Learning to Check LTL Satisfiability and to Generate Traces via Differentiable Trace Checking

Weilin Luo[1], Pingjia Liang [1], Junming Qiu [1], Polong Chen [1], Hai Wan [1*], Jianfeng Du [2,3*], Weiyuan Fang [1]

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2] Guangdong University of Foreign Studies, Guangzhou, China
[3] Bigmath Technology, Shenzhen, China

Content

Content

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $p_1 \, \mathcal{U} \bigcirc p_2$, output: SAT
- important applications, *e.g.*, model checking[6], goal-conflict analysis[5,18], and business process[21]
- PSPACE-complete[28]

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $p_1 \, \mathcal{U} \, \bigcirc p_2$, output: SAT
- important applications, *e.g.*, model checking [6], goal-conflict analysis [5,18], and business process [21]
- PSPACE-complete [28]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking [23,24], tableau [1,11,27,29], temporal resolution [8,26], anti-chain [30], and Boolean satisfiability (SAT) problem [12–17]
- sound and complete

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $p_1 \,\mathcal{U}\, \bigcirc p_2$, output: SAT
- important applications, *e.g.*, model checking[6], goal-conflict analysis[5,18], and business process[21]
- PSPACE-complete[28]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking[23,24], tableau[1,11,27,29], temporal resolution[8,26], anti-chain[30], and Boolean satisfiability (SAT) problem[12–17]
- sound and complete
- *heavily relies on well-design search heuristics*

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $p_1 \, \mathcal{U} \, \bigcirc p_2$, output: SAT
- important applications, *e.g.*, model checking[6], goal-conflict analysis[5,18], and business process[21]
- PSPACE-complete[28]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking[23,24], tableau[1,11,27,29], temporal resolution[8,26], anti-chain[30], and Boolean satisfiability (SAT) problem[12–17]
- sound and complete
- *heavily relies on well-design search heuristics*

Learning-based approaches

- promising approximators for checking LTL satisfiability *in polynomial time*[19]

Motivation

Linear temporal logic (LTL) satisfiability checking

- *e.g.*, input: $p_1 \, \mathcal{U} \, \bigcirc p_2$, output: SAT
- important applications, *e.g.*, model checking[6], goal-conflict analysis[5,18], and business process[21]
- PSPACE-complete[28]

Related work

- *logical approaches*: *e.g.*, based on logical reasoning mechanisms, such as model checking[23,24], tableau[1,11,27,29], temporal resolution[8,26], anti-chain[30], and Boolean satisfiability (SAT) problem[12–17]
- sound and complete
- *heavily relies on well-design search heuristics*

Learning-based approaches

- promising approximators for checking LTL satisfiability *in polynomial time*[19]
- Is it enough to answer the SAT or UNSAT enough?

Motivation

LTL satisfiability checking and trace generation (*SAT-and-GET*)

- *e.g.*, input: $p_1 \,\mathcal{U}\, \bigcirc p_2$, output: SAT because $(\{p_1, p_2\}, \{p_1\})^\omega \models p_1 \,\mathcal{U}\, \bigcirc p_2$

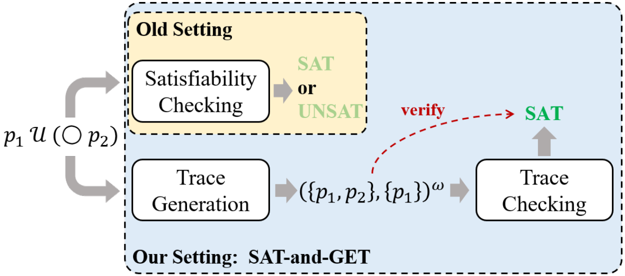


Figure 1: Overview of SAT-and-GET

Motivation

LTL satisfiability checking and trace generation (*SAT-and-GET*)

- *e.g.*, input: $p_1 \ \mathcal{U} \ \bigcirc p_2$, output: SAT because $(\{p_1, p_2\}, \{p_1\})^\omega \models p_1 \ \mathcal{U} \ \bigcirc p_2$

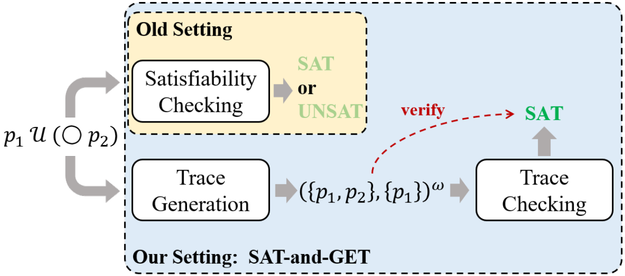


Figure 1: Overview of SAT-and-GET

Challenges

- bridging the gap between the *continuous domain* and the *discrete domain*

Motivation

LTL satisfiability checking and trace generation (*SAT-and-GET*)

- *e.g.*, input: $p_1 \, \mathcal{U} \bigcirc p_2$, output: SAT because $(\{p_1, p_2\}, \{p_1\})^\omega \models p_1 \, \mathcal{U} \bigcirc p_2$

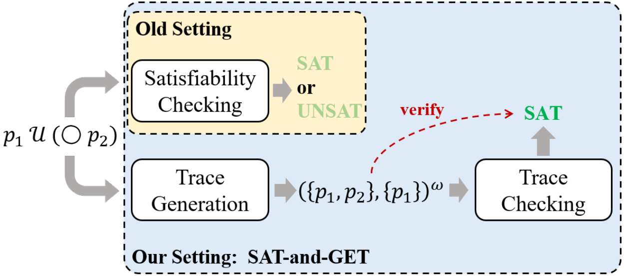


Figure 1: Overview of SAT-and-GET

Challenges

- bridging the gap between the *continuous domain* and the *discrete domain*
- *supervision confusion* from multiple satisfiable traces

Motivation

Deep reinforcement learning-based approach (OSUG)[20]

- relying on feedback from the environment implemented

Motivation

Deep reinforcement learning-based approach (OSUG)[20]

- relying on feedback from the environment implemented
- *sample inefficient and unstable training*
- *slight* performance improvements beyond random guessing

Motivation

Deep reinforcement learning-based approach (OSUG)[20]

- relying on feedback from the environment implemented
- *sample inefficient and unstable training*
- *slight* performance improvements beyond random guessing

> There is still a lack of study on *tractable* and *more effective* approaches to the SAT-and-GET problem.

Motivation

Deep reinforcement learning-based approach (OSUG)[20]

- relying on feedback from the environment implemented
- *sample inefficient and unstable training*
- *slight* performance improvements beyond random guessing

There is still a lack of study on *tractable* and *more effective* approaches to the SAT-and-GET problem.

Our contributions

- an *LTL encoding method* to parameterize a neural network and theoretically prove that its inference process (*neural trace checking*) is able to simulate LTL trace checking

Motivation

Deep reinforcement learning-based approach (OSUG)[20]

- relying on feedback from the environment implemented
- *sample inefficient and unstable training*
- *slight* performance improvements beyond random guessing

There is still a lack of study on *tractable* and *more effective* approaches to the SAT-and-GET problem.

Our contributions

- an *LTL encoding method* to parameterize a neural network and theoretically prove that its inference process (*neural trace checking*) is able to simulate LTL trace checking
- a novel approach *jointly* trains a neural network for LTL satisfiability checking and a neural network for trace generation guided by neural trace checking

Content

LTL Encoding

Definition 1 (LTL Encoding)

Let $\mathbb{P}$ be a set of atom propositions, ϕ an LTL formula over $\mathbb{P}$, $(V, E, v_1, \mathtt{lab}_V, \mathtt{lab}_E) = \mathtt{tree}(\phi)$, and $\langle v_1, \ldots, v_{|\phi|} \rangle = \mathtt{pretravel}(\mathtt{tree}(\phi))$. The *LTL encoding* η of ϕ is a 6-tuple

$$(\boldsymbol{\eta}_{right}, \boldsymbol{\eta}_{atom}, \boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}}),$$

where $\boldsymbol{\eta}_{right} \in \{0,1\}^{|\phi| \times |\phi|}$, $\boldsymbol{\eta}_{atom} \in \{0,1\}^{|\phi| \times |\mathbb{P}|}$, and $\boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}} \in \{0,1\}^{|\phi|}$; moreover, for all subscripts i and j, $(\boldsymbol{\eta}_{right})_{i,j} = 1$ if $\mathtt{lab}_E((v_i, v_j)) = R$ and $(\boldsymbol{\eta}_{right})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{atom})_{i,j} = 1$ if $\mathtt{lab}_V(v_i) = p_j$ and $(\boldsymbol{\eta}_{atom})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{\circ})_i = 1$ if $\mathtt{lab}_V(v_i) = \circ$ and $(\boldsymbol{\eta}_{\circ})_i = 0$ otherwise, where $\circ \in \{\neg, \wedge, \bigcirc, \mathcal{U}\}$.

LTL Encoding

Definition 1 (LTL Encoding)

Let $\mathbb{P}$ be a set of atom propositions, ϕ an LTL formula over $\mathbb{P}$, $(V, E, v_1, \mathtt{lab}_V, \mathtt{lab}_E) = \mathtt{tree}(\phi)$, and $\langle v_1, \ldots, v_{|\phi|} \rangle = \mathtt{pretravel}(\mathtt{tree}(\phi))$. The *LTL encoding* η of ϕ is a 6-tuple

$$(\boldsymbol{\eta}_{right}, \boldsymbol{\eta}_{atom}, \boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}}),$$

where $\boldsymbol{\eta}_{right} \in \{0,1\}^{|\phi| \times |\phi|}$, $\boldsymbol{\eta}_{atom} \in \{0,1\}^{|\phi| \times |\mathbb{P}|}$, and $\boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}} \in \{0,1\}^{|\phi|}$; moreover, for all subscripts i and j, $(\boldsymbol{\eta}_{right})_{i,j} = 1$ if $\mathtt{lab}_E((v_i, v_j)) = R$ and $(\boldsymbol{\eta}_{right})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{atom})_{i,j} = 1$ if $\mathtt{lab}_V(v_i) = p_j$ and $(\boldsymbol{\eta}_{atom})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{\circ})_i = 1$ if $\mathtt{lab}_V(v_i) = \circ$ and $(\boldsymbol{\eta}_{\circ})_i = 0$ otherwise, where $\circ \in \{\neg, \wedge, \bigcirc, \mathcal{U}\}$.

An LTL encoding characterizes the syntax tree of an LTL formula:

- $(\boldsymbol{\eta}_{right})_{i,j}$: whether the right sub-formula of ϕ_i is ϕ_j;

LTL Encoding

Definition 1 (LTL Encoding)

Let $\mathbb{P}$ be a set of atom propositions, ϕ an LTL formula over $\mathbb{P}$, $(V, E, v_1, \mathtt{lab}_V, \mathtt{lab}_E) = \mathtt{tree}(\phi)$, and $\langle v_1, \ldots, v_{|\phi|}\rangle = \mathtt{pretravel}(\mathtt{tree}(\phi))$. The *LTL encoding* η of ϕ is a 6-tuple

$$(\boldsymbol{\eta}_{right}, \boldsymbol{\eta}_{atom}, \boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}}),$$

where $\boldsymbol{\eta}_{right} \in \{0,1\}^{|\phi|\times|\phi|}$, $\boldsymbol{\eta}_{atom} \in \{0,1\}^{|\phi|\times|\mathbb{P}|}$, and $\boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}} \in \{0,1\}^{|\phi|}$; moreover, for all subscripts i and j, $(\boldsymbol{\eta}_{right})_{i,j} = 1$ if $\mathtt{lab}_E((v_i, v_j)) = R$ and $(\boldsymbol{\eta}_{right})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{atom})_{i,j} = 1$ if $\mathtt{lab}_V(v_i) = p_j$ and $(\boldsymbol{\eta}_{atom})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{\circ})_i = 1$ if $\mathtt{lab}_V(v_i) = \circ$ and $(\boldsymbol{\eta}_{\circ})_i = 0$ otherwise, where $\circ \in \{\neg, \wedge, \bigcirc, \mathcal{U}\}$.

An LTL encoding characterizes the syntax tree of an LTL formula:

- $(\boldsymbol{\eta}_{right})_{i,j}$: whether the right sub-formula of ϕ_i is ϕ_j;
- $(\boldsymbol{\eta}_{atom})_{i,j}$: whether ϕ_i is the atomic proposition p_j;

LTL Encoding

Definition 1 (LTL Encoding)

Let $\mathbb{P}$ be a set of atom propositions, ϕ an LTL formula over $\mathbb{P}$, $(V, E, v_1, \mathtt{lab}_V, \mathtt{lab}_E) = \mathtt{tree}(\phi)$, and $\langle v_1, \ldots, v_{|\phi|} \rangle = \mathtt{pretravel}(\mathtt{tree}(\phi))$. The *LTL encoding* η of ϕ is a 6-tuple

$$\left(\boldsymbol{\eta}_{right}, \boldsymbol{\eta}_{atom}, \boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}}\right),$$

where $\boldsymbol{\eta}_{right} \in \{0,1\}^{|\phi| \times |\phi|}$, $\boldsymbol{\eta}_{atom} \in \{0,1\}^{|\phi| \times |\mathbb{P}|}$, and $\boldsymbol{\eta}_{\neg}, \boldsymbol{\eta}_{\wedge}, \boldsymbol{\eta}_{\bigcirc}, \boldsymbol{\eta}_{\mathcal{U}} \in \{0,1\}^{|\phi|}$; moreover, for all subscripts i and j, $(\boldsymbol{\eta}_{right})_{i,j} = 1$ if $\mathtt{lab}_E((v_i, v_j)) = R$ and $(\boldsymbol{\eta}_{right})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{atom})_{i,j} = 1$ if $\mathtt{lab}_V(v_i) = p_j$ and $(\boldsymbol{\eta}_{atom})_{i,j} = 0$ otherwise, $(\boldsymbol{\eta}_{\circ})_i = 1$ if $\mathtt{lab}_V(v_i) = \circ$ and $(\boldsymbol{\eta}_{\circ})_i = 0$ otherwise, where $\circ \in \{\neg, \wedge, \bigcirc, \mathcal{U}\}$.

An LTL encoding characterizes the syntax tree of an LTL formula:

- $(\boldsymbol{\eta}_{right})_{i,j}$: whether the right sub-formula of ϕ_i is ϕ_j;
- $(\boldsymbol{\eta}_{atom})_{i,j}$: whether ϕ_i is the atomic proposition p_j;
- $(\boldsymbol{\eta}_{\circ})_i$: whether ϕ_i is defined to connect its sub-formulae via operator $\circ \in \{\neg, \wedge, \bigcirc, \mathcal{U}\}$.

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \,\mathcal{U}\, \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

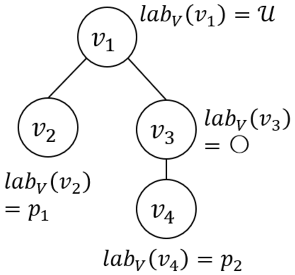


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \ \mathcal{U} \ \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\eta_{\mathcal{U}})_1 = 1$,

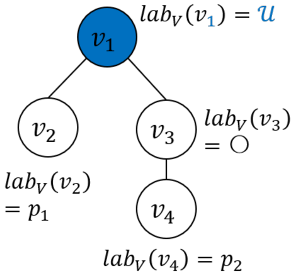


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \,\mathcal{U} \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\eta_{\mathcal{U}})_1 = 1$,
- $(\eta_{atom})_{2,1} = 1$,

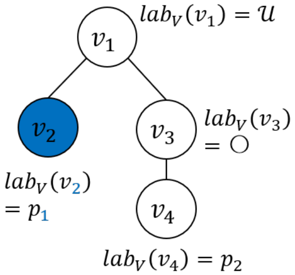


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \,\mathcal{U}\, \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\eta_{\mathcal{U}})_1 = 1$,
- $(\eta_{atom})_{2,1} = 1$,
- $(\eta_{\bigcirc})_3 = 1$,

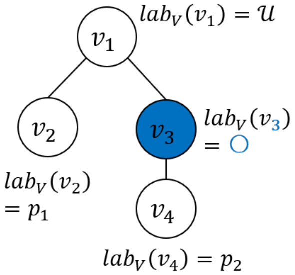


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \ \mathcal{U} \ \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\eta_{\mathcal{U}})_1 = 1$,
- $(\eta_{atom})_{2,1} = 1$,
- $(\eta_{\bigcirc})_3 = 1$,
- $(\eta_{atom})_{4,2} = 1$,

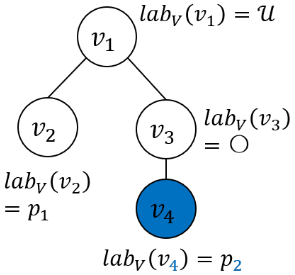


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \, \mathcal{U} \, \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\eta_{\mathcal{U}})_1 = 1$,
- $(\eta_{atom})_{2,1} = 1$,
- $(\eta_{\bigcirc})_3 = 1$,
- $(\eta_{atom})_{4,2} = 1$,
- $(\eta_{right})_{1,3} = 1$,

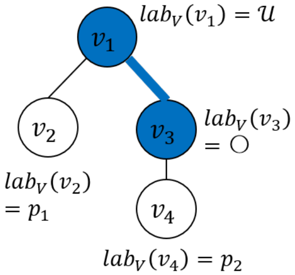


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

LTL Encoding

Example 1 (LTL Encoding)

Let $\phi = p_1 \ \mathcal{U} \ \bigcirc p_2$ be an LTL formula. $\texttt{pretravel}(\texttt{tree}(\phi)) = \langle v_1, v_2, v_3, v_4 \rangle$. The LTL encoding η of ϕ is

- $(\boldsymbol{\eta}_{\mathcal{U}})_1 = 1,$
- $(\boldsymbol{\eta}_{atom})_{2,1} = 1,$
- $(\boldsymbol{\eta}_{\bigcirc})_3 = 1,$
- $(\boldsymbol{\eta}_{atom})_{4,2} = 1,$
- $(\boldsymbol{\eta}_{right})_{1,3} = 1,$
- and all other elements of η are 0.

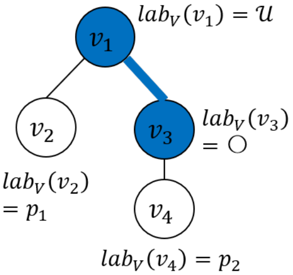


Figure 2: The syntax tree $\texttt{tree}(\phi)$ of ϕ

Neural Modeling of Trace Checking

Definition 2 (Tensorized Trace)

Let $\mathbb{P}$ be a set of atom propositions and π a trace over $\mathbb{P}$. The *tensorized trace* of π is a 2-tuple $(\boldsymbol{s}, \boldsymbol{l})$ computed by Algorithm 2, where $\boldsymbol{s} \in \mathbb{R}_{[0,1]}^{|\pi| \times |\mathbb{P}|}$ and $\boldsymbol{l} \in \mathbb{R}_{[0,1]}^{|\pi|}$, and where $\mathbb{R}_{[0,1]}$ denotes the real value range from 0 to 1.

Algorithm 2: TENSORIZE

Input: A trace π over a set of atomic propositions $\mathbb{P}$.
Output: The tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$ of π.

```
s ← 0 where s ∈ {0,1}^{|π|×|P|}, l ← 0 where l ∈ {0,1}^{|π|}
for each state π[i] of π do
    for each atomic proposition p_j ∈ P do
        if p_j ∈ π[i] then
            (s)_{i,j} ← 1
    if i is loop-start time then
        (l)_i ← 1
```

Neural Modeling of Trace Checking

Definition 2 (Tensorized Trace)

Let $\mathbb{P}$ be a set of atom propositions and π a trace over $\mathbb{P}$. The *tensorized trace* of π is a 2-tuple $(\boldsymbol{s}, \boldsymbol{l})$ computed by Algorithm 2, where $\boldsymbol{s} \in \mathbb{R}_{[0,1]}^{|\pi|\times|\mathbb{P}|}$ and $\boldsymbol{l} \in \mathbb{R}_{[0,1]}^{|\pi|}$, and where $\mathbb{R}_{[0,1]}$ denotes the real value range from 0 to 1.

Algorithm 2: TENSORIZE

Input: A trace π over a set of atomic propositions $\mathbb{P}$.
Output: The tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$ of π.

```
s ← 0 where s ∈ {0,1}^{|π|×|ℙ|}, l ← 0 where l ∈ {0,1}^{|π|}
for each state π[i] of π do
    for each atomic proposition p_j ∈ ℙ do
        if p_j ∈ π[i] then
            (s)_{i,j} ← 1
    if i is loop-start time then
        (l)_i ← 1
```

Example 2 (Tensorized Trace)

Let $\mathbb{P} = \{p_1, p_2\}$ be a set of atom propositions and $\pi = (\{p_1, p_2\}, \{p_1\})^\omega$ a trace over $\mathbb{P}$. The tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$ of π is shown as follows:

$$\boldsymbol{s} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad \boldsymbol{l} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (`NTCNet`)

- `NTCNet` is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1}, \end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}. \end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_{\neg})_j(1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_{\wedge})_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_{\bigcirc})_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_{\mathcal{U}})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1},\end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}.\end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1-(\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j(\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j(\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|}(\boldsymbol{l})_k(\boldsymbol{x}_k)_{j+1},\end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|}(\boldsymbol{\theta}_{atom})_{j,k}(\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|}(\boldsymbol{\theta}_{right})_{j,k}(\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases}(\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}),\end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|}(\boldsymbol{l})_k(\boldsymbol{u}_k)_j^{(t)}.\end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1-(\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1},\end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}.\end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned} (\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_{\neg})_j (1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_{\wedge})_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_{\bigcirc})_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_{\mathcal{U}})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1}, \end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a + b - 1)$	$\sigma(a + b)$	$1 - a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned} (\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}. \end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1},\end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}.\end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned}(\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1}, \end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned}(\boldsymbol{u}_i)_j^{(t)} = & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}. \end{aligned} \tag{4}$$

Neural Modeling of Trace Checking

Neural network parameterized by LTL encoding (NTCNet)

- NTCNet is parameterized by $\theta = \eta$.
- The satisfaction vectors $\boldsymbol{x}_i \in \mathbb{R}^{|\phi|}$ where $i \in [1, |\pi|]$ are initialized by $\mathbf{0}$.
- $(\boldsymbol{x}_i)_j$ is computed as follows:

$$\begin{aligned} (\boldsymbol{x}_i)_j = \sigma(\quad & (\boldsymbol{p}_i)_j + \\ & (\boldsymbol{\theta}_\neg)_j(1 - (\boldsymbol{x}_i)_{j+1}) + \\ & (\boldsymbol{\theta}_\wedge)_j \sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{r}_i)_j - 1) + \\ & (\boldsymbol{\theta}_\bigcirc)_j (\boldsymbol{x}_{i+1})_{j+1} + \\ & (\boldsymbol{\theta}_\mathcal{U})_j (\boldsymbol{u}_i)_j^{(|\pi|)} \quad), \\ (\boldsymbol{x}_{|\pi|+1})_{j+1} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{x}_k)_{j+1}, \end{aligned} \tag{1}$$

Table 1: Converting logical operations to soften forms.

formal logic	$a \wedge b$	$a \vee b$	$\neg a$
soften logic	$\sigma(a+b-1)$	$\sigma(a+b)$	$1-a$

$$(\boldsymbol{p}_i)_j = \sum_{k=1}^{|\mathbb{P}|} (\boldsymbol{\theta}_{atom})_{j,k} (\boldsymbol{s})_{i,k}. \tag{2}$$

$$(\boldsymbol{r}_i)_j = \sum_{k=j+2}^{|\pi|} (\boldsymbol{\theta}_{right})_{j,k} (\boldsymbol{x}_i)_k. \tag{3}$$

$$\begin{aligned} (\boldsymbol{u}_i)_j^{(t)} = \quad & \begin{cases} (\boldsymbol{r}_i)_j, \\ \sigma(\sigma((\boldsymbol{x}_i)_{j+1} + (\boldsymbol{u}_{i+1})_j^{(t-1)} - 1) + \\ (\boldsymbol{u}_i)_j^{(1)}), \end{cases} \\ (\boldsymbol{u}_{|\pi|+1})_j^{(t)} = \quad & \textstyle\sum_{k=1}^{|\pi|} (\boldsymbol{l})_k (\boldsymbol{u}_k)_j^{(t)}. \end{aligned} \tag{4}$$

- NTCNet$((\boldsymbol{s}, \boldsymbol{l})\,|\eta) = (\boldsymbol{x}_1)_1$ denotes the result of *neural trace checking* of the satisfaction relation between π and ϕ.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\texttt{NTCNet}(\textsc{Tensorize}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\texttt{NTCNet}(\textsc{Tensorize}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

- NTCNet is able to check the satisfaction relation between a possibly infinite trace π and an LTL formula ϕ.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\texttt{NTCNet}(\textsc{Tensorize}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

- NTCNet is able to check the satisfaction relation between a possibly infinite trace π and an LTL formula ϕ.
- Important step: the computation of $(\boldsymbol{u}_1)_j^{(|\pi|)}$ suffices to check whether $\pi \models \phi_j$.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\texttt{NTCNet}(\textsc{tensorize}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

- NTCNet is able to check the satisfaction relation between a possibly infinite trace π and an LTL formula ϕ.
- Important step: the computation of $(\boldsymbol{u}_1)_j^{(|\pi|)}$ suffices to check whether $\pi \models \phi_j$.
- Lemma 2 shows that $(\boldsymbol{u}_i)_j^{(t)} = 1$, if there exists a time i' where $0 \leq i' - i < t$ such that $\pi_{i'} \models \phi_k$ and $\pi_{i''} \models \phi_{j+1}$ for all $i'' \in [i, i')$.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\mathtt{NTCNet}(\text{TENSORIZE}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

- `NTCNet` is able to check the satisfaction relation between a possibly infinite trace π and an LTL formula ϕ.
- Important step: the computation of $(\boldsymbol{u}_1)_j^{(|\pi|)}$ suffices to check whether $\pi \models \phi_j$.
- Lemma 2 shows that $(\boldsymbol{u}_i)_j^{(t)} = 1$, if there exists a time i' where $0 \leq i' - i < t$ such that $\pi_{i'} \models \phi_k$ and $\pi_{i''} \models \phi_{j+1}$ for all $i'' \in [i, i')$.
- Lemma 3 shows that $\boldsymbol{u}$ is monotonically increasing with time points.

Neural Modeling of Trace Checking

Theorem 1 (Correctness of Neural Trace Checking)

Let ϕ be an LTL formula and η its LTL encoding. For any trace π, $\texttt{NTCNet}(\textsc{tensorize}(\pi)|\eta) = 1$ if and only if $\pi \models \phi$.

- NTCNet is able to check the satisfaction relation between a possibly infinite trace π and an LTL formula ϕ.
- Important step: the computation of $(\boldsymbol{u}_1)_j^{(|\pi|)}$ suffices to check whether $\pi \models \phi_j$.
- Lemma 2 shows that $(\boldsymbol{u}_i)_j^{(t)} = 1$, if there exists a time i' where $0 \leq i' - i < t$ such that $\pi_{i'} \models \phi_k$ and $\pi_{i''} \models \phi_{j+1}$ for all $i'' \in [i, i')$.
- Lemma 3 shows that $\boldsymbol{u}$ is monotonically increasing with time points.
- NTCNet is *differentiable*.

Overview of VSCNet

Train VSCNet

- SCNet extracts the formula representation ϕ and predicts the satisfiable probability p_{SC}.
- TGNet exploits ϕ to generate a tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$.
- jointly training of SCNet and TGNet with NTCNet:

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG} \qquad (5)$$

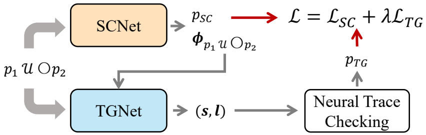


Figure 3: The overview of training VSCNet.

Overview of VSCNet

Train VSCNet

- SCNet extracts the formula representation ϕ and predicts the satisfiable probability p_{SC}.
- TGNet exploits ϕ to generate a tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$.
- jointly training of SCNet and TGNet with NTCNet:

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG} \tag{5}$$

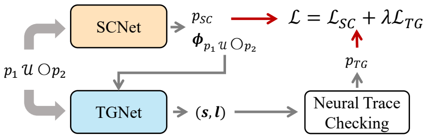


Figure 3: The overview of training VSCNet.

Apply VSCNet to solve SAT-and-GET

Algorithm 3: Solving the SAT-and-GET problem

Input: An LTL formula ϕ, and two threshold hyperparameters β_{T} for finding tensorized traces and β_{SAT} for determining the satisfiability of ϕ.

Output: Whether ϕ is satisfiable or not and a satisfiable trace π of ϕ if it is satisfiable.

$(p_{SC}, \boldsymbol{\phi}) \leftarrow \mathtt{SCNet}(\phi)$
$(\boldsymbol{s}, \boldsymbol{l}) \leftarrow \mathtt{TGNet}(\boldsymbol{\phi})$
$\pi \leftarrow \textsc{round}((\boldsymbol{s}, \boldsymbol{l}), \beta_{\mathrm{T}})$
if $\pi \models \phi$ *or* $p_{SC} > \beta_{SAT}$ **then**
  return *SAT*, π
else
  return *UNSAT*

Overview of VSCNet

Train VSCNet

- SCNet extracts the formula representation ϕ and predicts the satisfiable probability p_{SC}.
- TGNet exploits ϕ to generate a tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$.
- jointly training of SCNet and TGNet with NTCNet:

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG} \tag{5}$$

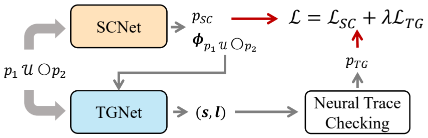


Figure 3: The overview of training VSCNet.

Apply VSCNet to solve SAT-and-GET

1. obtain the p_{SC} and a *soften* tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$

Algorithm 3: Solving the SAT-and-GET problem

Input: An LTL formula ϕ, and two threshold hyperparameters β_{T} for finding tensorized traces and β_{SAT} for determining the satisfiability of ϕ.

Output: Whether ϕ is satisfiable or not and a satisfiable trace π of ϕ if it is satisfiable.

$(p_{SC}, \boldsymbol{\phi}) \leftarrow \texttt{SCNet}(\phi)$

$(\boldsymbol{s}, \boldsymbol{l}) \leftarrow \texttt{TGNet}(\boldsymbol{\phi})$

$\pi \leftarrow \textsc{round}((\boldsymbol{s}, \boldsymbol{l}), \beta_{\mathsf{T}})$

if $\pi \models \phi$ *or* $p_{SC} > \beta_{SAT}$ **then**

  return *SAT*, π

else

  return *UNSAT*

Overview of VSCNet

Train VSCNet

- SCNet extracts the formula representation ϕ and predicts the satisfiable probability p_{SC}.
- TGNet exploits ϕ to generate a tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$.
- jointly training of SCNet and TGNet with NTCNet:

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG} \qquad (5)$$

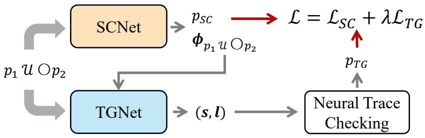


Figure 3: The overview of training VSCNet.

Apply VSCNet to solve SAT-and-GET

1. obtain the p_{SC} and a *soften* tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$
2. round probabilistic $(\boldsymbol{s}, \boldsymbol{l})$ to π

Algorithm 3: Solving the SAT-and-GET problem

Input: An LTL formula ϕ, and two threshold hyperparameters $\beta_{\top}$ for finding tensorized traces and β_{SAT} for determining the satisfiability of ϕ.

Output: Whether ϕ is satisfiable or not and a satisfiable trace π of ϕ if it is satisfiable.

```
(p_SC, φ) ← SCNet(φ)
(s, l) ← TGNet(φ)
π ← ROUND((s, l), β_⊤)
if π ⊨ φ or p_SC > β_SAT then
    return SAT, π
else
    return UNSAT
```

```
π[i] ← ∅ for each (s)_i
for each (s)_{i,j} ∈ s do
    if (s)_{i,j} > β_⊤ then
        π[i] ← π[i] ∪ {p_j}
k ← arg max_i((l)_i)
π ← π[1], ..., (π[k], ..., π[m])^ω
```

Overview of VSCNet

Train VSCNet

- SCNet extracts the formula representation ϕ and predicts the satisfiable probability p_{SC}.
- TGNet exploits ϕ to generate a tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$.
- jointly training of SCNet and TGNet with NTCNet:

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG} \qquad (5)$$

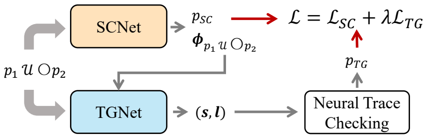


Figure 3: The overview of training VSCNet.

Apply VSCNet to solve SAT-and-GET

1. obtain the p_{SC} and a *soften* tensorized trace $(\boldsymbol{s}, \boldsymbol{l})$
2. round probabilistic $(\boldsymbol{s}, \boldsymbol{l})$ to π
3. check whether $\pi \models \phi$ by a classical trace checking algorithm [22]

Algorithm 3: Solving the SAT-and-GET problem

Input: An LTL formula ϕ, and two threshold hyperparameters β_{T} for finding tensorized traces and β_{SAT} for determining the satisfiability of ϕ.

Output: Whether ϕ is satisfiable or not and a satisfiable trace π of ϕ if it is satisfiable.

$(p_{SC}, \boldsymbol{\phi}) \leftarrow \texttt{SCNet}(\phi)$

$(\boldsymbol{s}, \boldsymbol{l}) \leftarrow \texttt{TGNet}(\boldsymbol{\phi})$

$\pi \leftarrow \textsc{Round}((\boldsymbol{s}, \boldsymbol{l}), \beta_{\mathsf{T}})$

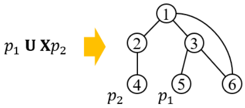
if $\pi \models \phi$ *or* $p_{SC} > \beta_{SAT}$ **then**

  return *SAT*, π

else

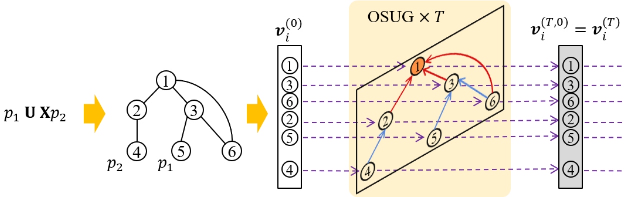
  return *UNSAT*

SCNet and TGNet



- construct the one-step unfolded graph (OSUG)[20]

SCNet and TGNet

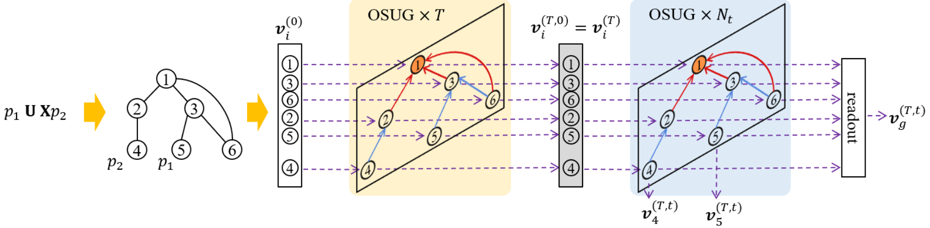


$$v_g^{(T)} = \text{MEAN}(v_1^{(T)}, \dots, v_6^{(T)})$$

$$p_{SC} = \text{MLP}(v_g^{(T)})$$

- construct the one-step unfolded graph (OSUG) [20]
- compute the graph embedding $v_g^{(T)} \in \mathbb{R}^{d_h}$ and vertex embeddings $v_i^{(T)} \in \mathbb{R}^{d_h}$ using a 10-layer OSUG-based extractor

SCNet and TGNet



$$\boldsymbol{v}_g^{(T)} = \mathrm{MEAN}(\boldsymbol{v}_1^{(T)}, \dots, \boldsymbol{v}_6^{(T)})$$

$$p_{SC} = \mathrm{MLP}(\boldsymbol{v}_g^{(T)})$$

$$\boldsymbol{l} = \mathrm{softmax}([x_1, \dots, x_{N_t}]),$$

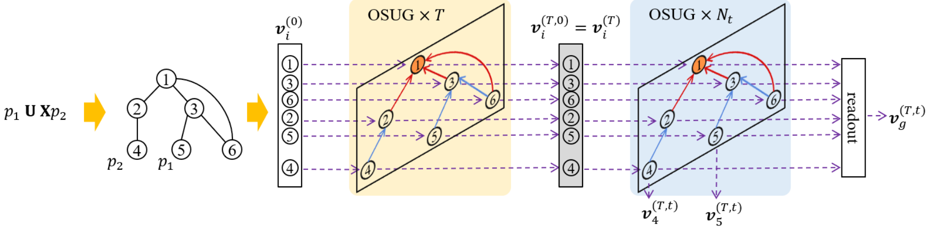
$$\text{where } x_i = \mathrm{MLP}(\boldsymbol{v}_g^{(T,i)}) \text{ for } i \in [1, N_t]$$

$$(\boldsymbol{s})_{i,j} = \left(\mathrm{softmax}\left(\mathrm{MLP}(\boldsymbol{v}_j^{(T,i)})\right)\right)_2$$

$$p_{TG} = \mathrm{NTCNet}((\boldsymbol{s}, \boldsymbol{l}), \eta)_2$$

- construct the one-step unfolded graph (OSUG) [20]
- compute the graph embedding $\boldsymbol{v}_g^{(T)} \in \mathbb{R}^{d_h}$ and vertex embeddings $\boldsymbol{v}_i^{(T)} \in \mathbb{R}^{d_h}$ using a 10-layer OSUG-based extractor
- iteratively generate a trace state by state

SCNet and TGNet



$$v_g^{(T)} = \mathrm{MEAN}(v_1^{(T)}, \dots, v_6^{(T)})$$

$$p_{SC} = \mathrm{MLP}(v_g^{(T)})$$

$$\mathcal{L} = \mathcal{L}_{SC} + \lambda \mathcal{L}_{TG}$$

$$\boldsymbol{l} = \mathrm{softmax}([x_1, \dots, x_{N_t}]), \quad \text{where } x_i = \mathrm{MLP}(v_g^{(T,i)}) \text{ for } i \in [1, N_t]$$

$$(\boldsymbol{s})_{i,j} = \left(\mathrm{softmax}\left(\mathrm{MLP}(v_j^{(T,i)})\right)\right)_2$$

$$p_{TG} = \mathrm{NTCNet}((\boldsymbol{s}, \boldsymbol{l}), \eta)_2$$

- construct the one-step unfolded graph (OSUG) [20]
- compute the graph embedding $v_g^{(T)} \in \mathbb{R}^{d_h}$ and vertex embeddings $v_i^{(T)} \in \mathbb{R}^{d_h}$ using a 10-layer OSUG-based extractor
- iteratively generate a trace state by state

Highlights of `VSCNet`

`VSCNet` ensures the soundness in $1/2$ of cases.

- Given an LTL formula ϕ, `SCNet` returns a satisfiability conclusion and `TGNet` returns a trace π.
- $\pi \models \phi$ and `SCNet` answers that ϕ is *satisfiable*: π confirms the answer of `SCNet`.
- $\pi \models \phi$ and `SCNet` answers that ϕ is *unsatisfiable*: π is able to correct the answer of `SCNet`.

Highlights of `VSCNet`

`VSCNet` ensures the soundness in $1/2$ of cases.

- Given an LTL formula ϕ, `SCNet` returns a satisfiability conclusion and `TGNet` returns a trace π.
- $\pi \models \phi$ and `SCNet` answers that ϕ is *satisfiable*: π confirms the answer of `SCNet`.
- $\pi \models \phi$ and `SCNet` answers that ϕ is *unsatisfiable*: π is able to correct the answer of `SCNet`.

`VSCNet` are able to check the satisfiability of an LTL formula in polynomial time.

- The inference time of both `SCNet` and `TGNet` is polynomial in terms of the size of the input formula.
- Although the inference time does not include the training time, model training can be done offline and once.

Content

Competitors and Datasets

Competitors: a random approach, two logic-based approaches, and seven neural network-based approaches

Table 2: Overview of Competitors

Approach	SC	TG
random	✓	✓
Aalta [14,17]	✓	✓
nuXmv [3]	✓	✓
TreeNN-inv [19]	✓	×
TreeNN-MP [19]	✓	×
TreeNN-con [19]	✓	×
Transformer-SC [19]	✓	×
Transformer-TG [10]	×	✓
Transformer	✓	✓
OSUG [20]	✓	✓
VSCNet-T (our)	✓	✓
VSCNet-G (our)	✓	✓

Competitors and Datasets

Competitors: a random approach, two logic-based approaches, and seven neural network-based approaches
Datasets: *SPOT* and large-scale datasets [17,19]

- *LTL-as-LTL$_f$* [25]: some coming from industrial scenes
- *LTL$_f$-Specific* [4,12]: generated by common patterns
- *NASA-Boeing* [2,9]: real-world specifications
- *DECLARE* [7]: widely used in the business process management

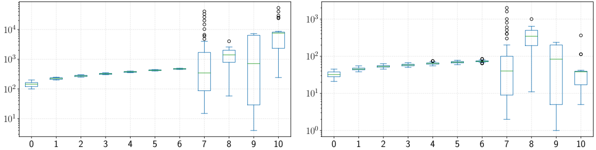


Figure 4: The boxplots of the formula size and the number of atomic propositions

Table 2: Overview of Competitors

Approach	SC	TG
random	✓	✓
`Aalta` [14,17]	✓	✓
`nuXmv` [3]	✓	✓
`TreeNN-inv` [19]	✓	×
`TreeNN-MP` [19]	✓	×
`TreeNN-con` [19]	✓	×
`Transformer-SC` [19]	✓	×
`Transformer-TG` [10]	×	✓
`Transformer`	✓	✓
`OSUG` [20]	✓	✓
`VSCNet-T` (our)	✓	✓
`VSCNet-G` (our)	✓	✓

Setups

Training and testing

- train all neural networks on the training set of *SPOT*-$[100, 200)$ with hyperparameters optimized on the validation set of *SPOT*-$[100, 200)$
- *directly* evaluate all neural networks on the test set of *SPOT*-$[100, 200)$ to demonstrate their performance on in-distribution datasets
- *directly* evaluate all neural networks on the *SPOT* test sets with larger formulae and on the large-scale datasets to test the generalizability of neural networks on out-of-distribution datasets

Setups

Training and testing

- train all neural networks on the training set of *SPOT*-$[100, 200)$ with hyperparameters optimized on the validation set of *SPOT*-$[100, 200)$
- *directly* evaluate all neural networks on the test set of *SPOT*-$[100, 200)$ to demonstrate their performance on in-distribution datasets
- *directly* evaluate all neural networks on the *SPOT* test sets with larger formulae and on the large-scale datasets to test the generalizability of neural networks on out-of-distribution datasets

Metrics

- LTL satisfiability checking: accuracy (acc.), precision (pre.), recall (rec.), and F1 score
- semantic accuracy (sacc.)[10], where D is a set of satisfiable formulae and π is the trace generated by approaches:

$$\text{sacc.} = \frac{|\{\phi \in D | \pi \models \phi\}|}{|D|}, \tag{6}$$

Comparison on Synthetic Datasets

Table 3: Evaluation results on the test set of *SPOT*-$[100, 200)$

approach	acc.	pre.	rec.	F1	sacc.	time
random	50.01	50.01	49.28	49.64	51.88	0
Aalta	100.00	100.00	100.00	100.00	100.00	12,784,302
nuXmv	100.00	100.00	100.00	100.00	100.00	6,848
TreeNN-inv	93.27	97.75	88.58	92.94	-	434
TreeNN-MP	86.54	90.74	81.40	85.82	-	455
TreeNN-con	89.38	95.96	82.22	88.56	-	459
Transformer-SC	72.56	74.24	69.10	71.58	-	**104**
Transformer-TG	-	-	-	-	47.67	5,484
Transformer	59.30	62.79	45.64	52.86	45.09	10,601
OSUG	98.47	99.15	97.79	98.46	55.22	2,816
VSCNet-T (our)	93.45	97.47	89.22	93.16	71.86	364
VSCNet-G (our)	**99.15**	**99.47**	**98.83**	**99.15**	**91.01**	**156**

- On in-distribution datasets, compared with neural network-based approaches, VSCNet achieves SOTA performance on SAT-and-GET.

Comparison on Synthetic Datasets

Table 3: Evaluation results on the test set of *SPOT*-$[100, 200)$

approach	acc.	pre.	rec.	F1	sacc.	time
random	50.01	50.01	49.28	49.64	51.88	0
Aalta	100.00	100.00	100.00	100.00	100.00	12,784,302
nuXmv	100.00	100.00	100.00	100.00	100.00	6,848
TreeNN-inv	93.27	97.75	88.58	92.94	-	434
TreeNN-MP	86.54	90.74	81.40	85.82	-	455
TreeNN-con	89.38	95.96	82.22	88.56	-	459
Transformer-SC	72.56	74.24	69.10	71.58	-	**104**
Transformer-TG	-	-	-	-	47.67	5,484
Transformer	59.30	62.79	45.64	52.86	45.09	10,601
OSUG	98.47	99.15	97.79	98.46	55.22	2,816
VSCNet-T (our)	93.45	97.47	89.22	93.16	71.86	364
VSCNet-G (our)	**99.15**	**99.47**	**98.83**	**99.15**	**91.01**	**156**

- On in-distribution datasets, compared with neural network-based approaches, VSCNet achieves SOTA performance on SAT-and-GET.
- Good generalizability of VSCNet against different formula sizes.

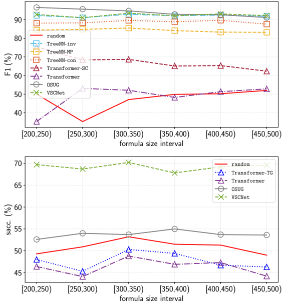


Figure 5: Evaluation results on other *SPOT* test sets.

Comparison on Synthetic Datasets

Table 3: Evaluation results on the test set of *SPOT*-$[100, 200)$

approach	acc.	pre.	rec.	F1	sacc.	time
random	50.01	50.01	49.28	49.64	51.88	0
Aalta	100.00	100.00	100.00	100.00	100.00	12,784,302
nuXmv	100.00	100.00	100.00	100.00	100.00	6,848
TreeNN-inv	93.27	97.75	88.58	92.94	-	434
TreeNN-MP	86.54	90.74	81.40	85.82	-	455
TreeNN-con	89.38	95.96	82.22	88.56	-	459
Transformer-SC	72.56	74.24	69.10	71.58	-	**104**
Transformer-TG	-	-	-	-	47.67	5,484
Transformer	59.30	62.79	45.64	52.86	45.09	10,601
OSUG	98.47	99.15	97.79	98.46	55.22	2,816
VSCNet-T (our)	93.45	97.47	89.22	93.16	71.86	364
VSCNet-G (our)	**99.15**	**99.47**	**98.83**	**99.15**	**91.01**	**156**

- On in-distribution datasets, compared with neural network-based approaches, VSCNet achieves SOTA performance on SAT-and-GET.
- Good generalizability of VSCNet against different formula sizes.
- VSCNet is significantly more efficient than logic-based SOTA approaches while achieving comparable performance.

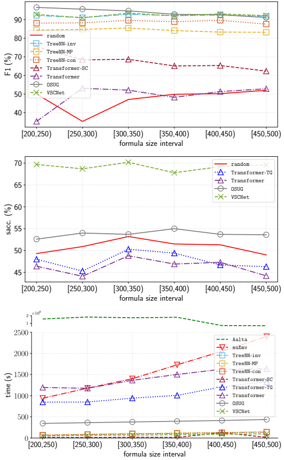


Figure 5: Evaluation results on other *SPOT* test sets.

Comparison on Large-Scale Datasets

Table 4: Evaluation results on the large-scale datasets.

approach	*LTL-as-LTL$_f$* acc.	F1	sacc.	time	*LTL$_f$-Specific* acc.	F1	sacc.	time	*NASA-Boeing* acc.	F1	sacc.	time	*DECLARE* acc.	F1	sacc.	time
random	50.75	65.46	54.96	0	51.59	47.88	92.96	0	62.90	77.23	24.19	0	53.21	69.46	0.00	0
`Aalta`	100.00	100.00	100.00	122,682	100.00	100.00	100.00	36,027	100.00	100.00	100.00	3,601	100.00	100.00	100.00	61,220
`nuXmv`	100.00	100.00	100.00	6,804	100.00	100.00	100.00	766	100.00	100.00	100.00	41	100.00	100.00	100.00	3,743
`TreeNN-inv`	88.05	93.53	-	453	**98.53**	**98.37**	-	305	83.87	91.23	-	33	78.38	87.88	-	176
`TreeNN-MP`	82.18	90.03	-	480	96.29	95.98	-	234	70.97	83.02	-	25	95.50	97.70	-	129
`TreeNN-con`	84.61	91.10	-	475	98.35	98.11	-	230	51.61	68.09	-	25	0.00	0.00	-	129
`Transformer-SC`	72.99	82.64	-	**161**	83.12	75.45	-	**55**	71.05	83.08	-	**3**	0.00	0.00	-	**3**
`OSUG`	8.62	15.87	55.26	11,444	55.71	71.55	93.09	5,173	3.23	6.25	**46.77**	341	10.09	18.33	0.00	15,730
`VSCNet-T` (our)	88.35	**93.71**	**81.46**	368	90.76	90.56	83.80	177	**90.32**	**94.92**	**46.77**	21	95.41	97.65	0.00	94
`VSCNet-G` (our)	**88.85**	94.03	51.68	**91**	45.94	61.66	**93.36**	**25**	83.87	91.23	43.55	**3**	**100.00**	**100.00**	0.00	**6**

- All neural network-based approaches exhibit varying degrees of performance fluctuations on the large-scale datasets.

Comparison on Large-Scale Datasets

Table 4: Evaluation results on the large-scale datasets.

approach	*LTL-as-LTL$_f$* acc.	F1	sacc.	time	*LTL$_f$-Specific* acc.	F1	sacc.	time	*NASA-Boeing* acc.	F1	sacc.	time	*DECLARE* acc.	F1	sacc.	time
random	50.75	65.46	54.96	0	51.59	47.88	92.96	0	62.90	77.23	24.19	0	53.21	69.46	0.00	0
`Aalta`	100.00	100.00	100.00	122,682	100.00	100.00	100.00	36,027	100.00	100.00	100.00	3,601	100.00	100.00	100.00	61,220
`nuXmv`	100.00	100.00	100.00	6,804	100.00	100.00	100.00	766	100.00	100.00	100.00	41	100.00	100.00	100.00	3,743
`TreeNN-inv`	88.05	93.53	-	453	**98.53**	**98.37**	-	305	83.87	91.23	-	33	78.38	87.88	-	176
`TreeNN-MP`	82.18	90.03	-	480	96.29	95.98	-	234	70.97	83.02	-	25	95.50	97.70	-	129
`TreeNN-con`	84.61	91.10	-	475	98.35	98.11	-	230	51.61	68.09	-	25	0.00	0.00	-	129
`Transformer-SC`	72.99	82.64	-	**161**	83.12	75.45	-	**55**	71.05	83.08	-	**3**	0.00	0.00	-	**3**
`OSUG`	8.62	15.87	55.26	11,444	55.71	71.55	93.09	5,173	3.23	6.25	**46.77**	341	10.09	18.33	0.00	15,730
`VSCNet-T` (our)	88.35	**93.71**	**81.46**	368	90.76	90.56	83.80	177	**90.32**	**94.92**	**46.77**	21	95.41	97.65	0.00	94
`VSCNet-G` (our)	**88.85**	94.03	51.68	**91**	45.94	61.66	**93.36**	**25**	83.87	91.23	43.55	**3**	**100.00**	**100.00**	0.00	**6**

- All neural network-based approaches exhibit varying degrees of performance fluctuations on the large-scale datasets.
- `VSCNet` is comparable with the logic-based approaches in LTL satisfiability checking.

Comparison on Large-Scale Datasets

Table 4: Evaluation results on the large-scale datasets.

approach	*LTL-as-LTL$_f$* acc.	F1	sacc.	time	*LTL$_f$-Specific* acc.	F1	sacc.	time	*NASA-Boeing* acc.	F1	sacc.	time	*DECLARE* acc.	F1	sacc.	time
random	50.75	65.46	54.96	0	51.59	47.88	92.96	0	62.90	77.23	24.19	0	53.21	69.46	0.00	0
`Aalta`	100.00	100.00	100.00	122,682	100.00	100.00	100.00	36,027	100.00	100.00	100.00	3,601	100.00	100.00	100.00	61,220
`nuXmv`	100.00	100.00	100.00	6,804	100.00	100.00	100.00	766	100.00	100.00	100.00	41	100.00	100.00	100.00	3,743
`TreeNN-inv`	88.05	93.53	-	453	**98.53**	**98.37**	-	305	83.87	91.23	-	33	78.38	87.88	-	176
`TreeNN-MP`	82.18	90.03	-	480	96.29	95.98	-	234	70.97	83.02	-	25	95.50	97.70	-	129
`TreeNN-con`	84.61	91.10	-	475	98.35	98.11	-	230	51.61	68.09	-	25	0.00	0.00	-	129
`Transformer-SC`	72.99	82.64	-	**161**	83.12	75.45	-	**55**	71.05	83.08	-	**3**	0.00	0.00	-	**3**
`OSUG`	8.62	15.87	55.26	11,444	55.71	71.55	93.09	5,173	3.23	6.25	**46.77**	341	10.09	18.33	0.00	15,730
`VSCNet-T` (our)	88.35	**93.71**	**81.46**	368	90.76	90.56	83.80	177	**90.32**	**94.92**	**46.77**	21	95.41	97.65	0.00	94
`VSCNet-G` (our)	**88.85**	94.03	51.68	**91**	45.94	61.66	**93.36**	**25**	83.87	91.23	43.55	**3**	**100.00**	**100.00**	0.00	**6**

- All neural network-based approaches exhibit varying degrees of performance fluctuations on the large-scale datasets.
- `VSCNet` is comparable with the logic-based approaches in LTL satisfiability checking.
- `VSCNet` is significantly more efficient than the logic-based approaches.

Ablation Study

Table 5: Ablation results of VSCNet-T and its variants on all *SPOT* test sets.

variants	[100, 200)			[200, 250)			[250, 300)			[300, 350)			[350, 400)			[400, 450)			[450, 500)		
	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.
VSCNet-T	93.45	93.16	**71.86**	93.10	92.78	**69.70**	91.60	91.05	**68.70**	93.80	93.47	**70.20**	92.55	92.08	**67.80**	**93.45**	**93.09**	**69.20**	**92.60**	**92.16**	**69.60**
- w/o SS	77.75	71.39	55.51	75.95	68.33	51.90	77.00	70.13	54.00	76.50	69.28	53.00	77.20	70.47	54.40	75.55	67.64	51.10	75.80	68.07	51.60
- w STS	93.66	**93.37**	54.53	**93.15**	**92.81**	51.30	**92.00**	**91.50**	52.10	**94.35**	**94.07**	53.60	**93.30**	**92.90**	52.30	93.35	92.97	51.10	92.45	91.97	48.60
- w/o NTC & w STS	**93.74**	93.44	55.10	92.85	92.44	52.30	91.85	91.28	53.60	93.60	93.22	54.20	**93.30**	92.87	54.60	92.85	92.41	53.70	92.55	92.04	53.90

- VSCNet-T vs. VSCNet-T w STS:
  - The supervision of satisfiable traces is harmful to generating satisfiable traces.
- VSCNet-T vs. VSCNet-T w/o NTC & w STS:
  - NTCNet guides trace generation better than directly using a satisfiable trace for supervision.

Ablation Study

Table 5: Ablation results of VSCNet-T and its variants on all *SPOT* test sets.

variants	[100, 200)			[200, 250)			[250, 300)			[300, 350)			[350, 400)			[400, 450)			[450, 500)		
	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.	acc.	F1	sacc.
VSCNet-T	93.45	93.16	**71.86**	93.10	92.78	**69.70**	91.60	91.05	**68.70**	93.80	93.47	**70.20**	92.55	92.08	**67.80**	**93.45**	**93.09**	**69.20**	**92.60**	**92.16**	**69.60**
- w/o SS	77.75	71.39	55.51	75.95	68.33	51.90	77.00	70.13	54.00	76.50	69.28	53.00	77.20	70.47	54.40	75.55	67.64	51.10	75.80	68.07	51.60
- w STS	93.66	**93.37**	54.53	**93.15**	**92.81**	51.30	**92.00**	**91.50**	52.10	**94.35**	**94.07**	53.60	**93.30**	**92.90**	52.30	93.35	92.97	51.10	92.45	91.97	48.60
- w/o NTC & w STS	**93.74**	93.44	55.10	92.85	92.44	52.30	91.85	91.28	53.60	93.60	93.22	54.20	**93.30**	92.87	54.60	92.85	92.41	53.70	92.55	92.04	53.90

- VSCNet-T vs. VSCNet-T w STS:
  - The supervision of satisfiable traces is harmful to generating satisfiable traces.
- VSCNet-T vs. VSCNet-T w/o NTC & w STS:
  - NTCNet guides trace generation better than directly using a satisfiable trace for supervision.
- VSCNet-T vs. VSCNet-T w/o SS:
  - The joint learning of the module for trace generation and the module for satisfiability checking is mutually reinforcing.
- The ablation study on VSCNet-G exhibits similar results.

Content

Conclusion and Discussion

Conclusion

1. Bridge between LTL trace checking and neural trace checking.
2. Design a joint approach of LTL satisfiability checking and trace generation, which can be realized by a neural network trainable with classical gradient descent.
3. Theoretically prove the correctness of the bridge and empirically confirmed the effectiveness of the joint approach.

Conclusion and Discussion

Conclusion

1. Bridge between LTL trace checking and neural trace checking.
2. Design a joint approach of LTL satisfiability checking and trace generation, which can be realized by a neural network trainable with classical gradient descent.
3. Theoretically prove the correctness of the bridge and empirically confirmed the effectiveness of the joint approach.

Discussion

1. Out-of-distribution datasets: plan to explore parameter-efficient fine-tuning to improve the generalizability in future work.
2. Fixed trace size: in practice, it may be sufficient to set N_t to a small constant for most cases and our approach also works for every LTL formula that has satisfiable traces whose sizes are smaller than N_t.
3. Verifiable unsatisfiable results: future work will also study how to explore neural network technology to generate unsatisfiable cores.

References I

[1] Matteo Bertello, Nicola Gigante, Angelo Montanari, and Mark Reynolds. Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In *IJCAI*, pages 950–956, 2016.

[2] Marco Bozzano, Alessandro Cimatti, Anthony Fernandes Pires, David Jones, Greg Kimberly, T. Petri, R. Robinson, and Stefano Tonetta. Formal design and safety analysis of AIR6110 wheel brake system. In *CAV*, volume 9206, pages 518–535, 2015.

[3] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, pages 334–342, 2014.

[4] Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. Efficient discovery of target-branched declare constraints. *Inf. Syst.*, 56:258–283, 2016.

[5] Renzo Degiovanni, Facundo Molina, Germán Regis, and Nazareno Aguirre. A genetic algorithm for goal-conflict identification. In *ASE*, pages 520–531, 2018.

[6] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and ω -automata manipulation. In *ATVA*, pages 122–129, 2016.

[7] Valeria Fionda and Gianluigi Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018.

[8] Michael Fisher, Clare Dixon, and Martin Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001.

[9] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, and Kristin Yvonne Rozier. Model checking at scale: Automated air traffic control design space exploration. In *CAV*, volume 9780, pages 3–22, 2016.

[10] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *ICLR*, 2021.

References II

[11] Yonit Kesten, Zohar Manna, Hugh McGuire, and Amir Pnueli. A decision algorithm for full propositional temporal logic. In *CAV*, volume 697, pages 97–109, 1993.

[12] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL satisfiability checking revisited. In *TIME*, pages 91–98, 2013.

[13] Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. Aalta: an LTL satisfiability checker over infinite/finite traces. In *FSE*, pages 731–734, 2014.

[14] Jianwen Li, Shufang Zhu, Geguang Pu, and Moshe Y. Vardi. Sat-based explicit LTL reasoning. In *HVC*, volume 9434, pages 209–224, 2015.

[15] Jianwen Li, Geguang Pu, Lijun Zhang, Moshe Y. Vardi, and Jifeng He. Accelerating LTL satisfiability checking by SAT solvers. *J. Log. Comput.*, 28(6):1011–1030, 2018.

[16] Jianwen Li, Lijun Zhang, Shufang Zhu, Geguang Pu, Moshe Y. Vardi, and Jifeng He. An explicit transition system construction approach to LTL satisfiability checking. *Formal Aspects Comput.*, 30(2):193–217, 2018.

[17] Jianwen Li, Shufang Zhu, Geguang Pu, Lijun Zhang, and Moshe Y. Vardi. Sat-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods in System Design*, 54(2):164–190, 2019.

[18] Weilin Luo, Hai Wan, Xiaotong Song, Binhao Yang, Hongzhen Zhong, and Yin Chen. How to identify boundary conditions with contrasty metric? In *ICSE*, pages 1473–1484, 2021.

[19] Weilin Luo, Hai Wan, Delong Zhang, Jianfeng Du, and Hengdi Su. Checking LTL satisfiability via end-to-end learning. In *ASE*, pages 21:1–21:13, 2022.

[20] Weilin Luo, Yuhang Zheng, Rongzhen Ye, Hai Wan, Jianfeng Du, Pingjia Liang, and Polong Chen. Sat-verifiable LTL satisfiability checking via graph representation learning. In *ASE*, pages 1761–1765. IEEE, 2023.

References III

[21] Fabrizio Maria Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In *BPM*, volume 8094, pages 81–96, 2013.

[22] Nicolas Markey and Philippe Schnoebelen. Model checking a path. In *CONCUR*, volume 2761, pages 248–262, 2003.

[23] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *SPIN*, volume 4595, pages 149–167, 2007.

[24] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. *International Journal on Software Tools for Technology Transfer*, 12(2):123–137, 2010.

[25] Viktor Schuppan and Luthfi Darmawan. Evaluating LTL satisfiability solvers. In *ATVA*, volume 6996, pages 397–413, 2011.

[26] Viktor Schuppan. Towards a notion of unsatisfiable cores for LTL. In *FSEN*, volume 5961, pages 129–145. Springer, 2009.

[27] Stefan Schwendimann. A new one-pass tableau calculus for PLTL. In *TABLEAUX*, volume 1397, pages 277–292, 1998.

[28] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[29] Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, pages 119–136, 1985.

[30] Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In *TACAS*, volume 4963, pages 63–77, 2008.

Thank you for your listening!

Contact us: luowlin5@mail.sysu.edu.cn (Weilin Luo)