Property-Driven Evaluation of RL-Controllers in Self-Driving Datacenters

Arnav Chakravarthy, Nina Narodytska, Asmitha Rathis, Marius Vilcu
VMware
Palo Alto, CA, USA
`chakravartha,nnarodytska,asmithar,mvilcu@vmware.com`

Mahmood Sharif
VMware, TAU
Tel Aviv, Israel
`sharifm@vmware.com`

Gagandeep Singh
VMware, UIUC
Champaign, IL, US
`gasingh@vmware.com`

Abstract

Reinforcement learning-based controllers (RL-controllers) in self-driving datacenters have evolved into complex dynamic systems that require continuous tuning to achieve higher performance than hand-crafted expert heuristics. The operating environment of these controllers poses additional challenges as it can often change significantly, thus the controllers must be adapted to new external conditions. To obtain trustworthy RL-controllers for self-driving datacenters, it is essential to guarantee that RL-controllers that are trained continuously in these changing environments behave according to the designer's notions of reliability and correctness. Traditionally, RL-controllers are evaluated by comparing their reward function statistics. However, that does not capture all desired properties, e.g., stability, of the controller. In this work, we propose enhancing the evaluation criteria for RL-controllers with a set of novel metrics that quantify how well the controller performs with respect to user-defined properties. We leverage formal methods for computing our novel metrics. Thus, our work makes a step forward toward improving trustworthiness of RL-controllers. We show that these metrics are useful in evaluating a standalone controller or in comparing multiple controllers that achieve the same reward.

1 Introduction

In this paper we describe the Aria Graph AI project at VMware that is the instantiation of VMware's long-term vision of self-driving data centers that help customers manage their datacenters automatically [VMware(2021)]. To deal with inherently dynamic systems, Aria Graph AI develops tools and techniques that can be continuously optimized and can adapt to exogenous factors to achieve better performance under current conditions. Designing and updating controllers manually to perform optimization is tedious, requiring substantial expertise and therefore does not scale well to the fast-changing demands of client applications running in such datacenters. To enable automated control and adaptation, Aria Graph AI project leverages reinforcement learning (RL) [Goodfellow et al.(2016), Sutton and Barto(2018), Kiran et al.(2020), Yu et al.(2021)], which has been previously successfully applied to outperform traditional techniques in many systems applications, like networking, cloud management, and congestion control [Dethise et al.(2019), Kazak et al.(2019), Mao et al.(2017), Jay et al.(2018)]. In particular, Aria Graph AI controller is designed to adjunct data-center resources usage based on observable performance metrics.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Lack of metrics for assessing the behavior of RL-controllers on unseen inputs. While RL-based adaptive controllers in self-driving data centers yield performance benefits, several new challenges about their trustworthiness arise. RL-based controllers often employ neural networks, which are black-box decision procedures. Therefore, it is hard to interpret or explain their decision-making process to the user in an intuitive manner. Moreover, in cases when the controller behaves in an unexpected manner, it is hard to troubleshoot and fix the root cause of such behavior. Another challenge comes from the continuous adaptation of controllers. Undoubtedly, one of the benefits of RL-controllers is that they can be continuously tuned using newly collected data. In practice, to implement such an adaptation, new models are built periodically to adapt to the changes in the environment. One important question here is how we can compare these new models before deployment and find a good balance between fitting to immediate trends while ensuring generalizability [Yan et al.(2020)]. A classic approach to compare controllers is to use a reward function as an evaluation metric [Sutton and Barto(2018)]. Additionally, Aria Graph AI monitors certain metrics and behavioral patterns of controllers on known workloads. However, we argue that while such metrics are practically useful, they still do not provide a reliable assessment of how the model will perform on unseen data 'in the wild'. This is because the predictions of neural networks can be unreliable on novel inputs even though they are similar to those in the test set [Szegedy et al.(2013)]. Therefore, it is essential to design metrics based on formal guarantees of RL controller behavior on novel samples before deployment.

This work: property driven evaluation of RL-controllers. We focus on the problem of comparing RL-based controllers that are trained to control a dynamic system. We propose extending the evaluation criteria for an RL-controller with a set of novel metrics based on formal methods that quantify how well the controller performs with respect to user-defined properties on unseen inputs. Namely, we consider *robustness* and *monotonicity* metrics of an RL-based controller. This approach allows us to take advantage of domain-specific knowledge about the intended behavior of the decision-making mechanism. Our second contribution is to propose an efficient approach to compute these evaluation metrics. We leverage recent developments in certifying neural networks, based upon abstract interpretation [Cousot and Cousot(1977)], to compute these metrics [Singh et al.(2019a), Gehr et al.(2018), Singh et al.(2019b)].

2 Reinforcement learning

We provide an overview of the RL-based framework for the self-driving datacenter scenario.

We start with a description of the Aria Graph AI optimization system and its RL-controller. We assume that the state of the system is described by a vector of features $s = (f_1, \ldots, f_n)$ which captures knowledge about the current status of the environment, e.g. performance metrics. An RL-controller π takes the state s as input and produces the next action a as the output, where the space of possible action values depends on a particular application. The action is executed by the system, e.g. the cache level is changed. Given a state and an action, the environment, which is an uncontrollable element in the framework, moves the system from the current to the next state. A dataset for training and testing the controller typically consists of a set of traces that are representative of the system behavior. Each trace is a sequence describing the set of states encountered by the system while moving from an initial state to a final state. The controller is implemented as a fully-connected, deep neural network. Trace collection for training, designing a reward function, and designing a training procedure depends on the concrete application.

In production phase, Aria Graph AI periodically trains its controller based on the traces that it collects from customer environments. Every time a new controller is trained, a decision needs to be made on whether to use the newly trained controller or keep the previous one. Note that we might also consider controller candidates from previous training steps.

The main question we address here is *how we can meaningfully choose which controller will perform 'better' in the next period by defining metrics to assess its performance on unseen inputs.* As we discussed in the introduction, the traditional way to select the best candidate controller is to use reward function statistics over the test set. However, these metrics do not provide any insight into the performance of the controller on the novel inputs that it will encounter when running in the customer environment. It is possible that two controllers have similar reward statistics but one has more desirable behavior with respect to user-defined properties, like robustness or monotonicity, on unseen inputs than the other. We show this case in our evaluation in Section 4.1 and Section 4.2.

3 Evaluation metrics

We presented two user-defined properties that specify the expected behavior of the controller on a potentially infinite set of unseen inputs.Using these properties, we build new metrics for assessing the trustworthiness of RL controllers. Note that our framework is not limited to these two properties and can be easily extended to support other domain-specific properties. Due to lack of space, we focus on the robustness property here. The monotonicity property is described in Appendix A where we also outline ways to combine these metrics.

Robustness property. Our first metric evaluates the stability of the prediction of an RL-controller based on the popular robustness property [Singh et al.(2019a), Gehr et al.(2018), Singh et al.(2019b)]. Robustness is essential for ensuring trustworthiness in many safety-critical applications, so developing tools and techniques to evaluate robustness is extremely important. Note that even for applications that are not safety-critical, it is highly undesirable to work with an unstable controller as instability can accumulate and, especially in the adaptive training loop, can eventually drive the system to bad performance. We recall that the RL-based controller π takes features that describe the state $s = (f_1, \dots, f_n)$ and outputs the action $a = \pi(s)$. Consider an input state s' and another input state s^* that are close, in some norm, to each other so that $||s^* - s'|| \leq \epsilon$. Here $\epsilon \in \mathbb{R}$ is a parameter provided by the user for identifying close states. We use the L_∞-norm in our work. Note that the number of states s^* close to s' can be infinite. All states s^* close to s' are usually not available in either the training or the testing set of the controller. We say that the controller is locally robust in a state s' iff for all states s^* such that $||s^* - s'|| \leq \epsilon$, the following property holds: $||\pi(s^*) - \pi(s')|| \leq \delta$, where $\delta \in \mathbb{R}$ is specified by the user for identifying acceptable deviation in the prediction of the controller on inputs close to s'. Informally, the controller is locally robust if small perturbations of its inputs do not lead to large changes in the corresponding outputs. Next, we consider how to use the above property to define a robustness metric. First, given a training dataset, we consider it as a set of individual states, i.e., we ignore the temporal dependency between the states and actions in traces, and subsample a set K of states. The robustness metric of a controller π is defined as a ratio of states where π is locally robust to the total number of sampled states: $r(\pi) = (\sum_{s \in K} robust(\pi, s, \epsilon, \delta))/|K|$, where $robust(\pi, s, \epsilon, \delta) = 1$ iff the robustness property holds around s and 0 otherwise. Finally, given multiple controllers, we can compare them using the robustness score, the higher the score, the more stable the controller output.

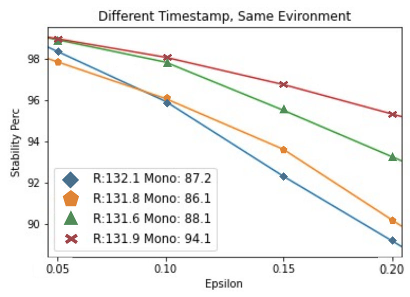
Example 1 *In the context of a continuous controller for the Aria Graph AI project, we perturb certain relevant input features by $+/-5\%$ based on expert knowledge. We say that the controller is locally robust for a state s' if the output is within $+/-5\%$ of $\pi(s')$. So, in this use-case $\epsilon = \delta = 0.05$. For a discrete controller, we have $\epsilon = 0.05, \delta = 0$, i.e., we check if the discrete action taken by the controller does not change even after perturbation.*

4 Experimental evaluation

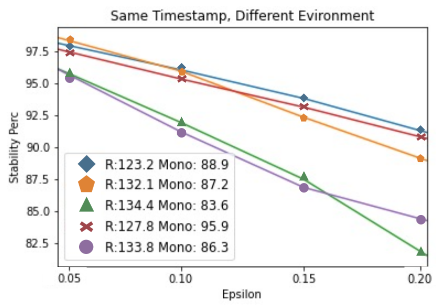
In this section, we present our experimental evaluation on several datasets and several discrete RL-controllers in Aria Graph AI. We perform two sets of experiments. We use synthetic data from a simulator to evaluate controllers. In all experiments, the controllers are trained to optimize the reward metric. In the first set, we consider four RL-based controllers for datacenter management that are trained at different time steps under workloads from the same distribution. The second experiment assesses how five controllers trained at the same step behave in different environments. The goal of these experiments is to investigate the additional value that the robustness and monotonicity metrics bring for analyzing RL-based controllers.

4.1 Scenario A: Stable Workloads

We consider four different RL-controllers trained at different times, about one week apart. We assume that the workloads were stable during this period. Table 1 and Figure 1a show our results. In Table 1, we name the controllers from 1 to 4 and attach a color label to match the corresponding line in Figure 1a. For example, '1(green, $\triangle$)' means the first controller that has green legend in Figure 1a. We show the value of the reward function on the test set in the last column for all four controllers in Table 1. As these workloads are similar, controllers have similar reward values. The last two columns



(a) Scenario A



(b) Scenario B

Figure 1: The robustness metric computed as a function of ϵ

RL-agent	$m(\pi)$	$r(\pi)$ $\epsilon = 0.05$	Reward
1 (green, $\triangle$)	88.1	99.0	131.5
2 (blue, $\diamond$)	87.2	98.6	132.1
3 (orange, ⬠)	86.1	98.0	131.8
4 (red, $\times$)	94.1	99.0	131.9

Table 1: Results for monotonicity and robustness properties for 4 agents (Scenario A)

RL-agent	$m(\pi)$	$r(\pi)$ $\epsilon = 0.05$	Reward
1 (orange, ⬠)	87.2	98.6	132.1
2 (blue, $\diamond$)	88.9	98.1	123.2
3 (green, $\triangle$)	83.6	96.1	134.4
4 (red, $\times$)	95.9	97.7	127.7
5(purple, $\bigcirc$)	86.3	96.1	133.8

Table 2: Results for monotonicity and robustness properties for 5 agents (Scenario B)

show the values of the monotonicity and robustness scores. Note that robustness scores are also quite similar for these controllers if $\epsilon = 0.05$. However, the monotonicity score varies significantly. The '4(red, $\times$)' controller has the best monotonicity score, which gives us an indication that the way it behaves meets our expectation on monotone traces the best. Figure 1a plots the robustness score as a function of the perturbation parameter ϵ for each controller. We recall that from Table 1 we observed that at $\epsilon = 0.05$, all controllers exhibit robust behavior. However, Figure 4 shows that as ϵ grows, the '4(red), $\times$' controller has significantly better score than the rest. Note that it also achieves the highest monotonicity score. While it is the second-best controller in terms of reward-metrics, our additional metrics show that it is more well-behaved compared to the other controllers.

4.2 Scenario B: Variable Workloads

Next, we compare five controllers trained at the same point in time under different workloads. Table 2 and Figure 1b show our results. Consider Table 2 first. In this case, we can see that the reward metric can be used to filter out two out of five controllers as their reward scores are significantly lower. However, distinguishing between the remaining three controllers is more difficult. Note that their monotonicity and robustness scores with $\epsilon = 0.05$ look similar. Let us look at Figure 1b that is plotted similar to Figure 1a. It can be seen that two controllers with the highest reward become unstable as ϵ grows. This gives a clear winner: '1(orange, , ⬠)' controller, which is only the third-best controller in terms of pure reward, but it does not lose much to '3(green, $\triangle$)' and '5(purple, $\bigcirc$)' controllers.

5 Conclusion and future work

In this work, we take a step towards obtaining trustworthy RL-controllers for self-driving datacenters. We propose novel evaluation metrics that reflect a designer's domain knowledge about an RL-controller's expected behavior. Namely, we propose to focus on two critical domain properties: robustness and monotonicity. We formalize these properties and propose the corresponding metrics. Our second contribution is to show that using state of the art techniques for the certification of neural networks, we can compute these metrics efficiently. Finally, our experimental results on Aria Graph AI show that the proposed metrics provide additional insights into the behavior of a controller compared to standard evaluation techniques. In the future, we are planning to evaluate these metrics in other system domains that take advantage of RL agents, investigate an effect of recent RL algorithms that allow us to train more robust controllers [Foret et al.(2021)] and look into other domain properties that can help a designer better understand the behavior of their RL-controller.

References

[Cousot and Cousot(1977)] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California) *(POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. `https://doi.org/10.1145/512950.512973`

[Dethise et al.(2019)] Arnaud Dethise, Marco Canini, and Srikanth Kandula. 2019. Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents. In *Proceedings of the 2019 Workshop on Network Meets AI & ML* (Beijing, China) *(NetAI'19)*. Association for Computing Machinery, New York, NY, USA, 29–36. `https://doi.org/10.1145/3341216.3342210`

[Foret et al.(2021)] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware Minimization for Efficiently Improving Generalization. In *International Conference on Learning Representations*. `https://openreview.net/forum?id=6Tm1mposlrM`

[Gehr et al.(2018)] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. 3–18. `https://doi.org/10.1109/SP.2018.00058`

[Goodfellow et al.(2016)] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

[Ivanov et al.(2019)] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proc. ACM International Conference on Hybrid Systems: Computation and Control, HSCC*. ACM, 169–178.

[Jay et al.(2018)] Nathan Jay, Noga H. Rotman, P. Brighten Godfrey, Michael Schapira, and Aviv Tamar. 2018. Internet Congestion Control via Deep Reinforcement Learning. `https://doi.org/10.48550/ARXIV.1810.03259`

[Kazak et al.(2019)] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. 2019. Verifying Deep-RL-Driven Systems. In *Proceedings of the 2019 Workshop on Network Meets AI & ML* (Beijing, China) *(NetAI'19)*. Association for Computing Machinery, New York, NY, USA, 83–89. `https://doi.org/10.1145/3341216.3342218`

[Kiran et al.(2020)] Bangalore Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick Pérez. 2020. Deep Reinforcement Learning for Autonomous Driving: A Survey. *CoRR* abs/2002.00444 (2020). arXiv:2002.00444 `https://arxiv.org/abs/2002.00444`

[Mao et al.(2017)] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 197–210. `https://doi.org/10.1145/3098822.3098843`

[Mohr et al.(2021)] Stefanie Mohr, Konstantina Drainas, and Juergen Geist. 2021. Assessment of Neural Networks for Stream-Water-Temperature Prediction. `https://doi.org/10.48550/ARXIV.2110.04254`

[Singh et al.(2019a)] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019a. An Abstract Domain for Certifying Neural Networks. *Proc. ACM Program. Lang.* 3, POPL, Article 41 (jan 2019), 30 pages. `https://doi.org/10.1145/3290354`

[Singh et al.(2019b)] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019b. Boosting Robustness Certification of Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. `https://openreview.net/forum?id=HJgeEh09KQ`

[Sutton and Barto(2018)] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. `http://incompleteideas.net/book/the-book-2nd.html`

[Szegedy et al.(2013)] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. `https://doi.org/10.48550/ARXIV.1312.6199`

[VMware(2021)] VMware. 2021. Aria Graph AI. `https://www.vmware.com/products/aria.html`

[Yan et al.(2020)] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 495–511. `https://www.usenix.org/conference/nsdi20/presentation/yan`

[Yu et al.(2021)] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. 2021. Reinforcement Learning in Healthcare: A Survey. *ACM Comput. Surv.* 55, 1, Article 5 (nov 2021), 36 pages. `https://doi.org/10.1145/3477600`

Appendix

A Evaluation metrics (missing parts)

A.1 Monotonicity property

Next, we describe the monotonicity property. Here we consider a sequence of system states $t = (s^1, s^2, \ldots, s^n)$ of length n. Suppose there is an ordering over these states, so we can determine that $s^i \leq s^j$ for any two states s^i, s^j. We say that a sequence is ordered if states in the sequence respect the ordering $s^1 \leq s^2 \leq \ldots \leq s^n$. Our monotonicity property is defined for an ordered sequence of states. Consider the behaviour of an RL-controller over a time period n. During the execution, we obtain a sequence of states t and the controller's predictions for each state $a = (a^1, a^2, \ldots, a^n)$. We say that the RL-controller is monotone for t iff its predictions $a = (a^1, a^2, \ldots, a^n)$ for the states in t are ordered, i.e., $a^1 \leq a^2 \leq \ldots \leq a^n$ or $a^1 \geq a^2 \geq \ldots \geq a^n$. We let the user decide whether the controller output should be increasing or decreasing. We consider ordering between the states defined using a subset of features. The top row of Figure 2 shows an example of an ordered trace based on two features. Here, the input features are monotonically increasing and the output of the controller is also increasing.

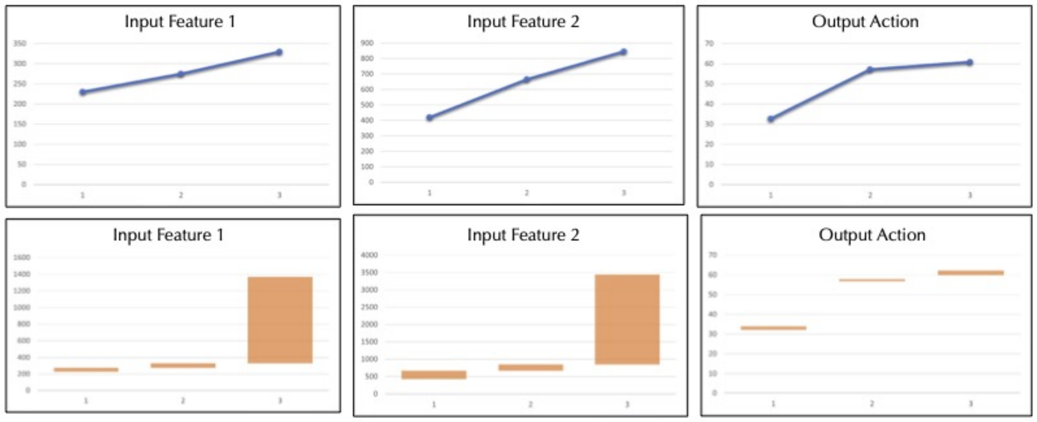


Figure 2: The top row shows an example where the output of the controller is monotonic with respect to an ordered trace. The ordering between states is defined based on two input features. The bottom row shows that the controller output also satisfies the monotonic property on a set of traces around the original trace.

Given a set of set traces T and corresponding controller's actions at each state, we first extract a subset of traces T^O where states are ordered. We call these traces ordered traces. Next, we define the monotonicity metric defined as a ratio of monotone traces to the total number of ordered traces.

$$m^b(\pi) = (\sum_{t \in T^O} monotone(\pi, t))/|T^O|.$$

$monotone(\pi, t) = 1$ iff the prediction of π is monotonic for the states in t otherwise it is 0. To reason about families of traces, we extend the monotonicity metric on individual traces in the following way. We first transform a trace into an infinite set of traces. While our approach is applicable in the general case, for ease of exposition, we assume that the state contains one feature f. So, an ordered trace is $f^1 \leq f^2 \leq \ldots \leq f^n$. We form a sequence of intervals $[f^1, f^2], [f^2, f^3], \ldots, [f^n, f^{n+1}]$ with $f^{n+1} \geq f^n$. We choose $f^{n+1} = min(f^n, 0.95 \cdot f_{max})$ in our experiments where f_{max} is the maximum value of the feature in the training dataset. Given an interval $[f^i, f^{i+1}]$, we compute $\pi([f^i, f^{i+1}])$ (we describe the computation in appendix A.3) which is another interval $[a^l_i, a^u_i]$. We say that a controller π is monotonically increasing for a set of traces if for each pair of consecutive actions a_i, a_{i+1}, we have that either $a^l_i \leq a^l_{i+1}$ or $a^u_i \leq a^u_{i+1}$ holds. The property for a controller to be monotonically decreasing is defined similarly. We use this to define our monotonicity metric as:

$$m(\pi) = (\sum_{t \in T^O} monotone(\pi, S[t]))/|T^O|,$$

where $S[t]$ is a set of traces around t as we described above and $monotone(\pi, S[t])) = 1$ iff the controller output satisfies the monotonicity property over $S[t]$ and 0 otherwise. The higher the monotonicity metric, the more the controller meets the user expectation about maintaining monotonic output.

Discussion on monotonicity: We note that the monotonicity property might not be suitable for all types of controllers. However, if we can define a notion of an order on inputs and outputs of a controller, e.g., if the controller inputs and outputs are numerical values, we envision that this property can be well-defined and useful. Another point to consider is how to define an ordering of states in a trace. It might not be possible to define ordering over all features of a state. Therefore, we need to identify a subset of the features that have a natural ordering.

Example 2 *We consider states with two features in Figure 2. The top row of Figure 2 shows an example of a trace of length three where these features and the predicted action are monotonically increasing. The bottom row shows the set of traces generated from the original trace based on our transformation described above. The last column of the bottom row shows how the controller output evolves over time for the set of traces. As one can see, both lower and upper bounds are increasing, therefore the monotonicity property holds.*

A.2 Comparing controllers

Given two RL-based controllers π_1 and π_2, let $w(\pi_i)$ be the reward value of a controller on the test dataset, $r(\pi_i)$ be its robustness score, and $m(\pi_i)$ be the monotonicity score of π_i. We compute an extended evaluation score of π_i as:

$$score(\pi_i) = \alpha \times w(\pi_i) + \beta \times r(\pi_i) + \gamma \times m(\pi_i), \quad (1)$$

where α, β, γ are hyperparameters. We envision that the reward value should be given the highest weight, while β and γ can be used to tie-break between high-performance models. We can use the extended score above to assess the trustworthiness of the controller. For example, we might find a controller with high reward and low robustness score or monotonicity score. In this case, the controller might be taking aggressive decisions to boost its reward but its output does not meet user-defined behavior. A recommendation would be to re-train the controller with different initializations. While we focus on robustness and monotonicity metrics, we point out that other domain-specific metrics that are useful to quantify a controller behavior can be handled similarly. We do rely on the user's domain expertise to write metrics specifications.

A.3 Computing evaluation metrics

We leverage formal methods for computing the robustness and monotonicity scores defined on an infinite set of inputs for a controller. We note that the computation of the exact controller output for an infinite set of inputs is an undecidable problem [Ivanov et al.(2019)]. Therefore we compute an overapproximation of the exact output using the popular DeepPoly abstraction [Singh et al.(2019a)] and use the overapproximation for computing the metrics. As a result of overapproximation, it can happen that we are unable to prove a robustness property when it actually holds. However, whenever we prove a robustness property, it is guaranteed to hold over the entire input set. For the monotonicity property, the property holds if for every pair of consecutive actions a_j, a_{j+1}, the approximated bounds satisfy $a_i^u \leq a_{i+1}^l$. We can prove that the property does not hold on the exact output if we find a pair of consecutive actions a_i, a_{i+1}, for which the approximated bounds satisfy $a_{i+1}^u < a_i^l$. For the remaining cases, the property may or may not hold on the exact output. We note that collecting the controller output on a finite set of samples from the input set computes an underapproximation of the true controller output. The underapproximation cannot formally prove that the property holds in any of the cases. This limitation makes sampling unsuitable for computing the robustness and monotonicity metrics in our framework.

B Related work

Recently, formal verification of neural networks has gained a lot of attention [Dethise et al.(2019), Singh et al.(2019a), Gehr et al.(2018), Singh et al.(2019b)]. These techniques can be broadly partitioned into two classes: complete and incomplete methods. The complete techniques [6,16,20] provide exact results, i.e., if a property does not hold then these methods provide a counterexample violating the property. Otherwise, we have a certificate that the property holds. However, complete verification is expensive and undecidable for general network architectures and properties. In contrast, the incomplete methods may incorrectly fail to prove a property [Singh et al.(2019a), Gehr et al.(2018), Singh et al.(2019b)]. However, these methods are computationally more efficient and applicable for verifying general network architectures and properties. That is why we chose the DeepPoly method [Singh et al.(2019a)] from this class as it is currently the most precise and scalable verification method. Using property-based evaluation metrics has not been widely explored to the best of our knowledge. The recent work of [Mohr et al.(2021)], makes similar observations regarding the need of additional criteria for evaluating neural networks deployed in environmental science.