Towards Verifying Monotonicity and Robustness Properties for Domain-Specific DRL Algorithms

Mohammad Zangooei [1] **Mina Tahmasbi Arashloo** [1] **Raouf Boutaba** [1]

Abstract

Deep Reinforcement Learning (DRL) algorithms have demonstrated significant performance benefits compared to traditional heuristics in various systems and networking applications. However, concerns surrounding their explainability, generalizability, and robustness pose obstacles to their widespread deployment. In response, the research community has explored the application of formal verification techniques to ensure the safe deployment of DRL algorithms.

This study focuses on DRL algorithms with discrete numerical action spaces, which are commonly utilized in systems and networking domains but have been under-explored when it comes to formal analysis. We investigate two fundamental properties: monotonicity and robustness. We employ formal methods to verify these properties beyond the confines of the training set, thereby enhancing generalization and validating robustness. To explore the effectiveness and tractability of the proposed verification approach, we implement a proof of concept using Gurobi and present a case study involving two systems and networking DRL algorithms.

1. Introduction

Deep Reinforcement Learning (DRL) algorithms have garnered significant interest within the systems and networking communities, as they have demonstrated significant performance benefits compared to traditionally handcrafted heuristics. For example, DRL-based methods have been applied to adaptive video streaming (Mao et al., 2017), job scheduling in data processing clusters (Mao et al., 2019), network congestion control (Lan et al., 2019), device placement for distributed machine learning (Bojja Venkatakrishnan et al., 2019), network planning (Zhu et al., 2021), buffer management (Wang et al., 2022), and resource management in multi-tenant environments (Alcaraz et al., 2022).

Despite the performance benefits of DRL-based approaches, they suffer from explainability, generalizability, and robustness issues which cast doubts on their deployment in real-world systems and networks (Hamadanian et al., 2022; Eliyahu et al., 2021). Specifically, a DRL model's decision is based on a Deep Neural Network (DNN) that operates as a black box and does not provide any explanations of the rationale behind its behavior. As such, there is a risk of unexpected and/or undesirable actions in scenarios that are different from those present in the training set (Yan et al., 2020; He et al., 2022). Moreover, empirical evidence suggests that DNNs are susceptible to vulnerabilities in robustness, as minor perturbations in the input can yield substantial alterations in the output. Such behavior significantly erodes the reliability and trustworthiness of the DRL algorithms (Chakravarthy et al., 2022; Dethise et al., 2021).

To address these concerns, the research community has started to explore the use of formal verification to guarantee the safe deployment of DRL algorithms in the systems and networking domain (Eliyahu et al., 2021; Amir et al., 2021; Dethise et al., 2021). These efforts formally model the DRL algorithm and the desirable properties and perform a rigorous analysis of the input space to either prove that a property always holds or identify concrete counter-examples in which the property is violated. Such counter-examples indicate that the DRL model's training has not been sufficient; either the training process should be prolonged or more data needs to be added to the training set. Additionally, counter-examples can also help identify situations where manual intervention is needed to override the DRL algorithm (Eliyahu et al., 2021).

While formal methods tools and techniques are known to have limited scalability, their application to DRL algorithms used in systems and networking has shown promising results. This is mainly because the DRL algorithms in this context are typically trained on structured input, in contrast to raw data (e.g., raw pixels in computer vision). As

[1]David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada. Correspondence to: Mohammad Zangooei <mzangooei@uwaterloo.ca>.

Presented at the 2^{nd} Workshop on Formal Verification of Machine Learning, co-located with the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA., 2023.

such, the sizes of the DNNs are much smaller, making them more tractable to analyze. Moreover, it is much easier to express the desirable properties of the DRL algorithm in terms of constraints over the model's input and output variables (Eliyahu et al., 2021).

In this study, we focus on DRL algorithms with a discrete numerical action space as they come up in numerous systems and networking applications and have not been systematically studied in prior work. Examples include but are not limited to selecting bitrate for video streaming (Mao et al., 2017), determining congestion window size (Lan et al., 2019), allocating a number of executors to a Spark job in a cluster (Mao et al., 2019), and managing resource allocation among tenants in a multi-tenant environment (Alcaraz et al., 2022). We focus on two main categories of properties:

- **Monotonicity** properties specify that the output should increase/decrease as one of the input features increases. Such properties are common in system and network components for which DRL algorithms are being proposed. For instance, suppose a DRL algorithm uses network packet loss rate as an input feature to decide the congestion window size. If the loss rate in the network increases, we expect the DRL algorithm to decrease the congestion window size. We encode monotonicity properties by comparing the model's output in two different scenarios that differ only in one feature.
- **Robustness** properties specify that small differences in the input should not lead to significant variations in the output. For instance, minor fluctuations in the packet loss rate should not result in substantial changes to the congestion window size. We follow a similar approach for encoding *robustness* – we compare the model's output in two distinct scenarios that are within a bounded L_∞ distance from each other.

We use two algorithms, one for resource allocation among tenants in a multi-tenant environment and one for selecting bitrate for video streaming, as case studies. Specifically, we specified one robustness and two monotonicity properties for each algorithm and verify the formulated properties for one algorithm. These case studies help demonstrate the effectiveness and feasibility of our proposed verification approach while pointing out potential scalability limitations associated with larger DNN sizes.

2. DRL Algorithms with Numerical Action Space

A DRL algorithm interacts with a system in a sequential manner. It makes decisions based on the current state of the system in each iteration, aiming to maximize the accumulated sum of rewards it receives from the system in future iterations (Sutton & Barto, 2018). The system's state is represented by a vector of features. In the system and networking domain, these features can include performance metrics such as throughput, latency, packet loss rate, resource utilization (CPU, memory), or power consumption. This state vector is provided as input to the DRL algorithm, which generates the next action to be taken.

One of the core components of a DRL algorithm is its DNN which consists of several layers connected through activation functions. The output layer of the DNN provides a score per action for them to be selected in a given input scenario. In deterministic contexts (which constitute the primary focus of this paper), the $argmax$ of the output layer is simply chosen as the action of the DRL algorithm.

The space of possible action values varies depending on the specific application. Here, we consider discrete numerical action spaces. That is, each action represents one of the several valid values for a parameter in the target system or network. For example, Pensieve (Mao et al., 2017) is a DRL algorithm used to determine the bit rate, or resolution, of the next video chunk to be streamed over the Internet. To accomplish this, the last layer of the DNN consists of neurons that correspond to the possible bit rates. During each decision step, the values of these neurons are evaluated to determine the scores of the corresponding actions. Ultimately, the DRL selects the action associated with the neuron having the highest score as its decision. This paper focuses on such DRL algorithms with numerical action space as they have not been systematically studied before despite their presence in various systems and networking scenarios

3. Encoding the Model and Properties

In this section, we present how the DRL model and properties of monotonicity and robustness are mathematically formulated as a Mixed-integer Linear Program (MILP). An MILP formulation allowed us to flexibly mix integer and real variables and explore the tractability of analyzing our properties of interest. That said, we plan to investigate formulations in other mathematical frameworks in the future to find the one best suited for analyzing monotoniciy and robustness properties.

3.1. Model Encoding

In this paper, we focus on encoding DNNs with fully connected layers, $ReLU$ activation functions, and a $Softmax$ output layer, following the literature (Eliyahu et al., 2021; Dethise et al., 2021). We discuss potential extensions to other architectures in Section 6.

In fully connected layers, the value of neuron j in layer i (x_i^j) is a weighted linear summation of the neurons in the previous layer after $ReLU$ is applied to them (y_i^j), as

explained in Equations 1 and 2. The coefficients a and b are learnable parameters that are determined during training.

$$x_i^j = \sum_{t=1}^{|X_{i-1}|} a_i^{t,j}\, y_{i-1}^t + b_i^j \tag{1}$$

$ReLU$ is a piece-wise linear function that captures non-linearities of the model as defined in Equation 2.

$$y_i^j = ReLU(x_i^j) = \begin{cases} x_i^j, & x_i^j \geq 0 \\ 0, & x_i^j < 0 \end{cases} \tag{2}$$

To encode a $ReLU$ function, we introduce 2 auxiliary variables, $z_i^j \in \{0,1\}$ and $s_i^j \in \mathbf{R}^+$ as defined in Equations 3a-3c.

$$y_i^j - s_i^j = x_i^j \tag{3a}$$
$$z_i^j = 1 \Rightarrow s_i^j = 0 \tag{3b}$$
$$z_i^j = 0 \Rightarrow y_i^j = 0 \tag{3c}$$

Finally, $Softmax$ function normalizes logits (output of the one-to-the-last layer) to produce the score (probability) of each action in the output layer as presented in Equation 4. Here, we assume $|X_{l-1}| = n$.

$$Softmax(X_{l-1})_j = \frac{e^{x_{l-1}^j}}{\sum_{k=1}^{n} e^{x_{l-1}^k}} \tag{4}$$

$Softmax$ is not a linear (and not even a piece-wise linear) function. However, as we discuss next (Section 3.2), our properties of interest are not dependent on the actual values in the output vector, but their relative order. $Softmax$ preserves that ordering (that is $\forall i,j \in \{1,2,\cdots,n\} : x_{l-1}^i \leq x_{l-1}^j \Rightarrow Softmax(X_{l-1})_i \leq Softmax(X_{l-1})_j$). As such, in deterministic scenarios where the action is simply the $argmax$ of output, we can safely ignore $Softmax$ and choose the $argmax$ of logits as the output of the model.

Our encoding of the the $argmax$ of the logits X_{l-1} is captured in equations 5a-5e. We assume the logits have bounded values: $\forall i \in \{1,\cdots,n\} : |x_{l-1}^i| \leq M$. The auxiliary variable m captures the maximum value among x_is and t is the $argmax$.

$$m \geq x_{l-1}^i, \qquad i = 1,\cdots,n \tag{5a}$$
$$m \leq x_{l-1}^i + 2M(1-q_i), \qquad i = 1,\cdots,n \tag{5b}$$
$$\sum_{i=1}^{n} q_i = 1 \tag{5c}$$
$$\sum_{i=1}^{n} i q_i = t \tag{5d}$$
$$m \in \mathbf{R}, \quad q \in \{0,1\}^n, \quad t \in \{1,\cdots,n\} \tag{5e}$$

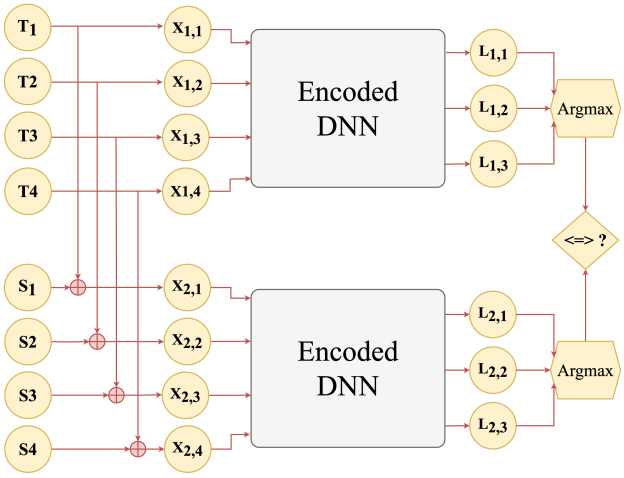

Figure 1. Property encoding scheme.

The only way to satisfy the first three constraint sets (5a-5c) is to have m as the maximum value and and $q_k = 1$ for $k = argmax_{i=1,\cdots,n}\, x_i$. Then t reads the $argmax$ in Equation 5d.

3.2. Property Encoding

Encoding monotonicity and robustness properties requires comparing the model's decision in two interrelated scenarios. Specifically, as illustrated in Figure 1, we encode two instances of the target model which are fed with two different sets of input. The base scenario is defined as $X_1 = T = \{T_i : i = 1,\cdots,N\}$ and the second scenario is $X_2 = T + S$ which is the base scenario shifted by slack variables $S = \{S_i : i = 1,\cdots,N\}$ that are constrained in accordance with the property of interest. That is,

$$X_{1,i} = T_i, \qquad i = 1,\cdots,N \tag{6a}$$
$$X_{2,i} = T_i + S_i, \qquad i = 1,\cdots,N \tag{6b}$$

Finally, the $argmax$ of the logits of each DNN, L_1 and L_2, are compared to verify the property of interest.

Now we formally define target properties. A DRL algorithm with a numerical action space is *robust* if the output of the corresponding DNN to the input X does not change more than d when a perturbation ϵ with maximum L_∞ norm of l is added to the input. This is mathematically defined as follows:

$$\forall X, |\epsilon|_\infty \leq l : \quad |DNN(X) - DNN(X+\epsilon)| \leq d \tag{7}$$

On the other hand, the output of a DRL algorithm is *monotonically* increasing with respect to its i-th input feature if the output of its DNN to an arbitrary input X is equal or

greater than that to the input of $X + d\overrightarrow{e_i}$, $d > 0$. The mathematical formulation goes as follows:

$$\forall X, d > 0: \quad DNN(X) \leq DNN(X + d\overrightarrow{e_i}) \qquad (8)$$

The decreasing case is formulated in a similar manner.

4. Case Studies

In this section, we discuss how the monotonicity and robustness properties apply to two DRL algorithms, CMARS, for resource allocation in a multi-tenant environment, and Pensieve, for bitrate selection in adaptive bitrate video streaming. We further analyze CMARS against those properties.

For the analysis, we considered using one of the existing specialized verification engines that have been developed for DNN verification, including Marabou (Katz et al., 2019) and CROWN (Wang et al., 2021). However, for our specific encoding, we were unable to use the current user interfaces of these tools for binary and integer variables, which we needed to compute and compare the $argmax$ of the DNN's output layer. As a result, we used Gurobi, an MILP solver, as our back-end.

4.1. Resource allocation in a multi-tenant environment

In next-generation mobile networks, network resources are allocated among different tenants[1] according to their specific performance requirements defined as service-level agreements (SLAs). To increase resource efficiency, different algorithms have been proposed with the goal of minimizing resource consumption while satisfying SLAs.

Inspired by (Alcaraz et al., 2022) and independent of this project, we have developed a DRL algorithm, called CMARS, which decides the number of radio resource blocks to allocate to a tenant in a mobile network. Previously realized performance statistics of a tenant, aggregated statistics of other tenants, network status, and amount of available resources are fed to the DRL model which will then decide the corresponding tenant's resource allocation as its action. That is, the set of possible actions includes integer variables from zero to the total amount of available radio resource blocks in the network.

The aggregate statistics from other tenants encompass three distinct features: the total number of users utilizing IoT service type, the average traffic volume of users employing constant bit rate service, and the average traffic volume of users utilizing variable bit rate service. Additionally, the network status measures the wireless link quality between users and the cell in terms of signal-to-noise ratio (SNR). To incorporate channel quality into the DRL algorithm, the

[1]A tenant is a group of network users under the same entity.

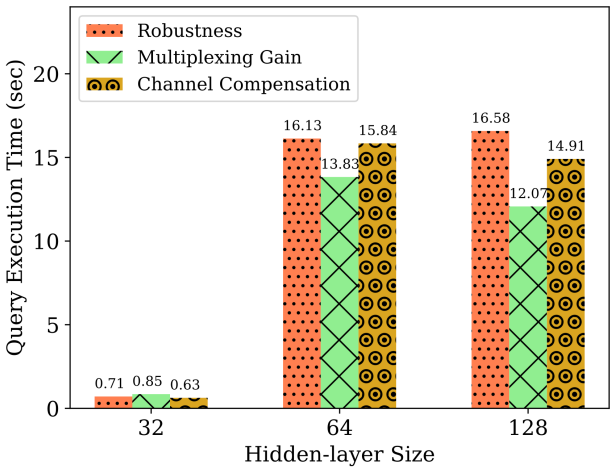

Figure 2. CMARS queries execution time.

average SNR values of the corresponding users for each tenant are calculated and provided as input to the DNN. Furthermore, all input features are normalized within the range of zero to one, according to the anticipated operational behavior of the system.

In the context of CMARS, there are two relevant monotonicity properties. First, we expect it to allocate an equal or greater amount of resources when the demand of other tenants decreases (*Multiplexing Gain*). Particularly, we use negative slack variables in the second encoded DNN's input features that correspond to the statistics of other tenants while other slack variables are kept zero. We then check for the $argmax$ of the output of the second DNN not to be less than that of the first one. Second, if the quality of the wireless channel between the tenant's users and the corresponding cell tower drops, more resources should be allocated for the same level of demand to maintain the required performance (*Channel Compensation*). Here, we use negative slack variables for the network status feature of the second DNN, and check that the $argmax$ of the output of the second DNN is not less than that of the first one. For robustness, we check that when the input features are slightly perturbed (using slack variables, as described in Equation 7), the allocation of resources should not undergo significant changes. All of these properties can be encoded using the proposed method in Section 3.

We encoded CMARS and the above properties as described in Section 3 for a DNN with 19 input features, a fully-connected layer with 32 neurons, and an action size of 30. In this case, *Channel Compensation* was the only property that held true. For the *Multiplexing Gain* property, the counter-example alluded to a corner case but important scenario that was not part of the training set. The counter-example for

the *Robustness* property was a surprising scenario in which a perturbation of $L_\infty = 0.0001$ led to a difference of 16 resource units (out of the total of 30 resource blocks) in the action. We envision such counter-examples to be leveraged to improve the algorithms further to enforce the violated properties (see Section 6 for more details).

To assess the scalability of the formulation and Gurobi, we conducted experiments by altering the model's size in terms of action size and the number and size of hidden layers. Specifically, we examined a scenario with a single hidden layer and an action size of 15. The average execution time (averaged over 10 runs) for various verification queries is illustrated in Figure 2. The findings indicate that the proposed approach is viable for small DNNs. When the number of hidden layers is increased to two, and 32 neurons are utilized in each hidden layer, the execution time amounts to 0.87 seconds. However, for a larger number of neurons in hidden layers, the execution time gets prohibitively large (e.g., hours). We plan to investigate whether other formulations or using other backends could lead to improvements (see Section 6 for more details).

4.2. Bitrate selection in adaptive bitrate video streaming

To optimize user quality of experience in video streaming, video clients employ adaptive bitrate algorithms (ABR) that dynamically selects the bitrate (resolution) at which the next video chunk (say, a 4-second video segment) should be downloaded from the server. The goal is for the bitrate to be high but with low rebuffering and low bitrate oscillations.

Pensieve (Mao et al., 2017) is a DRL-based ABR algorithm that takes the following as input: the network throughput measurements for the past k video chunks, the download time of the past k video chunks, the current buffer level, and the number of chunks remaining in the video. In the output, Pensieve's DNN will give the chosen bitrate for the next video chunk to be streamed, which can be any of 300, 750, 1200, 1850, 2850, 4300 kbps, pertaining to video modes in 240, 360, 480, 720, 1080, 1440p.

There are two relevant monotonicity properties for Pensieve. First, when the measured network throughput increases, the algorithm should not decrease the bitrate (*Capacity Utilization*). Second, in order to prevent interruptions in video playback, when there are fewer video chunks stored in the buffer, the selected bitrate should either remain the same or be lower compared to situations where there are more video chunks in the buffer (*Rebuffering Avoidance*).

This algorithm, at the surface, is quite different from CMARS, but similar monotonicity and robustness properties can be encoded following the same approach in Section 3. This provides an encouraging indication of the usefulness of formalizing these properties and the approach in Section 3 among the DRL algorithms with numerical action space in the system and networking domain.

5. Related Work

The authors of (Eliyahu et al., 2021) present a formalization of natural safety and liveness properties for deterministic DRL algorithms used in systems and networking, expressed as a set of constraints on the input and output of the embedded DNN. They use model checking to identify a scenario where the property does not hold. Similarly, (Dethise et al., 2021) investigates the property of adversarial robustness, calculating the minimum amount of required input perturbation size to alter the output.

However, these proposed methods only analyze what the output of the DRL algorithms should be for a certain input in a constrained subset of the input space. For example, the property formulated in (Eliyahu et al., 2021) specifies that the RL model should not select the worst resolution when the network condition is excellent. That is, in contrast to this work, they do not study the relationship between changes in the input and the changes in the output over the entire input space, and cannot be used to analyze the monotonicity and robustness properties defined in this work.

6. Discussion and Future Work

Scalability. In our proof of concept, we used Gurobi which is a general-purpose solver because binary and integer variables are not supported in specialized DNN verification engines. However, we observed a prohibitively large verification time as we increased the DNN size. As such, our natural and immediate next step is to investigate these scalability issues by exploring other formulations and backends (e.g., extending specialized DNN verification engines to support binary and integer variables). Specifically, we plan to explore specialized techniques for DNN verification, such as using optimized decision procedures, parallelization, and abstraction-refinement techniques to encode robustness and monotonicity properties (Singh et al., 2018; Tjeng et al., 2017; Zhang et al., 2022).

Other DNN architectures. In this paper, we focused on encoding DNNs with fully connected layers, $ReLU$ activation functions, and a $Softmax$ output layer, following the literature (Eliyahu et al., 2021; Dethise et al., 2021). However, our general approach to property encoding, i.e., capturing monotonicity and robustness properties through comparing the output of two interrelated scenarios, is not tied to this specific architecture. We plan to investigate other architectures that come up in systems and networking DRL algorithms, encode them in suitable verification engines (e.g., (Ostrovsky et al., 2022)), and analyze them against our proposed properties.

Property Enforcement. Verification can help us find scenarios in which a DRL algorithm does not behave as desired and/or expected. To further enforce the algorithm to meet the desired properties of monotonicity and robustness, we envision three different approaches.

First, one can inject the counter-examples into the training set to reinforce the desired behavior around those points in the model (Tan et al., 2021). Second, the ratio of monotonicity and robustness properties violations in a few samples around the input state of an iteration can be calculated (Chakravarthy et al., 2022) and fed to the DRL model as a cost value, following a Constrained DRL approach. The benefit of this approach is that it distinguishes the desired properties from the original goals of the DRL, making it possible to explicitly explore any potential trade-off between the two. Finally, one could try to proactively set an increasing/decreasing relationship between an input feature and the output of the DNN. We plan to look into whether it is possible to constrain the DNN to be trained to be within a class of functions that are increasing/decreasing with respect to a subset of their input features.

Probabilistic comparative verification. The *Softmax* does not lend itself well to common encodings given that it is not linear or piece-wise linear. In deterministic contexts where the final action is determined by the $argmax$ of the DNN, we can forgo *Softmax* and compare the final actions (Section 3). However, without encoding *Softmax*, we are unable to compare the final probabilities output by the DNN, which could be useful in scenarios where the choice of action is probabilistic. Overcoming this challenge will pave the way for verifying probabilistic DRL algorithms and probabilistic comparative verification.

7. Conclusion

In conclusion, this study focused on the application of formal verification techniques to ensure the safe deployment of DRL algorithms, specifically targeting those with discrete numerical action spaces commonly found in systems and networking domains. By concentrating on the fundamental properties of monotonicity and robustness, the investigation employed formal methods to verify these properties beyond the limitations of the training set.

References

Alcaraz, J. J., Losilla, F., Zanella, A., and Zorzi, M. Model-based reinforcement learning with kernels for resource allocation in ran slices. *IEEE Transactions on Wireless Communications*, 22(1):486–501, 2022.

Amir, G., Schapira, M., and Katz, G. Towards scalable verification of deep reinforcement learning. In *2021 formal methods in computer aided design (FMCAD)*, pp. 193–203. IEEE, 2021.

Bojja Venkatakrishnan, S., Gupta, S., Mao, H., Alizadeh, M., et al. Learning generalizable device placement algorithms for distributed machine learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Chakravarthy, A., Narodytska, N., Rathis, A., Vilcu, M., Sharif, M., and Singh, G. Property-driven evaluation of rl-controllers in self-driving datacenters. In *Workshop on Challenges in Deploying and Monitoring Machine Learning Systems, NeurIPS Virtual Workshop*, 2022.

Dethise, A., Canini, M., and Narodytska, N. Analyzing learning-based networked systems with formal verification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1–10. IEEE, 2021.

Eliyahu, T., Kazak, Y., Katz, G., and Schapira, M. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pp. 305–318, 2021.

Hamadanian, P., Schwarzkopf, M., Sen, S., and Alizadeh, M. How reinforcement learning systems fail and what to do about it. In *The 2nd Workshop on Machine Learning and Systems (EuroMLSys)*, 2022.

He, H., Wei, T., Zhang, H., Liu, C., and Tan, C. Characterizing neural network verification for systems with nn4sysbench. In *1st Workshop on Formal Verification of Machine Learning*, 2022.

Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, pp. 443–452. Springer, 2019.

Lan, D., Tan, X., Lv, J., Jin, Y., and Yang, J. A deep reinforcement learning based congestion control mechanism for ndn. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–7. IEEE, 2019.

Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pp. 197–210, 2017.

Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pp. 270–288. 2019.

Ostrovsky, M., Barrett, C., and Katz, G. An abstraction-refinement approach to verifying convolutional neural networks. In *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings*, pp. 391–396. Springer, 2022.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. Boosting robustness certification of neural networks. In *International conference on learning representations*, 2018.

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

Tan, C., Zhu, Y., and Guo, C. Building verified neural networks with specifications for systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '21, pp. 42–47, 2021.

Tjeng, V., Xiao, K., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Wang, M., Huang, S., Cui, Y., Wang, W., and Liu, Z. Learning buffer management policies for shared memory switches. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 730–739, 2022. doi: 10.1109/INFOCOM48880.2022.9796784.

Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.-J., and Kolter, J. Z. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

Yan, F. Y., Ayers, H., Zhu, C., Fouladi, S., Hong, J., Zhang, K., Levis, P. A., and Winstein, K. Learning in situ: a randomized experiment in video streaming. In *NSDI*, volume 20, pp. 495–511, 2020.

Zhang, H., Wang, S., Xu, K., Wang, Y., Jana, S., Hsieh, C.-J., and Kolter, Z. A branch and bound framework for stronger adversarial attacks of relu networks. In *International Conference on Machine Learning*, pp. 26591–26604. PMLR, 2022.

Zhu, H., Gupta, V., Ahuja, S. S., Tian, Y., Zhang, Y., and Jin, X. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pp. 258–271, 2021.