Towards Scalable Verification of Deep Reinforcement Learning

October 2021

Guy Amir    Michael Schapira    Guy Katz
Traditionally

Computer and networked systems are *handcrafted* by domain-specific experts
An Emerging Alternative

Deep Reinforcement Learning (DRL) solutions outperform the state-of-the-art in various contexts.

| System        | Application Domain                  |
|---------------|-------------------------------------|
| Aurora [29]   | congestion control                  |
| NeuroCuts [40]| packet classification              |
| [51]          | SQL optimization                    |
| NEO [49]      | SQL optimization                    |
| DeepRM [44]   | resource allocation                 |
| [72]          | resource allocation                 |
| [42]          | resource & power management         |
| [36]          | compiler phase ordering             |
| [52]          | device placement                    |
| Placeto [2]   | device placement                    |
| Decima [48]   | spark cluster job scheduling        |
| Pensieve [46] | adaptive video streaming            |
| AuTO [11]     | traffic optimizations               |
Reinforcement Learning (RL)
Reinforcement Learning (RL)

\[ \max(E[\sum_t \gamma^t r_t]) \]
Reinforcement Learning (RL)

- Infinite Runs
- Complex Policies
- Communication Domain
But...

How do we know a Deep Neural Network trained via Reinforcement Learning is safe?

“Testing shows the presence, not the absence of bugs”
Dijkstra, 1969

Challenge: These “black boxes” need to be formally verified for correct behavior
Our approach: **Formal Verification**!

*Provably guarantee that a learned policy meets our requirements, or identify concrete violations (bugs)*
Example: The **Aurora** Congestion Controller

[Jay, Rotman, et al., ICML 2019]

\[
\text{latency gradient}_{t} 
\geq 1 \\
\text{latency ratio}_{t} 
\geq 1 \\
\text{sending ratio}_{t} 
\geq 1 \\
\text{timestep } t
\]

- Increase sending rate
- Maintain sending rate
- Decrease sending rate
**Aurora Safety Properties**

*Safety* - “*Something bad never happens*”
(finite-long violations)

| Network Conditions | Wanted Output          |
|--------------------|------------------------|
| poor               | next-step decrease     |
| excellent          | next-step increase     |
Aurora Liveness Properties

Liveness - “Something good eventually happens”
(infinite-long violations)

| Network Conditions | Wanted Output          |
|--------------------|------------------------|
| poor               | eventual decrease      |
| excellent          | eventual increase      |
Our Verification Strategy

Defining a **state graph** & **transition function**

[Eliyahu-Kazak-Katz-Schapira, SIGCOMM 2021]
Transition System Graph

Defining a state_t

Defining a transition_{t,t'}

state_t = input_t + output_t

state_t = state_{t,t'}
Encoding Multiple Transitions
Our Verification Strategy

Defining a state graph & transition function
[Eliahu-Kazak-Katz-Schapira, SIGCOMM 2021]

Running a portfolio approach for checking k-long violations or k-long provable runs
[Amir-Schapira-Katz, FMCAD 2021]
Bounded Model Checking (BMC)

**Bounded Model Checking**

A method for checking violations of properties, for a given number of \( k \) steps
Bounded Model Checking (BMC)

Bounded Model Checking
A method for checking violations of properties, for a given number of \( k \) steps
Bounded Model Checking (BMC)

Bounded Model Checking
A method for checking violations of properties, for a given number of $k$ steps
BMC Setbacks

We can’t prove that any properties hold

We can’t analyze complex properties
BMC

Initial State

$k = 1$ step

$k = 2$ steps

$k = 3$ steps

Bad State
K-Induction

(k + 2) steps

Initial State

(k + 1) steps

k steps

Bad State
Our Verification Strategy

Defining a state graph & transition function
[Ellyahu-Kazak-Katz-Schapira, SIGCOMM 2021]

Running a portfolio approach for checking k-long violations or k-long provable runs
[Amir-Schapira-Katz, FMCAD 2021]
Portfolio Approach

input: + property

initialization: k=1 step

emulate k steps

increment: k ++

K-Induction

BMC

exit

correctness

bug
## Aurora Properties

| Property | Network Conditions | Desired Output | Property Holds? |
|----------|-------------------|----------------|-----------------|
| Property 1 | excellent 😊 | eventual change | ✓ |
| Property 2 | excellent 😊 | eventual increase | ❌ k=2 |
| Property 3 | poor 😞 | next-step decrease | ❌ k=1 |
| Property 4 | poor 😞 | eventual decrease | ✓ |
WhiRL 2.0 - Techniques

1. K-Induction
2. Invariant Inference
3. Abstraction
Invariant

A partition of the state space $S$ into two disjoint sets $S_1$ and $S_2$ such that:

$$s_1 \in S_1 \land s_2 \in S_2 \rightarrow (s_1, s_2) \notin \text{trans}$$
Invariant Inference

Templates:

✓ use *monotonicity* of properties
✓ fix *inputs* or *outputs*
✓ conduct a *binary search* on the non-fixed variables
✓ *dynamic*: user-chosen values

Strategy: search for the “2nd best” behavior
Execution of *invariance inference algorithms* based on the *templates*

For example, an invariant is found, *based on the following violated property*

| Network Conditions | Wanted Output | Property Holds? |
|--------------------|---------------|-----------------|
| Property 3         | poor          | next-step decrease | k=1 |
Invariant Inference - Aurora

Originally, “poor” network conditions:

\[ 2 \leq \text{sending\_ratio}_t \]

We can search for the \textit{worst-case sending ratio} for the output to decrease:

\[ \text{output}_t < 0 \]
Invariant Inference - Aurora

Initialization: \( 0 \leq \text{output}, \ 2 \leq sending\_ratio_t \leq M_{\text{user}} \)

Iterate:

- Binary-search the \( sending\_ratio_t \) lower bound
- Call a \textit{verifier} on the middle point
- Update \( sending\_ratio_t \) accordingly

Return: lower bound on worst case \( sending\_ratio_t \)
Invariant Inference - Aurora

\[ \text{verifier}\{\text{sending}_{ratio_t} \in [2, M]\} \rightarrow \text{SAT} \]

\[ \frac{1}{2} (2+M) \]

\[ \frac{1}{2} (2+M), M \]
**Invariant Inference - Aurora**

\[ \text{sending}_{ratio_t} \text{ lower bound for SAT} \rightarrow 2 \]

\[ \text{sending}_{ratio_t} \text{ lower bound for UNSAT} \rightarrow M \]

\[ \frac{1}{2} (M+2) \quad \ldots \quad M \]

**verifier** \( \{ \text{sending}_{ratio_t} \in \left[ \frac{1}{2} \left( \frac{1}{2} (M + 2) + M \right), M \right] \} \rightarrow \text{UNSAT} \)
Invariant Inference - Aurora

\[ \text{sending\_ratio}_t \text{ lower bound for SAT} \rightarrow \frac{1}{2}(M+2) \rightarrow \frac{1}{2}(M+2) \rightarrow \cdots \rightarrow M \]

\[ \text{sending\_ratio}_t \text{ lower bound for UNSAT} \rightarrow M \]
Invariant Inference - Aurora

after \( \log(M) \) iterations:

\[ \cdots \cdots \]

\[ M \]

\[ \text{return: } \mathres = \text{lower bound on worst-case } sending\_ratio_t \]
Techniques

1. K-Induction

2. Invariant Inference

3. Abstraction
See paper for...

- Abstraction techniques for generalization
- Methods for identifying undesirable policies
- Modules for improving interpretability

[Amir-Schapira-Katz, FMCAD 2021]
Summary

A (first?) method for proving properties of RL-driven systems

Automatic invariant inference of “2nd best” properties, in chosen scenarios

Explainability and interpretability of bad policies
Future Steps

- Improve scalability
- Focus on generalization
Questions