Analyzing Symbolic Properties for DRL Agents in Systems and Networking

ANONYMOUS AUTHORS

Deep reinforcement learning (DRL) has shown remarkable performance for complex control problems in systems and networking, including adaptive video streaming, wireless resource management, and congestion control. For safe deployment, however, it is critical to reason about how such agents behave across the range of system states they may encounter in practice. Existing verification-based approaches in this domain primarily focus on *point properties* – properties defined around fixed input states – which offer limited coverage and require substantial manual effort to identify relevant input-output pairs for analysis.

In this paper, we study *symbolic properties* – i.e., those that specify expected behaviors over ranges of input states – for DRL agents in systems and networking. We present a generic formulation for symbolic properties, such as monotonicity and robustness, and show how they can be analyzed using existing DNN verification engines. Our approach encodes symbolic properties as comparisons between related executions of the same policy and decomposes them into practically tractable sub-properties. These techniques serve as practical enablers for applying existing verification tools to symbolic analysis. Using our framework, diffRL, we conduct an extensive empirical study across three representative DRL-based control systems – adaptive video streaming, wireless resource allocation, and congestion control – covering both discrete and continuous action spaces. Through these case studies, we analyze symbolic properties over broad input ranges, examine how property satisfaction evolves during training, study the impact of model size on verifiability, and compare multiple verification backends. Our results show that symbolic properties provide substantially broader coverage than point properties and can uncover non-obvious, operationally meaningful counterexamples, while also revealing practical solver trade-offs and limitations.

ACM Reference Format:
Anonymous authors. 2026. Analyzing Symbolic Properties for DRL Agents in Systems and Networking. 1, 1 (January 2026), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Deep Reinforcement Learning (DRL) has emerged as a promising approach for control problems in systems and networking, where decisions must be made in complex and highly dynamic environments. In these settings, DRL agents are typically embedded in control loops: the agent observes the system state — represented as a feature vector of performance indicators such as throughput, latency, packet loss, or resource utilization – and selects actions according to its learned policy. These actions correspond to concrete numerical control decisions for resource allocation [1–4], traffic engineering and path selection [5–7], job scheduling [8–11], and tuning streaming video bitrate [12–14] or congestion window sizes [15–18].

To safely and effectively integrate DRL agents into such control loops, it is essential to reason about how "well-behaved" these agents are across the range of inputs they can encounter after deployment. This is particularly important as DRL agents make decisions based on Deep Neural Networks (DNNs) that function as black boxes with opaque input-output relationships from the

Author's Contact Information: Anonymous authors.

ACM XXXX-XXXX/2026/1-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

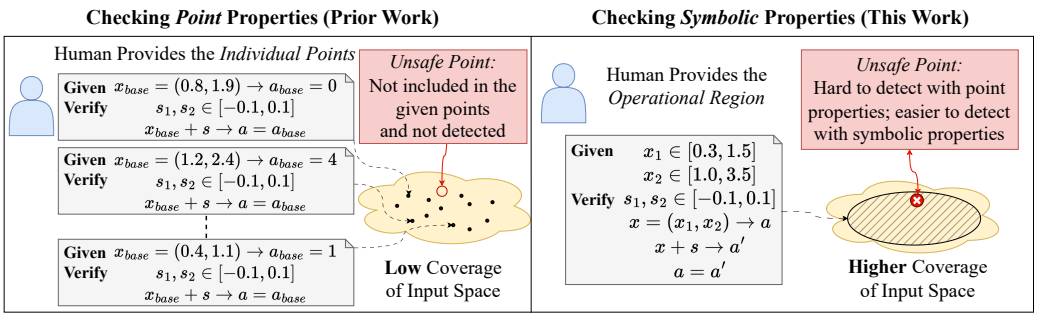

Fig. 1. **Point-wise versus symbolic property analysis.** Left (Prior Work): Analysis is performed at individual points that humans provide, checking that the property holds under bounded perturbations around one concrete point in the input space. In this example, the property asserts that the output action does not change. Checking properties only at individual points results in limited coverage of the input space and may miss unsafe behavior outside the considered points. Right (This Work): Analysis is performed over an entire operational region using symbolic properties. This provides higher coverage of the input space and enables the detection of unsafe points that are difficult to uncover with point properties.

perspective of human operators. As such, we would like to reason about whether a DRL agent's selected action a changes in undesirable ways when the observed state x is perturbed to $x + s$, where s is a small, bounded slack. Such properties are important as the input to these agents typically comes from real-world measurements that are susceptible to noise due to variability in measurement timing and limitations in measurement precision [19]. Moreover, undesirable output deviations in response to small input changes can expose systems to adversarial manipulation, e.g., by inducing disproportionate resource allocations through minor input perturbations [20].

Previous work [19, 21, 22] commonly relies on state-of-the-art DNN verification engines [23] or Mixed-Integer-Linear-Program (MIP) solvers [24] to reason about DRL agents in the systems and networking domain. However, these works can only check *point properties* where the system state x and the reference action a are *fixed to concrete constants*. For example, consider Pensieve [12], a DRL agent for adaptive video streaming that takes in the client's video buffer size and the measured throughput for the past streamed video segment as two of the input features determining the system state and decides between six possible bitrates from 300 Kbps to 4.3Mbps for the next video segment as its action. WhiRL [21], which uses the Marabou DNN verification engine [23], and similarly [19, 22], can only analyze Pensieve against point properties such as the following: If the video buffer has zero or one segment and the throughput for the past video segments is as low as a user-specified value (i.e., one concrete point among all possible system states), the DRL agent's output action should not be 4.3Mbps (i.e., a concrete action).

While checking each individual point property is typically feasible and efficient, relying on point properties to reason about DRL agents operating in complex, dynamic systems and networks has two main drawbacks (Figure 1). First, point properties have low coverage. Each one only provides assurance around a single concrete point in the agent's operational input space. Second, one must explicitly identify which input points are worth checking. This limitation is often manageable in supervised learning, where a (large) labeled data set explicitly captures the expected output value for a wide range of input points (e.g., AutoSpec [25]). In DRL, however, there is *no predefined ground truth for the expected behavior at a given state*, i.e. the best action is unknown prior to training and must be learned through interactions with the environment [26]. As such, making point properties effective in this setting requires substantial domain expertise to identify meaningful points, which is difficult to scale to a level that provides sufficient coverage.

In this paper, we focus on *symbolic properties* instead, where the system state x and the reference action a are left symbolic. This shifts analysis from isolated input points to entire regions of the agent's operational space, substantially increasing coverage and removing the need for domain experts to manually enumerate a large number of representative input–output pairs The key enabling insight behind our approach is that many properties of interest for DRL agents can be expressed as a symbolic comparison between two executions of the same policy evaluated on interrelated symbolic states – a "base" input and a perturbed one.

While conceptually simple, this perspective enables us to analyze symbolic properties using existing verification tools and, crucially, makes it practical to explore how far such analysis can be pushed for DRL agents in systems and networking. Concretely, it provides two main advantages. First, it allows symbolic properties to be systematically decomposed into a finite set of sub-properties that compare concrete output neurons of the two agent copies. Each sub-property remains symbolic over the input space but is more constrained than the original formulation. As a result, existing verification techniques – previously applied only to point properties – can be directly leveraged for their analysis. As our case studies demonstrate (§6-§8), this decomposition is often sufficient to make symbolic analysis tractable in practice for representative DRL agents in this domain.

Second, the encoding is deliberately chosen to be generic and solver-agnostic so that it can apply to a variety of DRL agents in systems and networking and is compatible with multiple existing verification and analysis techniques, such as MIP [27], Satisfiability Modulo Theories (SMT) [28, 29], Bound Propagation [30, 31], and Branch-and-Bound (BaB) on input domain [32–35]. This flexibility allows us to analyze the same set of sub-properties across multiple verification engines and empirically assess their complementary strengths, enabling broader coverage than would be possible with any single DNN verification technique alone.

Building on these insights, we conduct an empirical study of symbolic properties for DRL agents in systems and networking. Specifically, we create a framework called *diffRL* that takes in a DRL agent, the operational range of its input variables, and other property details. It then uses the encoding and decomposition strategies proposed in this work to generate queries for backend verification engines. We apply our approach across three representative control domains – adaptive video streaming (Pensieve [12]), wireless resource allocation (CMARS [1]), and congestion control (Aurora [36]) – covering both discrete and continuous action spaces.

Across these case studies, we analyze symbolic monotonicity and robustness properties over broad operational input ranges, examine how property satisfaction evolves during training, demonstrate the significance of the discovered counterexamples, study the impact of model size on verifiability, and compare the behavior of multiple verification backends, including MIP-, SMT-, and BaB–based engines [29, 31, 35, 37, 38]. Collectively, these results provide the first systematic view of how far symbolic property analysis can be pushed for integrating DRL agents in systems and networking, and which insights such analysis can – and cannot – reliably deliver in practice.

Summary of contributions. This paper makes the following contributions:

- We introduce a generic formulation of symbolic properties for DRL agents in systems and networking to capture expected high-level behaviors of DRL-based control policies over ranges of system states (§3).
- We enable analyzing symbolic properties via comparative encoding and decomposition into a finite set of sub-properties analyzable by existing DNN verification engines (§4).
- We conduct a systematic study of symbolic properties, using our framework diffRL, for DRL agents in adaptive video streaming, wireless resource allocation, and congestion control, examining verifiability, counterexamples, training dynamics, and the impact of model capacity and solver choice (§5-§8).

2 DRL Agents in Systems and Networking

A DRL agent uses a DNN to represent its policy or value function, with parameters that specify how system states are mapped to actions or expected rewards. During training, the agent observes the system's state, takes an action, receives feedback in the form of a reward, and adjusts the DNN's parameters accordingly. In the context of systems and networking, the system state is typically represented by a feature vector capturing performance indicators such as throughput, latency, packet loss, or resource utilization. The agent's actions correspond to control decisions such as allocating resources, scheduling jobs, tuning streaming video bit rate or congestion window sizes.

Deterministic policies. We focus on analyzing the DNN agent in its *post-training form* as the trained DNN is what defines the agent's decision-making logic. This means our approach is applicable to a wide range of DRL agents *regardless of their specific training algorithm*. Moreover, we observe that DRL agents predominantly operate *deterministically* in practical deployment scenarios in systems and networking. Some agents are inherently deterministic at runtime, such as those trained using Q-learning [15] or Deterministic Policy Gradient (DPG) [3, 4, 7]. Others are derived from stochastic policy-gradient agents [2, 6, 8–12, 17, 18, 36, 39–42]. That is, the DNN outputs a probability distribution over actions, representing the policy. During training, actions are sampled from this distribution to enable exploration, while during deployment it is common practice to deterministically select the action with the highest probability [21, 43]. This deterministic behavior ensures reproducibility and predictability of the agent's decision-making behavior in deployment.

Common architectures and numerical action spaces. DNN architectures for DRL agents in systems and networking often consist of fully connected or convolutional layers with Rectified Linear Units ($ReLU(x) = \max(0, x)$) as activation functions. The action space is typically *numerical* [1–4, 12–15, 17, 36, 41, 42, 44, 45], representing control decisions such as the amount of resources to allocate to each component, the congestion window size, or video streaming bitrate. This numerical action space can be discrete [1, 3, 12–15, 41, 42, 44, 45] or continuous [2, 4, 17, 36]. For agents operating in discrete action spaces (e.g., pick one of five video bitrates), the DNN typically includes one output neuron per possible action, each representing either the predicted Q-value (value-based methods) or the probability of selecting that action (policy-based methods). For continuous numerical action spaces (e.g., picking a memory size from a given range for resource allocation), the DNN output typically represents the parameters of a continuous probability distribution (e.g., the mean and variance of a Gaussian). Deterministic agents with continuous actions may also use a single output neuron per action dimension, directly representing the output action.

3 Defining Symbolic Properties

To safely and effectively integrate DRL agents into control loops in systems and networking, it is essential to reason about how an agent's selected action changes when the observed system state is perturbed. In practice, such perturbations arise from noisy measurements or transient fluctuations in system conditions. A symbolic property captures whether these input perturbations cause the agent's output to change in undesirable ways.

Formally, let $\pi : X \to A$ denote the deterministic policy function that determines the DRL agent's action, where $X \subseteq \mathbb{R}^n$ is the space of system states and $A \subseteq \mathbb{R}$ is the action space. In our setting, π is represented by a DNN. We define a symbolic property by specifying *constraints over pairs of executions* of π. Specifically, for any input state $x \in X$, we compare the agent's output at x and at a perturbed state $x + s$, where a comparison function f and a threshold d capture the notion of acceptable change in the action. Moreover, the allowable perturbations $s \in \mathbb{R}^n$ are restricted by per-dimension lower and upper bounds on elements of s:

$$\forall x \in X, l_{s_i} \leq s_i \leq u_{s_i} : \quad f(\pi(x), \pi(x + s)) \leq d \tag{1}$$

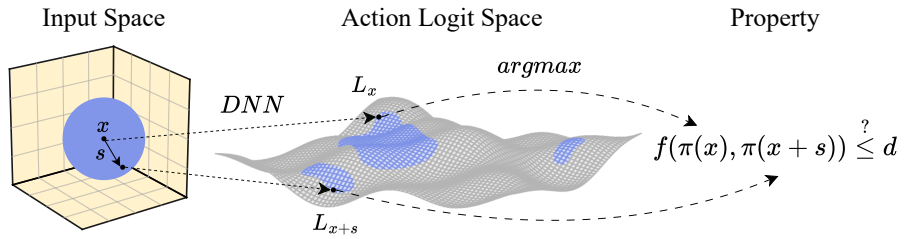

Fig. 2. **Symbolic properties over DRL agent execution pairs (§3).** A symbolic input x from the region of interest in the input space and its bounded perturbation $x + s$ induce two executions of the agent's policy π. The policy π is based on a non-linear DNN. As such, input perturbations may result in jumps in the DNN's multi-dimensional continuous output space (see blue highlights in the center). The DNN's output may represent predicted action values or action probabilities, which we refer to as action logits in general. For deterministic agents with discrete actions (§2), the policy π is defined as the *argmax* of these logits. Our symbolic properties constrain the two resulting actions $\pi(x)$ and $\pi(x + s)$ with a function f and tolerance d.

In other words, the property is satisfied iff for all $x \in X$ and all perturbations s within the specified bounds, the output comparison $f(\pi(x), \pi(x + s))$ is bounded by d. Verifying properties of this form is not trivial, as the policy is based on a non-linear DNN and thus small changes in its input may lead to jumps in its output, see Figure 2.

The comparison function f and perturbation bounds are left abstract to allow the same symbolic formulation to capture a wide range of properties. This allows properties to vary both in what aspects of the output are compared and how input perturbations are structured, without changing the underlying analysis pipeline. Based on common patterns in the choice of f and the perturbation bounds, we instantiate this general formulation with two broad classes of symbolic properties that arise naturally in systems and networking: *robustness* and *monotonicity*.

Category 1: Symbolic robustness properties. We want to ensure that the DRL agent is *robust*, meaning for any input in the agent's given operational range, small perturbations to the input should not cause disproportionately large changes in the output. For example, if a DRL agent uses network packet loss rate as an input feature to decide the congestion window size, minor fluctuations in the measured loss rate (e.g., due to noisy measurements or small transient fluctuations in network conditions) should not result in substantial changes to the congestion window size.

Formally, our robustness property checks whether for any input $x \in X$, the agent's output changes by at most $d > 0$ when the perturbation has an L_∞-norm bounded by a small $\epsilon > 0$. This definition is consistent with prior work on observation robustness [46], which aims to find policies that are insensitive to small perturbations of their input observations. Within our property specification framework, this translates to setting all perturbation lower bounds to $-\epsilon$ and all upper bounds to ϵ, and defining f as the absolute value of the difference between its input actions:

$$\forall i, l_{s_i} = -\epsilon, u_{s_i} = \epsilon \qquad f(y, z) = |y - z| \tag{2}$$

Category 2: Symbolic monotonicity properties. Another important class of properties concerns *monotonicity*, which captures expected directional relationships between input features (the system state) and the agent's output. Informally, monotonicity requires that, over the agent's given operational range, increasing a specific input feature should cause the output action to predictably increase or decrease in the expected direction. For instance, if the network loss rate increases, we expect a DRL-based congestion control agent to decrease the congestion window size.

Formally, we say that the policy π is monotonically increasing with respect to its i^{th} input feature if, for any input $x \in X$, increasing the i^{th} component of x by up to $\epsilon_i > 0$ results in an equal or greater output action with a tolerance of d. The tolerance parameter d accounts for small valid

output fluctuations and prevents the property from being overly restrictive. Within our property specification framework, this translates to (i) setting the perturbation lower bounds and all but the i^{th} upper bounds to zero, (ii) setting u_{s_i} to ϵ_i, and (iii) defining f as the directional change:

$$\forall j \neq i, l_{s_j} = u_{s_j} = 0 \qquad l_{s_i} = 0, u_{s_i} = \epsilon_i \qquad f(y, z) = z - y \tag{3}$$

The monotonically decreasing property is formulated analogously, by flipping the direction of either f or the bounds. We provide several concrete properties in the case study sections §6-8. Notably, our experience in defining these properties shows that doing so does not require deep knowledge of the underlying DRL model or its training process. Instead, the properties encode expected relationships between system metrics and control actions that are already well understood by domain experts in systems and networking (e.g., higher loss should not increase sending rate). In practice, defining a monotonicity property amounts to selecting the relevant input feature, the expected direction of change, and appropriate tolerance parameters (ϵ_i, d), while the logical structure of the property remains fixed and can be analyzed automatically in a symbolic manner.

Beyond robustness and monotonicity. While our work focuses on robustness and monotonicity as representative and widely applicable property classes, our formulation of symbolic properties is not limited to these instances and can express a variety of other properties through alternative choices of analytical f and perturbation bounds. For example, one can explore richer directional properties – similar to monotonicity but across multiple input features – by specifying directional perturbations for multiple input features and adjusting f accordingly to capture the desired direction of the output change. Similarly, f does not necessarily need to be related to the difference between $\pi(x)$ and $\pi(x + s)$. For example, if needed, it can be any linear combination of the two. Crucially, all such properties can be expressed as an instance of Eq. 1 and handled by the same encoding and decomposition strategy described in §4.

3.1 Other Property Definition Considerations

Accounting for multi-step dependencies. In some DRL agents like Pensieve [12] and Aurora [36], the system state includes metrics collected over the past k steps. Let these inputs be represented by $x_1, \cdots, x_k$. In our analysis, we consider the full operational range of all x_i. The sequence of x_i influences the chosen action, which in turn affects the subsequent state, and specifically, the next observed metric in the history (x_1). As such, some combinations of x_is are more likely to occur in practice, while certain ones may be uncommon. When it comes to safety properties, capturing more combinations (i.e., over-approximation) does not compromise safety guarantees. If the analysis deems an agent safe, the agent will be safe even if the more infrequent combinations do not happen. If the analysis returns a counterexample that is believed to be uncommon, it can be ruled out by further refining of the input and slack bounds.

Moreover, the symbolic properties defined above represent a single-step execution of the DRL agent. While DRL agents usually interact with the environment over multiple steps, analyzing a single-step transition is typically sufficient for the properties considered in this work. This is because our symbolic properties are defined and analyzed over the agent's operational input space as identified by the user. As such, it can capture the agent's behavior transitioning from an arbitrary state. That is, the agent's response from any state in that range is accounted for.

Properties as safety checks, not effectiveness guarantees. While these two classes of properties are essential, satisfying them alone does not guarantee that a DRL agent is effective in its decision making. For instance, a DNN that outputs a constant action regardless of its inputs may trivially satisfy these properties, yet such a DRL agent is inherently ineffective and fails to achieve meaningful behavior. However, when these properties are applied to a DRL that performs well in terms of the achieved rewards in the training scenarios, they can serve as critical safety checks.

Specifically, they can help assess the generalizability and trustworthiness of a DRL agent beyond the specific scenarios encountered during training. For example, property violations often indicate deficiencies in the training procedure (such as overfitting) that can be addressed to improve the DRL agent. Consequently, while these properties do not guarantee effective decision making on their own, they are indispensable for developing DRL agents that are both effective and dependable. We examine how property satisfaction evolves during training in our case studies (§6)

4 Enabling Symbolic Property Analysis with diffRL

Once the user expresses a symbolic property in the form of Eq. 1, our goal is to determine whether it holds over the specified operational range of the system state variables x. Similar to prior work [19, 21, 22, 25], we seek to leverage existing DNN verification engines to perform this analysis. However, unlike prior approaches that verify properties for a fixed, concrete input x, our goal is to keep x symbolic so that the result applies to all states within the specified range.

A practical challenge in doing so lies in representing the agent's selected action $\pi(x)$ in a way that is compatible with existing verification tools. As discussed in §2, many DRL agents in systems and networking [1, 3, 12, 14, 15, 41, 42, 44] operate over a finite, discrete numerical action space, where each output neuron corresponds to a valid control decision (e.g., the bitrate choice in Pensieve). In practical deployment, these agents act deterministically – either inherently or after collapsing stochastic policies – by selecting the action corresponding to the maximum-valued output neuron, i.e., $\pi(x)$ is usually the *argmax* over the output layer.

Existing DNN verification engines – whether MIP-based [27], SMT-based [28, 29], or BaB-based [32–35] – are most naturally applicable to properties when the policy's selected action is known a priori, i.e., when the property can be expressed by constraining a specific output of the network relative to the others. Intuitively, when the selected action is fixed, the property boils down to a relatively small set of linear inequalities between known outputs, which significantly constrains the search space. If the selected action is not fixed, the *argmax* introduces a combinatorial, non-convex case distinction over all possible "winning" output logits, greatly increasing the number of regions the verifier must consider. As a result, these verifiers do not natively support non-linear, discrete selection operators such as *argmax* in a form amenable to symbolic reasoning. This does not pose a problem for prior work on verifying DRL agents in systems and networking, which instantiates x to a concrete value and therefore fixes which output neuron is selected by the policy.

In contrast, our symbolic properties quantify over the entire specified range of inputs. As a result, the selected output neuron, and consequently $\pi(x)$, depends on the input and cannot be fixed a priori. We address this mismatch by combining two ideas: (1) encoding symbolic properties by symbolically comparing two inter-related copies of the DRL agent, and (2) a simple decomposition strategy that reduces the symbolic property into a collection of sub-properties, each of which conditions on a fixed output action and can be analyzed using existing verification techniques.

This combination enables a practical advantage that is central to our case studies. As we show in §6-8, existing verification engines exhibit complementary strengths across different sub-properties. By keeping our encoding and decomposition strategy generic and solver-agnostic, we do not need to rely on a single engine to handle all cases – different engines can resolve different sub-properties. As we show in our case studies, this flexibility improves the practical tractability of analyzing symbolic properties and enables us to make progress beyond point properties for realistic DRL agents used in systems and networking.

Comparative encoding. Given a property in the form of Eq. 1, we encode it by *symbolically* comparing the output actions of two copies of the target DRL agent whose inputs are related through property constraints. This is illustrated in Figure 3. Suppose the system state x is a vector of size n. The input to the first copy, $x_1^1, \cdots, x_n^1$ represents the *base* scenario, where $x_i^1 \in [l_{x_i}, u_{x_i}]$ is

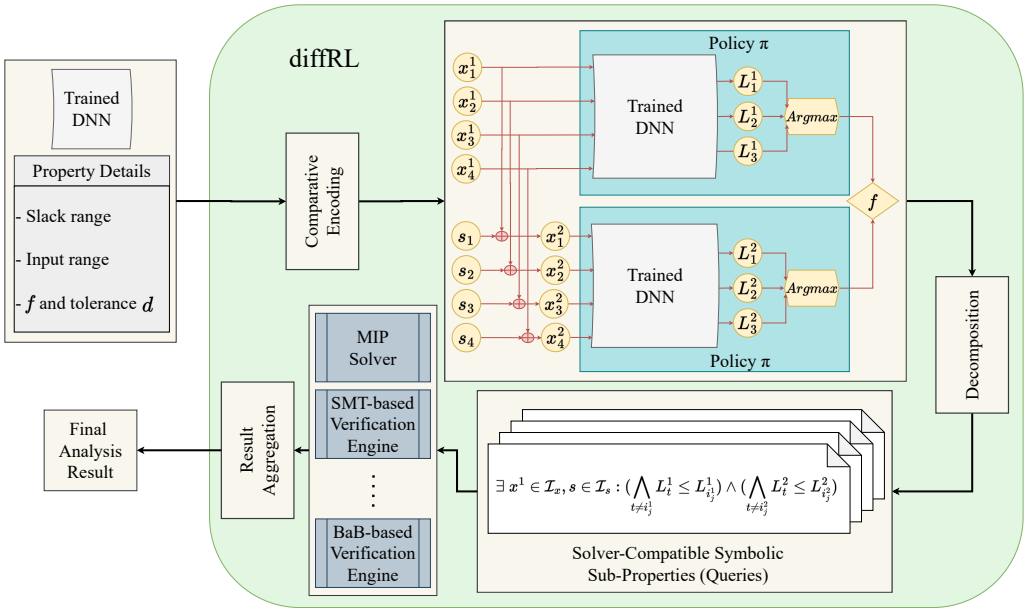

Fig. 3. **diffRL**'s overview (§4). Starting from a trained DNN and a symbolic property specification, **diffRL** applies a comparative encoding to construct two coupled executions of the same policy under a bounded input perturbation. The resulting formulation is decomposed into multiple queries, dispatched to heterogeneous solvers. The individual solver outcomes are then aggregated to produce the final verification result.

allowed to take any value within the operational range of the corresponding state variable, specified as lower and upper bounds l_{x_i} and u_{x_i}. The input to the second copy, $x^2 = x^1 + s$, is the base scenario shifted by the slack vector s, whose elements are constrained under the property of interest. For example, the robustness property in Eq. 2 constrains each element s_i to be between $-\epsilon$ and ϵ.

Property decomposition. Let L^1 and L^2 denote vectors of output neurons of the DNNs in the first and second copies, respectively (Figure 3). As discussed above, the agent's action $\pi(x^1)$ and $\pi(x^2)$ correspond to *argmax* of L^1 and L^2, respectively. Symbolic properties expressed using Eq. 1 constrain the relationship between these two selected actions via the function f and threshold d, i.e., they check whether $f(\pi(x^1), \pi(x^2)) \leq d$ holds. To enable analysis using DNN verification engines, we decompose this symbolic comparison into a finite set of sub-properties, each corresponding to a concrete pair of output neurons.

Specifically, for each pair of output neurons (L^1_i, L^2_j), diffRL automatically determines whether having L^1_i and L^2_j as the *argmax* of the output neurons is a valid or invalid outcome. An invalid outcome means that, given the assumed relationship between the two sets of inputs, selecting L^1_i and L^2_j as the *argmax* of their respective DNN copies violates the property. Similarly, a pair is an valid outcome if this combination of selected actions is permitted by the property. As a concrete example, consider a symbolic robustness property for Pensieve (see §6) that requires the output bitrate to change by no more than two resolution levels under small input perturbations. If the output neuron corresponding to resolution $1440p$ is selected as the *argmax* in the base copy, then output neurons corresponding to $480p$, $360p$, or $240p$ becoming the *argmax* in the second copy would violate the property, and thus form invalid pairs with $1440p$.

The symbolic property is violated *if and only if at least one invalid pair is feasible* under the input and slack constraints. As such, for each invalid pair, diffRL generates a query checking its feasibility and invokes existing DNN verification engines to analyze it. If any invalid pair is feasible, diffRL

obtains a concrete counter-example for the symbolic property. If none are feasible, the symbolic property is guaranteed to hold over the given operational input range. Formally, and as shown in Figure 3, given a DRL agent and a property, diffRL automatically generates the set of invalid output neuron pairs $\mathcal{P} = \{(i_1^1, i_1^2), \cdots, (i_m^1, i_m^2)\}$ for the two DNN copies. For each invalid pair $(i_j^1, i_j^2) \in \mathcal{P}$, diffRL creates a query Q_j that checks if any input to the DNN copies – within the specified bounds – leads to i_j^1 and i_j^2 being the *argmax* in output layers L^1 and L^2, respectively:

$$Q_j : \exists\, x^1 \in \mathcal{I}_x, s \in \mathcal{I}_s : \Big(\bigwedge_{t \neq i_j^1} L_t^1 \leq L_{i_j^1}^1\Big) \wedge \Big(\bigwedge_{t \neq i_j^2} L_t^2 \leq L_{i_j^2}^2\Big) \tag{4}$$

Here, $\mathcal{I}_x = [l_{x_1^1}, u_{x_1^1}] \times \cdots \times [l_{x_n^1}, u_{x_n^1}]$ and $\mathcal{I}_s = [l_{s_1}, u_{s_1}] \times \cdots \times [l_{s_n}, u_{s_n}]$ denote the bounds on the input state and slack variables. Each query is directly supported by existing DNN verification engines. If the verification engine finds a concrete set of input variables that satisfy the query constraints, diffRL reports a violation of the symbolic property. If none of the queries for the invalid pairs are satisfiable, the property holds and is reported as such by diffRL.

Domain factors that keep symbolic analysis manageable. DRL agents used in systems and networking use DNNs that are substantially more compact than those used in other application domains [21]. Unlike domains such as computer vision, which require deep architectures to extract meaningful features from raw pixel data, systems and networking agents operate on structured, high-level inputs such as network latency, throughput, and buffer occupancy. As a result, their DNN architectures are shallower and contain fewer neurons. This makes individual queries for these agents – despite involving symbolic input ranges rather than single input points – often tractable for existing DNN verification engines in practice.

Moreover, the size of the action space, which determines the number of output neurons, is usually small. For example, Pensieve [12], QueuePilot [41], FIRM [3], and CMARS [1] expose 6, 6, 15, and 30 actions, respectively. This is not incidental: as the number of actions increases, DRL agents require significantly more data and computation to accurately estimate action values, limiting the model's ability to learn optimal policies in high-dimensional environments [26]. This is a manifestation of the curse of dimensionality [26]. Consequently, practical DRL agents deliberately limit the action space. As a result, the number of queries generated by decomposition remains manageable.

Together, these characteristics are the reason symbolic property analysis can be feasible for DRL agents in systems and networking. By recognizing this and leveraging it through our comparative encoding and decomposition strategy, we are able to empirically explore how far symbolic analysis can be pushed for representative DRL agents in this domain (see §6-§8).

Agents with continuous action spaces. Some DRL agents in systems and networking operate over continuous-valued action spaces. In these agents, the policy DNN does not select an action via an *argmax* over discrete outputs. Instead, the DNN typically outputs the parameters of a continuous action distribution, from which the action is sampled. In practice, this distribution is most often chosen to be Gaussian, with its mean and variance produced by the DNN's output layer; in some implementations, the variance is held fixed and only the mean is learned [36]. Concretely, letting L_0 and L_1 denote the output logits corresponding to the mean and variance, respectively, the action $\pi(x)$ is sampled from the normal distribution $\mathcal{N}(L_0, L_1)$.

This results in two adjustments to the definition and analysis of our symbolic properties. First, since action selection does not involve an *argmax* operation, there is no need for output decomposition. Instead, our properties can be defined by comparing the distribution parameters produced by the DNN – which the agents use to sample their actions – and can be directly analyzed using existing DNN verification engines without decomposition. This effectively allows robustness and monotonicity properties to be defined with respect to the expected action. In other words, these

properties compare the mean outputs of the two DNN copies under related inputs, in the same spirit as our comparative encoding for discrete-action agents. Second, our symbolic properties can additionally include bounds on the distribution parameters of the first DNN copy to anchor the analysis to practically relevant regions in the action space. One example of such property is $\exists\, x^1 \in \mathcal{I}_x,\ s \in \mathcal{I}_s,\ L_0^1 \in \mathcal{I}_L : \left(|L_0^1 - L_0^2| \leq d\right)$, where L_0^1 and L_0^2 are the distribution means produced by the two respective DNN copies. We use diffRL to analyze properties for an agent with a continuous action space in §8.

5 Experimental Methodology and Solver Setup

Leveraging diffRL, we conduct a systematic study of symbolic robustness and monotonicity properties across three representative DRL agents – Pensieve [12] (§6), CMARS [1] (§7), and Aurora [36] (§8) – which span different kinds of control loops in systems and networking. This section describes the overall experimental setup for our case studies.

As described in §4, for each symbolic property, diffRL generates a collection of verification queries via comparative encoding and decomposition Our case studies examine how these query sets behave in practice: which properties can be verified, where counterexamples arise, and how different verification backends perform across the resulting sub-properties, among other aspects. We analyze the generated queries using three existing DNN verification engines presented below. Each query is analyzed with a timeout of 1000 seconds and classified as safe, unsafe, or unknown. Recall that the queries generated from decomposition express invalid cases that, if feasible, violate the property. A query is safe if it is proven infeasible for all inputs in the specified range, unsafe if the verifier finds an that makes it feasible, i.e., a counterexample that violates the symbolic property, and unknown if the verifier times out.

All experiments were conducted on an Intel(R) Xeon(R) Silver 4314 machine with a 2.40GHz CPU. To support reproducibility, we plan to release the trained models, property specifications, and verification engine configurations upon paper acceptance.

We use the following three backend DNN verification engines to analyze the queries generated by diffRL (Eq. 4) in our case studies. Different engines have distinct strengths and limitations, and their performance can depend on appropriate preprocessing and configuration choices. Accordingly, for each backend we adopt solver-aware but conventional techniques – such as query partitioning for Marabou, bound tightening before encoding for MIP, and combining input-domain splitting with output-constraint–based bound tightening for Alpha-Beta-CROWN – to ensure that the queries produced by diffRL are analyzed effectively.

Marabou (SMT-Based). Marabou [29] is an SMT-based DNN verification engine that can analyze properties specified as linear and piecewise-linear constraints. Its Deep Sum-of-Infeasibilities procedure handles piecewise-linear constraints via case analysis. Marabou also supports a Split-and-Conquer (SnC) mode, which partitions a query into independent subproblems that can be analyzed in parallel. We use Marabou v2 in SnC mode with default settings.

MIP-based. This approach encodes the DNN verification problem as a mixed-integer linear (MIP) that general-purpose solvers such as Gurobi [37] or CPLEX [24] can analyze [27]. The DNN's affine transformations are modeled using real-valued variables and linear constraints, while binary variables encode the activation state of non-linear components such as *ReLU*. To reduce the number of binary variables and improve verification time, we apply fast bound propagation techniques [38] to identify *ReLU* neurons that operate in fixed (active or inactive) regions prior to encoding each query. We use Gurobi under an academic license with 28 threads.

Alpha-Beta-CROWN (BaB-Based). This approach combines bound propagation with systematic partitioning of the input or activation domains [32]. Bound propagation computes sound

over-approximations of neuron values layer by layer under input constraints [30, 31] while branching recursively splits the domain into smaller subdomains. Subdomains that are proven to satisfy the property are pruned; the remaining ones are further partitioned until either a counterexample is found or all subdomains are verified [32].

We build on Alpha-beta-CROWN [31, 35, 38], a state-of-the-art BaB-based verifier and VNN-COMP winner (2021–2023) [47], which supports various branching and bounding strategies. In our experiments, two features were particularly important for queries generated by diffRL. Incorporating output constraints from our decomposition (Eq. 4) to tighten intermediate *ReLU* relaxations substantially improves performance, building on recent advances using Lagrangian multipliers [35]. This is especially effective because our decomposed queries often impose multiple conjunctive constraints on the output layer. We also found that branching over the input domain is more effective than branching over *ReLU* activation states. Tightening bounds using output constraints was proposed in [35] for *ReLU*-based splitting, and we extended Alpha-Beta-CROWN to support its combination with input-domain branching.

6 Case Study 1: Adaptive Video Streaming using Pensieve

Pensieve agent's DNN architecture. Adaptive bitrate (ABR) algorithms aim to improve user Quality of Experience (QoE) in video streaming by dynamically selecting the bitrate for each video segment, typically of length four seconds. Under varying network conditions, the objective is to maximize average bitrate while minimizing playback interruptions (rebuffering) and excessive bitrate fluctuations.

Pensieve [12] is a widely studied DRL-based ABR agent that makes bitrate decisions based on a compact, structured representation of the system state. Its input features comprise: (1) the bitrate selected for the previous video segment, (2) network throughput measurements for the past k segments, (3) download times for the past k segments, (4) the current playback buffer level, (5) the number of remaining video segments, and (6) the set of available bitrates out of six options for the next segment. We assume all bitrates are available. Pensieve uses a history of the previous eight steps ($k = 8$) for two of these input features, resulting in a total of 25 input variables. The policy network first computes separate embeddings for each of the six input features. These embeddings are then concatenated and passed through two fully connected (FC) layers with *ReLU* activations, producing six output logits corresponding to the available bitrate options of 300, 750, 1200, 1850, 2850, or 4300 kbps. $FC(H)$ layer contains H neurons, yielding the following architecture:

$$\begin{aligned}\pi_H^{\text{Pensieve}} : \texttt{Input}(25) &\xrightarrow{splitting} [\texttt{Input}(1), \texttt{Input}(8), \texttt{Input}(8), \texttt{Input}(1), \texttt{Input}(1), \texttt{Input}(6)] \\ &\rightarrow [\texttt{FC}(H),\ \texttt{FC}(H),\ \texttt{FC}(H),\ \texttt{FC}(H),\ \texttt{FC}(H),\ \texttt{FC}(H)] \xrightarrow{concatenate} ReLU \\ &\rightarrow \texttt{FC}(H) \rightarrow ReLU \rightarrow \texttt{FC}(6) \xrightarrow{argmax} \texttt{Output}(1)\end{aligned}$$

Symbolic Property 1: Capacity Utilization (Monotonicity). If the available network throughput increases, an adaptive bitrate algorithm should not respond by selecting a lower video bitrate. We refer to this expected behavior as "Capacity Utilization". We express this as a symbolic monotonicity property (See Eq. 1 and Eq. 3) by constraining the slack vector so that only the input feature corresponding to measured throughput increases by at most ϵ while keeping all other input features unchanged. The property is violated if there exists an input state for which an increase in measured throughput causes the selected bitrate to decrease by more than d levels.

Symbolic Property 2: Rebuffering Avoidance (Monotonicity). When the playback buffer contains only a small number of video segments, the system is more vulnerable to transient throughput drops, making aggressive bitrate choices more likely to cause buffer depletion. In such situations, an adaptive bitrate algorithm should act conservatively. We refer to this expected behavior as "Rebuffering Avoidance". Intuitively, for the same network conditions, the bitrate

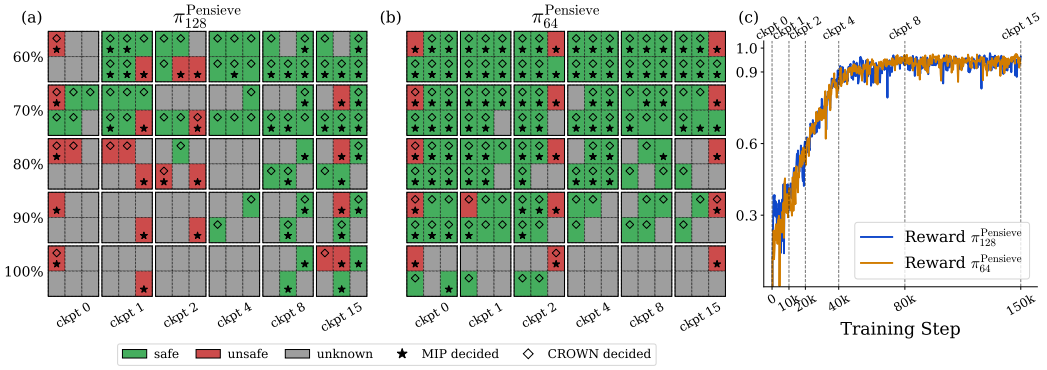

Fig. 4. (a) and (b) illustrate the analysis results for the *Capacity Utilization* property for DRL agents $\pi_{128}^{\text{Pensieve}}$ and $\pi_{64}^{\text{Pensieve}}$, respectively. Each group of six cells represents the results for a specific model checkpoint (ckpt) throughout training (x-axis) and a specific per–input-feature coverage percentage (y-axis). Each cell in the group cell corresponds to one of the queries generated by diffRL through property decomposition. (c) presents the training reward curves for $\pi_{128}^{\text{Pensieve}}$ and $\pi_{64}^{\text{Pensieve}}$, with vertical dashed lines indicating the checkpoints at which models are extracted for analysis.

selected when the buffer is sparsely filled should not exceed the bitrate selected when the buffer is well filled. We capture this also as a symbolic monotonicity property (See Eq. 1 and Eq. 3) by constraining the slack vector so that only the input feature corresponding to buffer occupancy is allowed to increase and by at most ϵ. The property is violated if there exists an input state for which increasing buffer occupancy leads to a decrease in the selected bitrate by more than d levels.

Symbolic Property 3: Pensieve Robustness. In practice, input features such as measured throughput and download time are subject to noise and small transient fluctuations. An adaptive bitrate algorithm should therefore avoid reacting to such minor perturbations with large changes in selected bitrate, as this can lead to unstable user experience. We express this as a symbolic robustness property similar to Eq. 2 by allowing small bounded perturbations (between $-\epsilon$ to ϵ) across the input features. The property is violated if there exists an input state for which these perturbations cause the selected bitrate to change by more than d levels.

6.1 Analysis Results

We use diffRL to analyze the above symbolic properties for two configurations of Pensieve's policy network, with hidden layer sizes $H = 128$ and $H = 64$. For all properties, we set the perturbation bound to $\epsilon = 0.01$ and the tolerance to $d = 3$ bitrate levels. With this choice of d, diffRL generates 6 invalid output neuron pairs – and thus 6 verification queries[1] – for each monotonicity property and 12 for the robustness property, corroborating our insight that the number of generated queries from decomposition remains manageable in practice (§4). Each invalid pair is analyzed independently using the backend DNN verification engines. Due to architectural constraints, not all backends are applicable to Pensieve. In particular, Pensieve's policy network computes separate embeddings for each input feature via input slicing, which is unsupported by Marabou. Consequently, we analyze Pensieve's properties using the remaining two verification backends.

Figure 4 shows verification outcomes for the 6 queries generated for the Capacity Utilization property for the two Pensieve policies with different hidden-layer sizes of 128 ($\pi_{128}^{\text{Pensieve}}$) and 64 ($\pi_{64}^{\text{Pensieve}}$). In Figure 4(a) and (b), the x-axis corresponds to the model at a specific training checkpoint, where ckpt 0 refers to the randomly initialized policy before any training and ckpt i corresponds to

[1] $(720, 240)$, $(1080, 240)$, $(1080, 360)$, $(1440, 240)$, $(1440, 360)$, $(1440, 480)$

the model checkpoint at $i \times 10$k training steps. The y-axis corresponds to the level of input-domain coverage, ranging from 60% to 100% of each feature's entire operational range[2]. For a certain checkpoint and coverage range, six individual cells are grouped together to represent the results for the property's six queries. Cell colors indicate verification outcomes for individual decomposed queries (safe, unsafe, or unknown), where unsafe results correspond to concrete counterexamples in which higher measured throughput leads to a lower selected bitrate and unknown is reported when the verification engines timeout before returning a conclusive result. The symbols ★ and ⋄ indicate whether MIP and Alpha-Beta-Crown successfully resolved the query, respectively.

Impact of input coverage. Across both model sizes, diffRL successfully resolves most queries at coverage levels 60–80%, producing safe and unsafe outcomes. As coverage increases toward 100%, the fraction of unknown results grows, reflecting the increase in problem difficulty due to the larger input domain and expansion of the search space. Nevertheless, even in these challenging regimes, the analysis continues to yield conclusive results. Overall, these results demonstrate that symbolic property checking for DRL agents in systems and networking can be feasible and informative in practice, and can provide substantially broader coverage than individual point properties.

Training checkpoints reveal evolving property satisfaction. For $\pi_{128}^{\text{Pensieve}}$, early training checkpoints exhibit a higher proportion of unsafe outcomes, indicating violations of the Capacity Utilization property in the initial stages of training. As training progresses, the fraction of safe outcomes increases, suggesting that the learned policy increasingly aligns with monotonic bitrate adaptation as throughput increases. Notably, a small number of violations persist even at the final checkpoints, and some queries that were previously verified as safe become unsafe after further training. This can be explained by the fact that continued updates to the model parameters during training change the policy's decision behavior, which can invalidate earlier safety results.

Impact of model size on verifiability and reward. Although $\pi_{128}^{\text{Pensieve}}$ with 103 430 parameters and $\pi_{64}^{\text{Pensieve}}$ with 27 142 parameters follow nearly identical reward trajectories (Figure 4 (c)), their verification outcomes differ substantially. The smaller model produces approximately 45% fewer unknown results, indicating that reduced model size can significantly improve symbolic verifiability without sacrificing the agent's effectiveness and performance. As such, this underscores the value of symbolic analysis as a complementary evaluation lens for DRL agents in systems and networking.

Benefits of multi-engine verification. Aggregating the outcomes across multiple verification engines substantially reduces the number of unknown results compared to relying on any single engine alone. For $\pi_{128}^{\text{Pensieve}}$, $\sim$ 60% of the resolved queries are decided by only one of the engines and the other timed out, whereas this fraction is about 35% for $\pi_{64}^{\text{Pensieve}}$. These observations highlight the complementary strengths of different solvers and the necessity of multi-engine verification for enabling the analysis of symbolic properties, as relying on a single backend would leave many properties unresolved. This is especially true for larger models, whose verification search space is considerably more complex than that of smaller models.

Other Pensieve properties. Figure 5 reports the analysis results for Rebuffering Avoidance and Robustness for both $\pi_{64}^{\text{Pensieve}}$ and $\pi_{128}^{\text{Pensieve}}$ across training checkpoints and multiple input coverage levels. In contrast to Figure 4, which presents fine-grained, per-query outcomes, this figure aggregates the results at each checkpoint and coverage level, showing the total number of queries classified as safe, unsafe, and unknown.

Similar to Capacity Utilization, for both properties and both model sizes, most queries are conclusively resolved at 60% and 80% coverage, with a clear dominance of safe outcomes, especially for Rebuffering Avoidance. As the coverage increases to 100%, the fraction of unknown results

[2]if an input feature's entire range is (a, b), the queries corresponding to the 60% level restrict this feature to the interval $(a + 0.2(b - a), b - 0.2(b - a))$

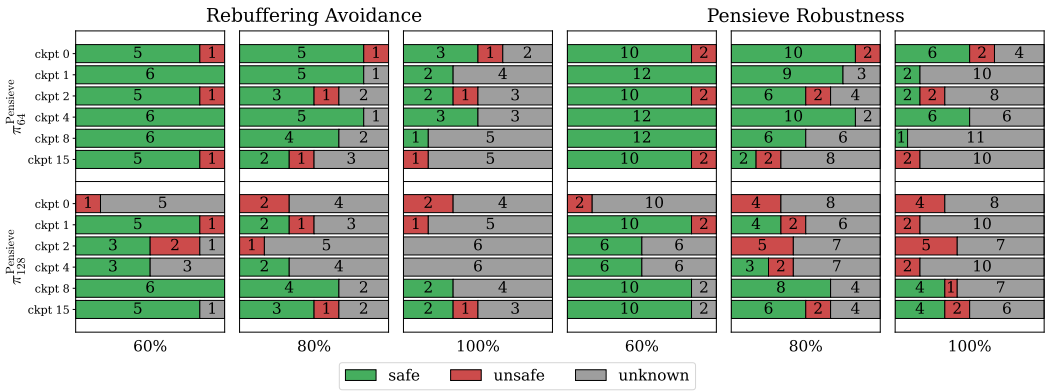

Fig. 5. Analysis results for Rebuffering Avoidance (left) and Robustness (right) for $\pi^{\text{Pensieve}}_{64}$ (top) and $\pi^{\text{Pensieve}}_{128}$ (bottom) at different training checkpoints (rows) from Figure 4(c). Each column corresponds to a coverage level of the input domain (60%, 80%, and 100%). For each model checkpoint and coverage, the stacked horizontal bars show the total number of verification queries classified as safe (green), unsafe (red), and unknown (gray), with the exact counts annotated inside each segment.

grows substantially, particularly for the Robustness property. This mirrors the trend observed for Capacity Utilization, reflecting the rapid growth of the verification search space as the input domain expands. Nevertheless, even at full coverage, diffRL continues to identify concrete safe and unsafe behaviors, highlighting the benefits of symbolic properties.

7 Case Study 2: Wireless Resource Allocation using CMARS

CMARS agent's DNN architecture. Next-generation mobile networks allocate wireless resources across multiple network slices, where each slice groups users with similar service requirements and is governed by a service-level agreement (SLA). The network operator must allocate radio resources efficiently while ensuring that each slice meets the performance guarantees of its SLA.

We study CMARS [1], a DRL agent that dynamically assigns radio resource blocks to individual slices with the objective of minimizing total resource usage while satisfying slice-level SLAs. Its input features comprise: (1) historical performance statistics of the target slice, (2) current network quality, represented by the average signal-to-noise ratio (SNR) between users and the base station, computed per slice, (3) the amount of available radio resources, and (4) aggregated statistics from other slices, which include the number of Internet-of-Things users, the average traffic of constant-bitrate users, and the average traffic of variable-bitrate users. All input features are normalized to the range $[0, 1]$ based on expected operational limits. CMARS outputs a discrete action corresponding to the number of radio resource blocks allocated to the target slice, ranging from zero to the total number of available blocks M.

We analyze CMARS models with two architectural variants – using either two or three fully connected layers – and two action-space sizes, with $M \in \{15, 30\}$ possible allocation levels. Each fully connected layer contains 32 neurons with ReLU activations, yielding the following architectures:

$$\pi^{\text{CMARS}}_{2,M} : \texttt{Input(19)} \to \texttt{FC(32)} \to \mathit{ReLU} \to \texttt{FC(32)} \to \mathit{ReLU} \to \texttt{FC}(M) \xrightarrow{\mathit{argmax}} \texttt{Output(1)}$$

$$\pi^{\text{CMARS}}_{3,M} : \texttt{Input(19)} \to \texttt{FC(32)} \to \mathit{ReLU} \to \texttt{FC(32)} \to \mathit{ReLU} \to \texttt{FC(32)} \to \mathit{ReLU} \to \texttt{FC}(M) \xrightarrow{\mathit{argmax}} \texttt{Output(1)}$$

Symbolic Property 1: Contention-Aware Allocation (Monotonicity). In a sliced wireless network, increased resource demand from other slices places additional strain on the shared radio resources. As such, for a fixed target slice, CMARS should not increase its allocation when competing slices experience higher demand, and should ideally reduce it. We refer to this expected behavior as "Contention-Aware Allocation". We encode this as a symbolic monotonicity property (Eq. 3),

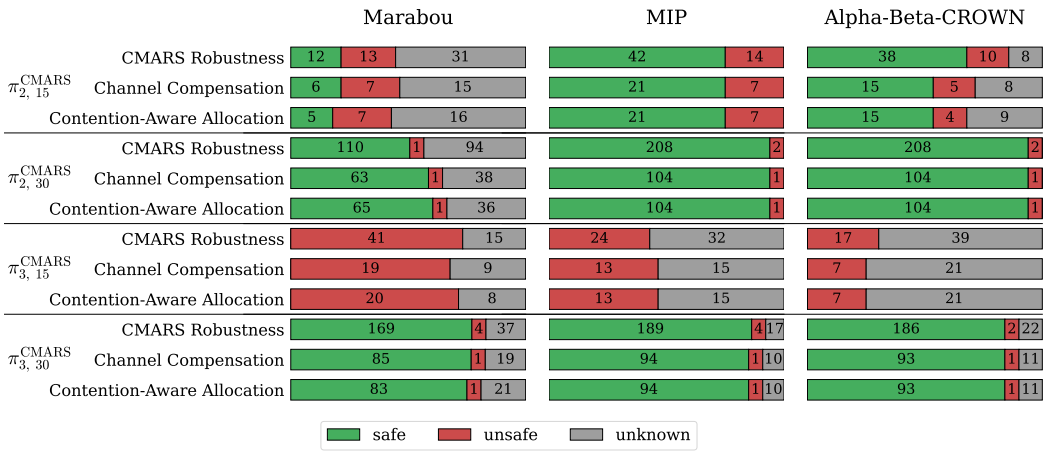

Fig. 6. Comparison of analysis outcomes across solvers and properties. The figure reports the number of safe, unsafe, and unknown results obtained using Marabou, MIP, and Alpha-Beta-CROWN for queries generated by diffRL for three properties evaluated on different CMARS policies. Each horizontal bar corresponds to a property-policy pair, with segment lengths and overlaid counts indicating the solver's outcome distribution.

using a slack vector that permits only increases up to ϵ in the input features capturing aggregated demand from other slices, while the other features remain unchanged. The property is violated if there exists an input state in which increased cross-slice demand causes the second DNN copy to select an allocation that exceeds that of the first by more than d resource units.

Symbolic Property 2: Channel Compensation (Monotonicity). Poor channel conditions reduce the effective bitrate per radio resource block. To maintain slice-level SLAs under such conditions, a resource allocation policy should compensate by allocating additional radio resources to the affected slice. We refer to this expected behavior as "Channel Compensation". We encode it as a symbolic monotonicity property (Eq. 3), where the slack vector only allows decreases up to ϵ in the input feature capturing channel quality (e.g., average SNR) while the rest of the features remain unchanged. The property is violated if there exists an input state in which a degradation in channel quality causes the second DNN copy to select an allocation that is lower than that of the first by more than d resource units.

Symbolic Property 3: CMARS Robustness. In practice, CMARS input features can be subject to measurement noise and small transient fluctuations. A well-behaved resource allocation policy should therefore avoid large changes in allocated resources in response to minor perturbations of its inputs, as such sensitivity can lead to unstable behavior and inefficient resource usage. Moreover, excessive sensitivity can expose the system to adversarial manipulation, where small input changes trigger disproportionate resource allocations. We formalize this as a symbolic robustness property (Eq. 2), in which all input features are allowed to vary within a small bounded range (ϵ), and the property is violated if there exists an input state for which these perturbations cause the second DNN copy to select an allocation that differs from that of the first by more than d resource units.

7.1 Analysis Results

We use diffRL to analyze the symbolic properties defined above across the two CMARS architecture, $\pi^{\text{CMARS}}_{2,\,M}$ and $\pi^{\text{CMARS}}_{3,\,M}$, described above, for $M = 15$ and $M = 30$. For all properties, we set the perturbation bound to $\epsilon = 0.01$. We use tolerance values of $d = 8$ for $M = 15$ and $d = 16$ for $M = 30$ resource units, corresponding to moderate and large allocation changes relative to the action-space size. With these parameters, diffRL generates 28 and 105 invalid output neuron pairs for the two

monotonicity properties when $M = 15$ and $M = 30$, respectively; the number of queries is doubled for the robustness property. This again highlights that, while the number of queries grows with the action space, decomposition remains tractable in practice.

Figure 6 summarizes the analysis outcomes for CMARS across different properties, policy configurations, and verification backends. Each horizontal bar corresponds to a property–policy pair, with colored segments showing how many of the generated queries each solver classifies as safe, unsafe, or unknown.

Significance of counterexamples. For both *Contention-Aware Allocation* and *Channel Compensation*, the identified counterexamples correspond to corner-case but operationally meaningful scenarios that appear to be underrepresented in the training data. These cases highlight potential weaknesses in the model's generalization that are unlikely to surface through standard evaluation metrics. The robustness counterexamples are particularly striking. In one case, a perturbation of $L_\infty = 0.001$ caused a change of 16 resource units, more than half of the total available blocks in the 30-action model. Such a disproportionate response to a minor input change indicates a high sensitivity to small fluctuations and highlights potential vulnerability to measurement noise or adversarial manipulation.

These counterexamples illustrate the practical value of symbolic analysis: they expose rare but impactful behaviors that are difficult to uncover through point-based analysis alone. We envision leveraging such counterexamples to guide targeted retraining and robustness-aware learning, as explored in recent work on counterexample-guided DRL refinement [48, 49] (see §9).

Deeper networks are not necessarily safer. Policies $\pi^{\text{CMARS}}_{2,30}$ and $\pi^{\text{CMARS}}_{3,30}$ exhibit the highest levels of property compliance across the evaluated criteria, with relatively few violations and a large fraction of safe outcomes, suggesting that these models behave reliably with respect to the analyzed properties. In contrast, $\pi^{\text{CMARS}}_{3,15}$ shows substantially more violations and a higher number of unknown results, particularly for the *CMARS Robustness* property. Notably, this model performs considerably worse than its "shallower" counterpart $\pi^{\text{CMARS}}_{2,15}$. These results indicate that increasing network depth does not necessarily improve symbolic property compliance. Moreover, deeper models tend to be more computationally challenging to analyze, as reflected by the larger fraction of unknown outcomes, echoing a similar trend observed in the Pensieve case study.

Benefits of multi-engine verification. For smaller CMARS models ($\pi^{\text{CMARS}}_{2,\,15}$ and $\pi^{\text{CMARS}}_{2,\,30}$), the MIP-based approach is the only one that resolves all queries without timeouts, demonstrating its reliability for compact DNNs in this case study. As model complexity increases, for $\pi^{\text{CMARS}}_{3,\,15}$ and $\pi^{\text{CMARS}}_{3,\,30}$, MIP begins to encounter scalability limitations and times out on a subset of queries. In this regime, different engines expose complementary strengths For $\pi^{\text{CMARS}}_{3,\,15}$, Marabou identifies up to 30% more violations compared to MIP, and performs better than Marabou for the more compact $\pi^{\text{CMARS}}_{2,\,15}$ counterpart with over 1000 fewer parameters. Conversely, for $\pi^{\text{CMARS}}_{3,\,30}$, MIP proves more queries to be safe than Marabou while detecting the same number of violations. Alpha-beta-CROWN terminates on all queries for $\pi^{\text{CMARS}}_{3,\,30}$, which meets the properties in most cases. While it verifies a larger fraction of queries as safe, it detects fewer violations than Marabou. This trend is consistent across most CMARS configurations.

Overall, these results reinforce the benefits of a general symbolic property formulation and decomposition strategy that is compatible with multiple verification engines. Different backends excel at different aspects of the verification task, and relying on a single engine would leave a non-trivial fraction of queries unresolved.

8 Case Study 3: Congestion Control using Aurora

Aurora agent's DNN architecture. Aurora [36] is a DRL-based congestion control algorithm that operates over a continuous action space, making it qualitatively different from the discrete-action

agents studied earlier. The agent observes recent traffic statistics and continuously adjusts the sender's rate in response. At each decision step, Aurora's policy network takes as input a vector of recent measurements capturing: (1) the latency ratio (current latency normalized by the minimum observed latency), (2) the packet acknowledgment ratio (packets acknowledged normalized by packets sent) by destination, and (3) the latency gradient, indicating whether latency is increasing or decreasing. These signals are collected over the most recent k decision steps, yielding an input of size $3k$. The original work shows that a model with $k = 2$ performs similar to one with $k = 10$. We select $k = 3$ to keep the size of the model low.

At each decision step t, Aurora's DNN outputs the mean of a normal distribution from which a continuous action a_t is sampled – the standard deviation is fixed to a constant, $\sigma = 0.5$ in this case. The sign of a_t determines the direction of rate adjustment (increase or decrease) and its magnitude controls the adjustment extent. Specifically, if the previous sending rate is x_{t-1}, the updated rate x_t is set to $x_{t-1} \cdot (1 + \alpha a_t)$ for $a_t \geq 0$, and to $x_{t-1}/(1 - \alpha a_t)$ otherwise. Finally, the original Aurora implementation uses hyperbolic tangent activations, which are incompatible with many DNN verification engines. Following prior verification work [21], we replace these with ReLU activations in the following architecture, and use a retrained model from [50] that obtains comparable performance:

$$\pi_{128}^{\text{Aurora}} : \texttt{Input(9)} \rightarrow \texttt{FC(128)} \rightarrow \textit{ReLU} \rightarrow \texttt{FC(128)} \rightarrow \textit{ReLU} \rightarrow \texttt{FC(128)} \rightarrow \texttt{Output(1)}$$

Property 1: Ack-Driven Capacity Utilization (Monotonicity). An increasing packet acknowledgment ratio indicates that a larger fraction of transmitted packets is successfully delivered, reflecting favorable network conditions. Under such conditions, Aurora should not reduce its sending rate, and should ideally increase it. We refer to this expected behavior as "Ack-Driven Capacity Utilization". We encode this as a symbolic monotonicity property by allowing a positive slack (at most ϵ) only on the packet acknowledgment ratio input features and checking if the resulting rate adjustment *reverses direction* from increasing to decreasing.

Because Aurora selects actions by sampling from a distribution whose mean is produced by the DNN, as described in §4, we define the property over the DNN outputs that determine these means (the standard deviation is fixed to σ). Concretely, we flag a potential violation when the means of the two DNN copies are separated such that the sign of the sampled action would differ with non-negligible probability across executions. Formally, following the notation in §3 and §4, we set $f(x, y) = x - y$, where x and y will be L_0^1 and L_0^2, the respective output means of the two DNN copies, set $d = 2 \times \mu$, and set the bound for L_0^1 to $[\mu, \infty)$. Note that this effectively means the "invalid" outputs occur when $L_0^1 \geq \mu \wedge L_0^2 \leq -\mu$. Under standard distributional assumptions, if the above inequalities hold, the probability of the selected action's sign changing from positive in the first DNN copy to negative in the second is $\Phi(\frac{\mu}{\sigma})^2$, where Φ is the standard normal CDF (§A). In our experiments, we set it to half the standard deviation $\frac{\sigma}{2}$, giving a non-negligible probability, $\sim 40\%$, of a change in action direction.

Property 2: Latency-Aware Capacity Utilization (Monotonicity). A lower latency ratio indicates that the current end-to-end latency is close to the minimum observed latency, suggesting lighter-filled queues along the path. As such, when the latency ratio decreases, Aurora should not reduce its sending rate, and should ideally increase it to better utilize available bandwidth. We refer to this expected behavior as "Latency-Aware Capacity Utilization". We encode this similar to the previous property, but with negative slack for the latency ratio inputs and check if the rate adjustment would change direction from increasing to decreasing.

Property 3: Aurora Robustness. This property captures the expectation that small perturbations in Aurora's observed state – arising from noise, measurement variability, or transient fluctuations – should not cause qualitatively different control decisions. Specifically, we check if bounded perturbations (up to ϵ) *across all input features*, including historical observations, can

Property	Coverage	Marabou	MIP	Alpha-Beta-CROWN
Aurora Robustness	70%	safe	safe	safe
	100%	unknown	unsafe	unknown
Ack-Driven Capacity Utilization	70%	safe	safe	safe
	100%	unknown	unsafe	unknown
Latency-Aware Capacity Utilization	70%	safe	safe	safe
	100%	unknown	unsafe	unknown

Table 1. Aurora analysis results for different solvers and per-input-feature coverage range.

cause a reversal in the direction of the rate adjustment. Because robustness concerns both possible direction changes, we consider two symmetric cases: a change from increasing to decreasing, and the converse. Each case is analyzed separately using the same comparative encoding as in the monotonicity properties.

Analysis Results. Table 1 summarizes the analysis outcomes for Aurora's symbolic properties under a perturbation bound of $\epsilon = 0.01$, considering both 70% and 100% coverage of each input feature. Similar to Pensieve (§6), we observe that all engines can resolve the queries for 70% per-input-feature coverage, with all three properties verified as safe. As coverage increases to 100%, the MIP-based approach is the only method that resolves all queries before the time-out period, identifying concrete unsafe behaviors for all three properties.

Besides demonstrating how our approach can extend to continuous action spaces, these results reinforce two broader insights. First, analysis coverage over the input space plays a critical role in revealing problematic behaviors. Second, with the current state-of-the-art verification techniques, relying on a single engine is not sufficient for analyzing symbolic properties. A generic symbolic property formulation that is compatible with multiple solvers enables users to push anaylzing such properties over as wide ranges as possible with existing verification techniques.

9 Discussion and Future Work

Broader applicability to DRL agents in networked systems While our evaluation focuses on three representative DRL agents, the symbolic property formulation enabled by diffRL applies more broadly to DRL agents in networked systems. We briefly illustrate how similar monotonicity and robustness properties naturally arise in other settings. Consider QueuePilot [41], which is a DRL-based Active Queue Management agent aiming to address the challenge of managing small buffers in backbone routers. It controls the probability of Explicit Congestion Notification (ECN) marking to balance link utilization, packet loss, and queueing delay. Its input features capture traffic intensity, link utilization, proportion of marked packets, and queue occupancy, delay, and loss statistics. Its discrete action space consists of a set of ECN marking probabilities. In this setting, natural symbolic monotonicity properties arise: for example, the ECN marking probability should increase as queue length, delay, or drop rate increase. Similarly, robustness properties are desirable to prevent small fluctuations in traffic measurements from causing large oscillations in marking behavior. These properties can be directly encoded using diffRL and analyzed with existing verification engines.

A second example is FIRM [3], a DRL agent to dynamically adjust resource allocations across CPU, memory, cache, disk, and network bandwidth to prevent SLA violations in microservices. The agent's input features include workload characteristics, resource utilization, and SLA satisfaction metrics. Here, symbolic monotonicity properties naturally express expectations such as allocating fewer resources as SLA satisfaction improves or resource utilization decreases, while robustness properties capture stability under small measurement noise. These properties can be encoded using

diffRL and, given FIRM's relatively small network size, we expect of them to be tractable to analyze using existing verification engines.

Other DNN architectures. Consistent with prior verification-based studies in this domain [19, 21], our evaluation focuses on policy networks with fully connected (and convolutional) layers and ReLU activations. Nevertheless, our symbolic property formulation and comparative encoding are architecture-agnostic: they rely only on comparing the outputs of two related executions of the same policy. As such, the same properties apply to agents using other architectures or activations (e.g., RNNs, tanh, LeakyReLU), which appear in some systems and networking DRL agents [41]. Extending analysis to these models primarily depends on backend solver support, which continues to improve (e.g., Alpha-Beta-CROWN [51]).

Verification-aware DRL design and property enforcement. As demonstrate by our case studies, the architecture of a DRL policy's DNN impacts its verifiability, particularly choices such as activation functions and network size. When these choices do not compromise performance, using piecewise-linear activations and more compact networks can substantially simplify symbolic analysis. Moreover, analysis of symbolic properties can help inform the enforcement of desired behaviors. When violations are discovered, the resulting counterexamples can be incorporated into training to reinforce expected behavior [48, 49]. Alternatively, symbolic property compliance can be potentially used as an auxiliary cost metric in a constrained DRL formulation. Finally, monotonicity can be enforced directly through architectural constraints on the policy network, leveraging recent advances in monotonic neural networks from the supervised learning literature [52, 53].

Probabilistic action selection. DRL agents can use a $softmax$ layer to convert output logits into a probability distribution over discrete actions, enabling stochastic exploration and uncertainty-aware decision making. However, the $softmax$ function is neither linear nor piecewise linear, making it difficult to encode directly using existing DNN verification techniques. In settings where the final action is selected deterministically via an argmax, this challenge can be avoided by reasoning directly about the ordering of logits, as we do in this work. However, omitting $softmax$ precludes reasoning about action probabilities themselves, which is useful for agents that *select one of the discrete actions probabilistically*. Extending our approach to such cases is an important open direction, and recent work on probabilistic and convex relaxations of $softmax$-based policies offers promising building blocks toward this goal [54].

Non-Numerical action spaces. Most DRL agents used in systems and networking operate over *numerical action spaces*, where actions correspond to ordered control values such as rates, resource allocations, or thresholds, and naturally support comparisons and trends such as monotonicity and bounded change [1, 3, 12, 14, 15, 41, 42, 44]. Extending our approach to DRL agents with non-numerical or unordered action spaces – such as categorical decisions without an inherent ordering [8, 39, 40] – is an interesting direction for future work.

10 Related Work

Verifying DRL agents in systems and networking. Several recent works have explored using general-purpose MIP solvers and DNN verification engines to analyze DRL-based control policies in systems and networking. WhiRL [21] uses the Marabou SMT-based verifier to check safety and liveness properties of Pensieve under *fixed, extreme input conditions*, such as excellent or worst-case network states. Similarly, Dethise et al. [19] and UINT [22] encode DRL policies as MIP or SMT problems to verify *local robustness properties* around concrete input–output pairs. These approaches demonstrate the feasibility of applying formal verification tools to DRL agents. However, they are inherently limited to *point-based* or local properties defined around specific inputs, which provide limited coverage of the agent's operational input space (see Figure 1). Our work enables analyzing symbolic properties defined over ranges of inputs rather than individual points.

Empirical analysis and interpretability without formal guarantees. A complementary line of work focuses on understanding or stress-testing DRL agents through empirical or interpretability-based techniques. Metis [55] approximates DRL policies with decision trees to derive human-interpretable rules, but at the cost of reduced faithfulness to the original model; for instance, the authors report a faithfulness of only 84% with respect to the original DNN. Other approaches use DRL to synthesize network conditions under which a given algorithm underperforms [56], or use active learning to automate the performance evaluation of congestion control schemes [57]. In [58], the authors propose a robustness metric for a DRL-based controller, defined as the ratio of states where the DNN is locally robust to the total number of sampled states. Finally, [59] applies interpretability tools to analyze model inputs and identify anomalous or undesirable behaviors. While effective for uncovering performance or effectiveness issues, they do not provide formal guarantees of safety or correctness of input regions.

Robustness via training-time regularization. Shen et al. [60] propose a training-time regularization technique that encourages smoother DRL policies by penalizing differences between actions taken under nominal and perturbed state inputs. Such robustness-oriented regularization can empirically reduce sensitivity to input noise, but it does not provide formal guarantees about the resulting policy's behavior. Our verification-based approach is complementary: rather than modifying the training objective, it enables post-training, formal assessment of whether a learned policy satisfies robustness and other safety properties over specified input regions.

Sensitivity analysis using Lipschitz-based methods. The Lipschitz constant has been proposed as a measure of neural network sensitivity [61]. Given a function f, it bounds the maximum change in the norm of the output vector relative to changes in the norm of the input. When f has a multi-dimensional output, this bound is defined over norms of the full output vector and does not track changes in the induced decision rule – e.g., the *argmax* over action logits, which determines the DRL agent's action. Methods such as AutoLip [61] compute upper bounds on the Lipschitz constant of a DNN using Jacobian-based analysis, and are therefore well-suited for reasoning about numerical output sensitivity. However, in DRL policies with discrete action spaces, where actions are selected via an *argmax* over output neurons, small changes in the logits – well within a Lipschitz bound – can still lead to different action selections. As a result, global Lipschitz bounds on network outputs do not directly characterize the stability of the resulting control decisions in DRL agents considered by this work.

Formal methods in systems and networking. Formal methods have been extensively applied to non-ML-based systems and networking algorithms [62–66]. Our work contributes to the emerging effort [19, 21, 67] to bring similar rigor to DRL-based systems, complementing prior verification efforts while expanding their applicability to symbolic, range-based properties.

11 Conclusion

Deep reinforcement learning is increasingly used in control loops in systems and networking, yet reasoning about agent behavior beyond individual input points remains difficult. In this work, we studied symbolic properties that capture expected behaviors over ranges of system states and showed how they can be analyzed using existing DNN verification engines via comparative encoding and decomposition. Through an extensive empirical study using our framework, diffRL, across adaptive video streaming, wireless resource allocation, and congestion control, we demonstrated that symbolic analysis substantially broadens coverage beyond point-based checks, uncovers non-obvious counterexamples, and exposes practical trade-offs related to training, model size, and solver choice. These results clarify both the promise and capacity of symbolic property analysis for DRL agents in systems and networking.

References

[1] Mohammad Zangooei, Morteza Golkarifard, Mohamed Rouili, Niloy Saha, and Raouf Boutaba. Flexible ran slicing in open ran with constrained multi-agent reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 2023.

[2] Jinwoo Park, Jaehyeong Park, Youngmok Jung, Hwijoon Lim, Hyunho Yeo, and Dongsu Han. Topfull: An adaptive top-down overload control for slo-oriented microservices. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 876–890, 2024.

[3] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pages 805–825, 2020.

[4] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. Cellular network traffic scheduling with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[5] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 191–205, 2018.

[6] Zhiying Xu, Francis Y Yan, Rachee Singh, Justin T Chiu, Alexander M Rush, and Minlan Yu. Teal: Learning-accelerated optimization of wan traffic engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 378–393, 2023.

[7] Fei Gui, Songtao Wang, Dan Li, Li Chen, Kaihui Gao, Congcong Min, and Yi Wang. Redte: Mitigating subsecond traffic bursts with real-time and distributed traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 71–85, 2024.

[8] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288. 2019.

[9] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56, 2016.

[10] Suraj Jog, Zikun Liu, Antonio Franques, Vimuth Fernando, Sergi Abadal, Josep Torrellas, and Haitham Hassanieh. One protocol to rule them all: Wireless {Network-on-Chip} using deep reinforcement learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 973–989, 2021.

[11] Woo-Hyun Ko, Ushasi Ghosh, Ujwal Dinesha, Raini Wu, Srinivas Shakkottai, and Dinesh Bharadia. EdgeRIC: Empowering real-time intelligent optimization and control in NextG cellular networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1315–1330, Santa Clara, CA, April 2024. USENIX Association.

[12] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.

[13] Lianchen Jia, Chao Zhou, Tianchi Huang, Chaoyang Li, and Lifeng Sun. Rdladder: Resolution-duration ladder for vbr-encoded videos via imitation learning. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[14] Lianchen Jia, Chao Zhou, Tianchi Huang, Chaoyang Li, and Lifeng Sun. Dancing with shackles, meet the challenge of industrial adaptive streaming via offline reinforcement learning. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 2169–2178. IEEE, 2024.

[15] Siyu Yan, Xiaoliang Wang, Xiaolong Zheng, Yinben Xia, Derui Liu, and Weishan Deng. Acc: Automatic ecn tuning for high-speed datacenter networks. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 384–397, 2021.

[16] Chen Tessler, Yuval Shpigelman, Gal Dalal, Amit Mandelbaum, Doron Haritan Kazakov, Benjamin Fuhrer, Gal Chechik, and Shie Mannor. Reinforcement learning for datacenter congestion control. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):43–46, 2022.

[17] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 632–647, 2020.

[18] Yiqing Ma, Han Tian, Xudong Liao, Junxue Zhang, Weiyan Wang, Kai Chen, and Xin Jin. Multi-objective congestion control. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 218–235, 2022.

[19] Arnaud Dethise, Marco Canini, and Nina Narodytska. Analyzing learning-based networked systems with formal verification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[20] Zikun Liu, Changming Xu, Yuqing Xie, Emerson Sie, Fan Yang, Kevin Karwaski, Gagandeep Singh, Zhao Lucis Li, Yu Zhou, Deepak Vasisht, et al. Exploring practical vulnerabilities of machine learning-based wireless systems. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1801–1817, 2023.

[21] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[22] Yangfan Huang, Yuling Lin, Haizhou Du, Yijian Chen, Haohao Song, Linghe Kong, Qiao Xiang, Qiang Li, Franck Le, and Jiwu Shu. Toward a unified framework for verifying and interpreting learning-based networking systems. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 01–10. IEEE, 2023.

[23] Marabou. https://github.com/NeuralNetworkVerification/Marabou, v2.0.0. Accessed: September, 2024.

[24] IBM Corporation. *IBM ILOG CPLEX Optimization Studio*. IBM, 2019.

[25] Shuowei Jin, Francis Y Yan, Cheng Tan, Anuj Kalia, Xenofon Foukas, and Z Morley Mao. Autospec: Automated generation of neural network specifications. *arXiv preprint arXiv:2409.10897*, 2024.

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

[27] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[28] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

[29] Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, et al. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, pages 249–264. Springer, 2024.

[30] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[31] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

[32] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

[33] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020.

[34] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 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.

[35] Suhas Kotha, Christopher Brix, J Zico Kolter, Krishnamurthy Dvijotham, and Huan Zhang. Provably bounding neural network preimages. *Advances in Neural Information Processing Systems*, 36:80270–80290, 2023.

[36] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[37] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2024.

[38] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

[39] Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, Mohammad Alizadeh, et al. Learning generalizable device placement algorithms for distributed machine learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Yi Hu, Chaoran Zhang, Edward Andert, Harshul Singh, Aviral Shrivastava, James Laudon, Yanqi Zhou, Bob Iannucci, and Carlee Joe-Wong. Giph: Generalizable placement learning for adaptive heterogeneous computing. *Proceedings of Machine Learning and Systems*, 5, 2023.

[41] Micha Dery, Orr Krupnik, and Isaac Keslassy. Queuepilot: Reviving small buffers with a learned aqm policy. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[42] Jiaxin Tang, Sen Liu, Yang Xu, Zehua Guo, Junjie Zhang, Peixuan Gao, Yang Chen, Xin Wang, and H Jonathan Chao. Abs: Adaptive buffer sizing via augmented programmability with machine learning. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 2038–2047. IEEE, 2022.

[43] Alessandro Montenegro, Marco Mussi, Alberto Maria Metelli, and Matteo Papini. Learning optimal deterministic policies with stochastic policy gradients. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, Vienna, Austria, 2024. JMLR.org.

[44] Zibo Wang, Pinghe Li, Chieh-Jan Mike Liang, Feng Wu, and Francis Y Yan. Autothrottle: A practical {Bi-Level} approach to resource management for {SLO-Targeted} microservices. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 149–165, 2024.

[45] Mowei Wang, Sijiang Huang, Yong Cui, Wendong Wang, and Zhenhua Liu. Learning buffer management policies for shared memory switches. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 730–739, 2022.

[46] Janosch Moos, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, and Jan Peters. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315, 2022.

[47] Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. *arXiv preprint arXiv:2312.16760*, 2023.

[48] David Boetius and Stefan Leue. Counterexample-guided repair of reinforcement learning systems using safety critics. *arXiv preprint arXiv:2405.15430*, 2024.

[49] Briti Gangopadhyay, Pallab Dasgupta, and Soumyajit Dey. Counterexample-guided policy refinement in multi-agent reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 1606–1614, 2023.

[50] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Chen Wang, Hubertus Franke, Zbigniew T Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. Flash: Fast model adaptation in ml-centric cloud platforms. *Proceedings of Machine Learning and Systems*, 6:524–544, 2024.

[51] Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 315–335. Springer, 2025.

[52] Davor Runje and Sharath M Shankaranarayana. Constrained monotonic neural networks. In *International Conference on Machine Learning*, pages 29338–29353. PMLR, 2023.

[53] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *Advances in neural information processing systems*, 32, 2019.

[54] Dennis Wei, Haoze Wu, Min Wu, Pin-Yu Chen, Clark Barrett, and Eitan Farchi. Convex bounds on the softmax function with applications to robustness verification. In *International Conference on Artificial Intelligence and Statistics*, pages 6853–6878. PMLR, 2023.

[55] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[56] Tomer Gilad, Nathan H Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. Robustifying network protocols with adversarial examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 85–92, 2019.

[57] Parsa Pazhooheshy, Soheil Abbasloo, and Yashar Ganjali. Harnessing ml for network protocol assessment: A congestion control use case. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 213–219, 2023.

[58] Arnav Chakravarthy, Nina Narodytska, Asmitha Rathis, Marius Vilcu, Mahmood Sharif, and Gagandeep Singh. 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.

[59] Arnaud Dethise, Marco Canini, and Srikanth Kandula. 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*, pages 29–36, 2019.

[60] Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. Deep reinforcement learning with robust and smooth policy. In *International Conference on Machine Learning*, pages 8707–8718. PMLR, 2020.

[61] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[62] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. Toward formally verifying congestion control behavior. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 1–16, 2021.

[63] Anup Agarwal, Venkat Arun, Devdeep Ray, Ruben Martins, and Srinivasan Seshan. Towards provably performant congestion control. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 951–978, 2024.

[64] Mina Tahmasbi Arashloo, Ryan Beckett, and Rachit Agarwal. Formal methods for network performance analysis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 645–661, 2023.

[65] Bingzhe Liu, Gangmuk Lim, Ryan Beckett, and P. Brighten Godfrey. Kivi: Verification for cluster management. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 509–527, Santa Clara, CA, July 2024. USENIX Association.

[66] Xudong Sun, Wenjie Ma, Jiawei Tyler Gu, Zicheng Ma, Tej Chajed, Jon Howell, Andrea Lattuada, Oded Padon, Lalith Suresh, Adriana Szekeres, and Tianyin Xu. Anvil: Verifying liveness of cluster management controllers. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 649–666, Santa Clara, CA, July

2024. USENIX Association.

[67] Haoze Wu, Clark Barrett, Mahmood Sharif, Nina Narodytska, and Gagandeep Singh. Scalable verification of gnn-based job schedulers. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):1036–1065, 2022.

A Probability Calculation Details for Aurora Case Study

Let $Y_1 \sim \mathcal{N}(L_0^1, \sigma^2)$ and $Y_2 \sim \mathcal{N}(L_0^2, \sigma^2)$ represent the selected action of each policy. Consider the constraints $(L_0^1 \geq \mu)$ and $(L_0^2 \leq -\mu)$ derived from the first property for Aurora in §8. Under these constraints, we can calculate the probability of getting a positive action y_1 from the first policy copy and a negative action y_2 from the second policy copy in the following way.

The "hardest" case under those constraints is at the boundary $(L_0^1 = \mu,\ L_0^2 = -\mu)$. Then, $\Pr(Y_1 > 0) = \Phi\left(\frac{\mu}{\sigma}\right)$, and $\Pr(Y_2 < 0) = \Phi\left(\frac{\mu}{\sigma}\right)$, where $\Phi(\cdot)$ is the standard normal Cumulative Distribution Function (CDF). For both events to hold jointly, assuming independence, $\Pr(Y_1 > 0,\ Y_2 < 0) = \Pr(Y_1 > 0)\Pr(Y_2 < 0) = \Phi\left(\frac{\mu}{\sigma}\right)^2$.

On the other hand, if we want to make sure with a given probability Q, we get a positive action y_1 from the first policy copy and a negative action y_2 from the second policy copy by putting the constraints $(L_0^1 \geq \mu)$ and $(L_0^2 \leq -\mu)$, we should select $\mu = \sigma\,\Phi^{-1}(\sqrt{Q})$.