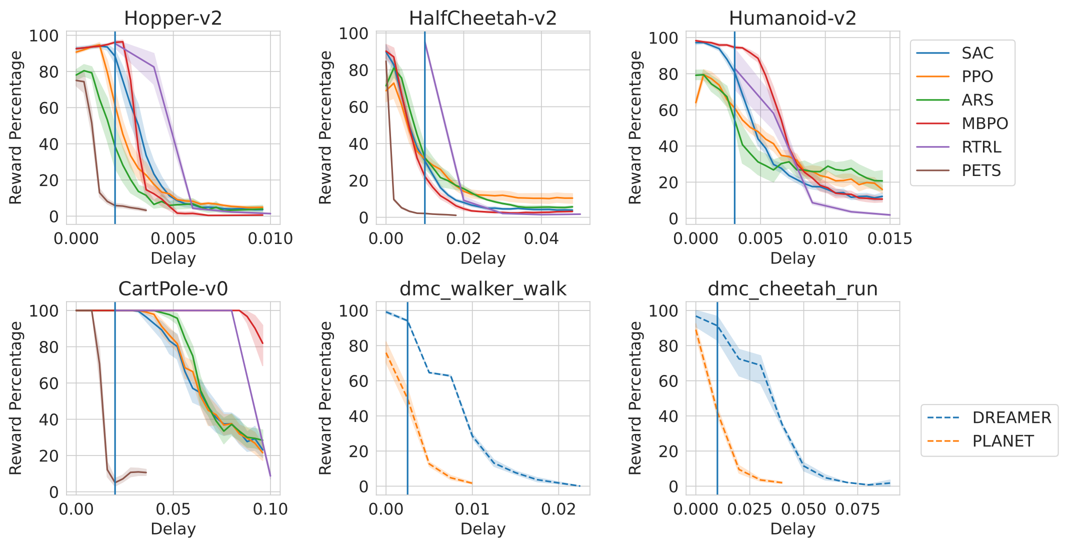

Figure 3: Degradation of performance on continuous control environments with varying amounts of inference delay. The vertical blue bar represents one timestep during training. RTRL and MBPO seems to be slightly more robust to inference delay than other algorithms.

In Figure 3, we illustrate the robustness of RL algorithm to time delay by displaying the decrease in performance for each algorithm percentage-wise. RTRL appears to be the most resilient algorithm for two reasons. One, the previous action is passed as input to the model, and two, the model is by default trained in a fixed real-time setting where a delay of one-time step (environment dependent) is applied. The second-best performing algorithm appears to be MBPO which shares a similar underlying policy as SAC, PPO and RTRL.

The difference is that a model of the dynamics of the environment is used to further train the model offline. Figure 3 suggests that MBPO is more resistant to time-delay supporting the general hypothesis that model-based algorithms tend to generalize better. All the results presented in Figure 3 are averaged over 20 seeds except PETS, DREAMER and PLANET which was too computationally expensive (N=5). The variance of the returns on those algorithms was found to be relatively low. We also trained all the algorithms on locomotion tasks such as Pusher and Fetch-PickAndPlace. However, none of the algorithms considered managed to solve the environments, so the results are not displayed in the main part of this paper.

7.3. Delay-adjusted performance

We now analyze the performance of algorithms modulated by their computational use. Practically, for a given CPU budget, we calculate the inference time and corresponding performance for this given delay (using the data from the previous section). This enables us to to compare algorithms depending on the computational power available. To enhance the quality of the visualization of the results, we increase the temporal granularity by linearly interpolating the CPU computation time obtained in section 7.1.

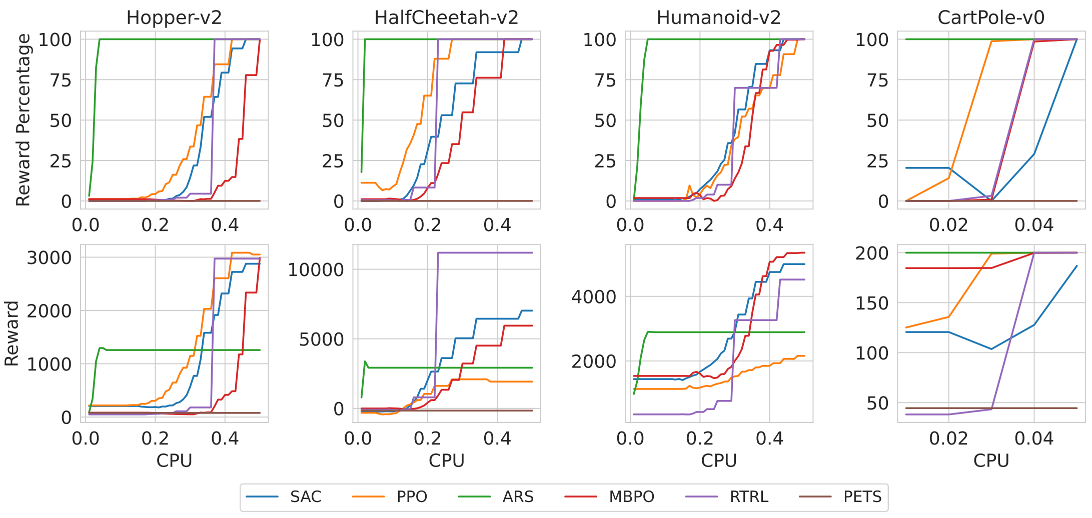

Figure 4: Performance adjusted by the inference delay on varying amount of CPU. The top row shows the performance degradation when varying the amount of CPU. The bottom rows displays the reward obtained by each algorithm. The performance of linear model (ARS) outperforms complex algorithms under resource constraint but under-performs otherwise. Simulation-based algorithm performs poorly due to inference time. All the other algorithm perform similarly underlying the importance of considering inference time when developing RL algorithms.

In Figure 4 we present the performance of each algorithms under varying computational constraints. ARS is resilient to limited compute due to the light computational requirements of linear models. For most environments, if the CPU constraint is under 0.2, ARS performs better than all the other models. For reference, an Arduino has can only run 20 million cycle per seconds (0.01 CPU). In contrast, simulation-based algorithm such as PETS suffer very strongly from the delay induced by their slow inference process. The performance of other algorithms such as PPO, SAC, RTRL and MBPO are rather similar due to their similar computational costs. MBPO appears to slightly outperform other algorithm due to its real-time robustness displayed in Figure 3. The time-step for CartPole-v0 is quite large (0.01) which means that most algorithms are able to perform well even with a limited access to CPU (0.05). We also analyze the performance of image-based algorithm adjusted for inference delay. PLANET has a large inference time (see Figure 2) both on CPU and GPU which would result in a delay of roughly 300 timesteps. In contrast, the inference time of DREAMER is within real-time requirements (0.5 timestep with GPU and 5 timesteps with CPU).

8. Discussion and conclusion 2.0

Delay-invariant models: When testing the performance of algorithms in real-time, we changed the discretization factor to allow for finer granularity. However, this induces a distribution shift in the environment. For example, an algorithm could learn to overshoot a target because it won't be able to change action in the next second. Most RL algorithms are trained in this unrealistic setting where inference is instantaneous, whereas in practice, they should be trained according to a pre-defined inference delay. The delay unfortunately depends on the hardware available, which means either this needs to be known in advance, or one could design algorithms that are adaptive to the inference time. For example, continuous time models often take the discretization factor as input to the model and adapt their prediction based on it. A similar strategy should be used for decision-making algorithms that adapt predictions based on inference time and available hardware. Taking this one step further, a simulation-based algorithm could have adaptive inference time based on the state of the environment.

Emulation of simulation-based algorithms: As displayed in the experiments, simulation based algorithms suffer the most from real-time requirements. One interesting approach is to learn an emulation model (policy network) based on this simulation offline and deploy the fast emulator in practice. A similar strategy has been proposed for MPC controllers in control theory (Wang et al., 2021).

Conclusion: In this benchmark, we demonstrate the importance of considering real-time control due to the major impact inference time has on performance (bigger than architectural decision). Different RL algorithms display significant differences in computational profiles (model-based versus model-free) underlining the importance to consider inference delay when developing decision-making algorithms. Furthermore, there exists a discrepancy in the data used during training and testing due to inference time. This opens up new research directions for RL algorithms that either model this delay (continuous-time models) or train the algorithm with the hardware constraint in mind.

Acknowledgements: This research was generously funded by the Engineering and Physical Sciences Research Council (EPSRC) and the Alan Turing Institute. The computing infrastructure was provided by Google through the Google Research Credit program. We would like to thank the organizer of the pre-register workshop for a particularly constructive review process. Finally, we thank Markus Kaiser, Eric Meissner, Christian Cabrera-Jojoa and Andrei Paleyes for helpful and insightful discussion on this paper.

References

Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 437–445, 2012.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame Polytechnique Montreal Christopher Pal Mila, and Polytechnique Montreal Jonathan Binas. REINFORCEMENT LEARNING WITH RANDOM DELAYS. Technical report, Polytechnique Montréal, 2020.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128, aug 2021. ISSN 18728286. doi: 10.1016/j.neucom.2021.04.015.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Liliana Cucu-Grosjean. Probabilistic Real-Time Systems. Technical report, INRIA France, 2013.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

Benjamin Ellenberger. Pybullet gymperium. https://github.com/benelot/pybullet-gym, 2018–2019.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019b.

Jack K Hale. Functional differential equations. In *Analytic theory of differential equations*, pages 9–22. Springer, 1971.

JE Hamlin. A general description of the national aeronautics and space administration real time computing complex. In *Proceedings of the 1964 19th ACM national conference*, pages 12–201, 1964.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. Evaluating the performance of reinforcement learning algorithms. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4962–4973. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/jordan20a.html.

Hamed Kiani Galoogahi, Ashton Fagg, Chen Huang, Deva Ramanan, and Simon Lucey. Need for speed: A benchmark for higher frame rate object tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1125–1134, 2017.

Woosuk Kwon and A Pearson. Feedback stabilization of linear systems with delayed control. *IEEE Transactions on Automatic control*, 25(2):266–269, 1980.

Ernest Bruce Lee and Lawrence Markus. Foundations of optimal control theory. Technical report, Minnesota Univ Minneapolis Center For Control Sciences, 1967.

Mengtian Li, Yu-Xiong Wang, and Deva Ramanan. Towards streaming perception. In *European Conference on Computer Vision*, pages 473–488. Springer, 2020.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1805–1814, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Edward A Nelson. A working definition of real-time control. Technical report, RAND CORP SANTA MONICA CALIF, 1965.

Michael O Rabin. Real time computation. *Israel Journal of Mathematics*, 1(4):203–211, 1963.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI*, 2018.

Simon Ramstedt and Christopher Pal. Real-time reinforcement learning. *arXiv preprint arXiv:1911.04448*, 2019.

Luca Santinelli and Liliana Cucu-Grosjean. A probabilistic calculus for probabilistic real-time systems. *ACM Transactions on Embedded Computing Systems*, 14(3):1–30, apr 2015. ISSN 15583465. doi: 10.1145/2717113. URL https://dl.acm.org/doi/10.1145/2717113.

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.

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

Richard F Serfozo. An equivalence between continuous and discrete time markov decision processes. *Operations Research*, 27(3):616–620, 1979.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13:12, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dmcontrol: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Songda Wang, Tomislav Dragicevic, Gustavo Figueiredo Gontijo, Sanjay K Chaudhary, and Remus Teodorescu. Machine learning emulation of model predictive control for modular multilevel converters. *IEEE Transactions on Industrial Electronics*, 68(11):11628–11634, 2021. doi: 10.1109/TIE.2020.3038064.

Max Welling. Intelligence per Kilowatthour, Aug 2018. URL https://www.youtube.com/watch?v=5DbBQDoBNYc.

Melanie N. Zeilinger, Davide M. Raimondo, Alexander Domahidi, Manfred Morari, and Colin N. Jones. On real-time robust model predictive control. *Automatica*, 50(3):683–694, 2014. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2013.11.019. URL https://www.sciencedirect.com/science/article/pii/S0005109813005360.

Appendix A. Appendix

The CPUs used for inference were Intel Xeon E5 v4 (Broadwell E5). The GPU used is an NVIDIA Tesla T4.

The codebase used to train the algorithm were used from either well-tested repository or published by the authors themselves.

- ARS: https://github.com/ray-project/ray
- RTRL: https://github.com/rmst/rtrl
- SAC, PPO: https://github.com/openai/spinningup
- MBPO, PETS, PLANET: https://github.com/facebookresearch/mbrl-lib
- DREAMER: https://github.com/danijar/dreamerv2

All the hyper-parameters used were the one recommended by the the libraries. The training performance can be visualized in a wandb dashboard https://wandb.ai/pierthodo/RTDM_train, the inference time https://wandb.ai/pierthodo/RTDM_inference and the robustness to real time https://wandb.ai/pierthodo/RTDM_performance. The dockerfile used to train the algorithm can be found https://github.com/pierthodo/benchmark_real_time_RL.

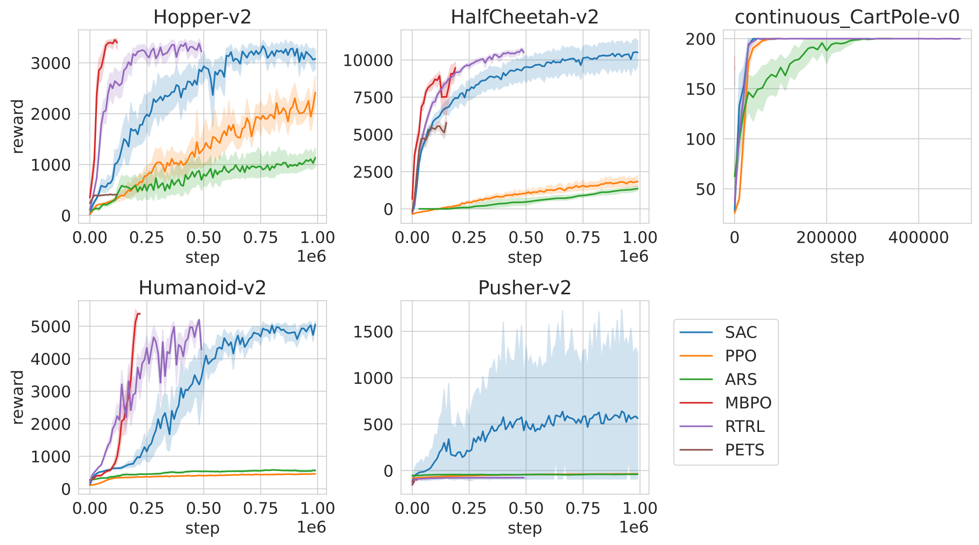

Figure 5: Training performance of RL algorithms on Mujoco environments.