SAT-verifiable LTL Satisfiability Checking via Graph Representation Learning

1st Weilin Luo
Sun Yat-sen University
Guangzhou, China
luowlin3@mail2.sysu.edu.cn

2nd Yuhang Zheng
Sun Yat-sen University
Guangzhou, China
zhengyh29@mail2.sysu.edu.cn

3rd Rongzhen Ye
Sun Yat-sen University
Guangzhou, China
yerzh@mail2.sysu.edu.cn

4th Hai Wan[†]
Sun Yat-sen University
Guangzhou, China
wanhai@mail.sysu.edu.cn

5th Jianfeng Du[†]
Guangdong University of Foreign Studies
Pazhou Lab
Guangzhou, China
jfdu@gdufs.edu.cn

6th Pingjia Liang
Sun Yat-sen University
Guangzhou, China
liangpj3@mail2.sysu.edu.cn

7th Polong Chen
Sun Yat-sen University
Guangzhou, China
chenplong@mail2.sysu.edu.cn

***Abstract*—With the superior learning ability of neural networks, it is promising to obtain highly confident results for linear temporal logic (LTL) satisfiability checking in polynomial time. However, existing neural approaches are limited in inductive ability and in supporting with an arbitrary number of atomic propositions. Besides, there is no mechanism to verify the results for satisfiability checking. In this paper, we propose an approach to checking the satisfiability of an LTL formula and meanwhile generating a satisfiable trace if the LTL formula is satisfiable, where the satisfiable trace verifies the satisfiability result. The core contribution is a new graph representation for LTL formulae – one-step unfolded graph (OSUG) to incorporate the syntax and semantic features of LTL. Preliminary results show that our approach is superior to the state-of-the-art neural approaches on synthetic datasets and confirms the effectiveness of OSUG.**

***Index Terms*—linear temporal logic, satisfiability checking, graph representation learning**

I. Introduction

Linear temporal logic (LTL) satisfiability checking aims to check whether an LTL formula is satisfiable (SAT) or unsatisfiable (UNSAT). It is the core theoretical problem of LTL and has wide applications, *e.g.*, model checking [1], goal-conflict analysis [2, 3], and business process [4]. However, checking LTL satisfiability is PSPACE-complete [5]. Various logic-based approaches have been proposed. They are mainly based on model checking [6, 7], tableau [8, 9, 10, 11], temporal resolution [12], antichains [13], and SAT [14, 15, 16, 17, 18]. The core of logic-based approaches is to design efficient search heuristics to alleviate the exploding search space. However, designing good heuristics is highly non-trivial. Therefore, developing new approaches over different paradigms remains to be an interesting and necessary research direction.

Recently, Luo et al. (2022) checked LTL satisfiability via end-to-end neural networks [19]. Their work shows some potential to obtain highly confident results for LTL satisfiability checking in polynomial time. However, they require to capture some inductive bias from variable features. It is difficult to match the permutation invariance of atomic propositions (Definition 1), which means that exchanging some atomic propositions in a formula does not change the satisfiability of the formula. Ignoring permutation invariance tends to reduce the ability of neural networks in generalizing bias. Besides, since different vectors are required to distinguish atomic propositions while only m vectors of different atomic propositions are trained, their approach will not be able to solve formulae with more than m different atomic propositions. It severely limits the application of their approach.

By now there is no mechanism to verify the result of the aforementioned neural approaches. Inspired by the work [20], we are able to verify satisfiable results by utilizing neural networks to additionally generate a satisfiable trace in polynomial time. With a polynomial-time trace checking algorithm [21], we can use polynomial time to check LTL satisfiability and perform SAT verification. SAT-verifiable LTL satisfiability checking is crucial for some domains such as safety-critical systems and counterexample-driven approaches. However, their approach [20] fails in the formulae with a large number of variables because the search space of traces is exponential in terms of the number of variables.

In this paper, we propose a new approach to checking the satisfiability of an LTL formula and meanwhile generating

[†]Corresponding authors.

a satisfiable trace if the LTL formula is satisfiable. Its core is a new graph representation for LTL formulae – one-step unfolded graph (OSUG), which incorporates syntax and semantic features. Intuitively, OSUG concatenates each sub-formula through a syntax tree-like structure, which represents the syntax features; OSUG unfolds temporal operators into constraints that need to be satisfied at this time, or at the next time, which represents semantic features. Since OSUG uses different vertices to distinguish different atomic propositions, it is not limited by the maximum number of atomic propositions and conforms to permutation invariance. With the OSUG, we design a feature extractor based on graph neural networks (GNNs). According to different training ways, the extractor is used to capture inductive bias suitable for checking LTL satisfiability and generating traces.

Preliminary results show that, compared with state-of-the-art (SOTA) neural approaches, our approach checks the satisfiability with more than 70X speedup, while still achieving significant performance improvements (average up to 6% improvement in F1 score). Besides, it confirms that GNNs are able to effectively learn the features of LTL formulae from OSUG and to use them for LTL satisfiability checking and trace generating. Our code and datasets are publicly available at https://github.com/sysulic/OSUG.

II. Background

The syntax of linear temporal logic (LTL) [22] is defined by a finite set of atomic propositions $\mathbb{P}$, logical operators ($\vee$ and $\neg$), and temporal modal operators (next ($\bigcirc$) and until ($\mathcal{U}$)). Operator or ($\vee$), release ($\mathcal{R}$), eventually ($\Diamond$), always ($\Box$), and weak-until ($\mathcal{W}$) can be defined by the above fundamental operators. Therefore, for brevity, we only consider the fundamental operators in this paper. Our approach allows us to receive all temporal modal operators of LTL.

LTL formulae are interpreted over infinite traces of propositional states. An infinite trace is represented in the form $\pi = \pi[1], \dots, (\pi[k], \dots, \pi[n])^\omega$, where $\pi[i] \in 2^{\mathbb{P}}$ is a state at time i and $(\pi[k], \dots, \pi[n])^\omega$ is a loop and means that $\pi[k], \dots, \pi[n]$ appears in order and infinitely. The size of trace π is n, denoted by $|\pi|$. π_i denotes a sub-trace of π beginning from the state $\pi[i]$, particularly, $\pi = \pi_1$. The satisfaction relation $\models$ is defined as follows, where ϕ', ϕ'' are LTL formulae, $i, j, k \in \mathbb{N}$, and $p \in \mathbb{P} \cup \{\top\}$:

$$\begin{array}{lll} \pi_i \models p & \text{iff} & p \in \pi[i], \\ \pi_i \models \neg\phi' & \text{iff} & \pi_i \not\models \phi', \\ \pi_i \models \phi' \wedge \phi'' & \text{iff} & \pi_i \models \phi' \text{ and } \pi_i \models \phi'', \\ \pi_i \models \bigcirc\phi' & \text{iff} & w_{i+1} \models \phi', \\ \pi_i \models \phi' \,\mathcal{U}\, \phi'' & \text{iff} & \exists i \le k, w_k \models \phi'' \text{ and} \\ & & \forall i \le j < k, w_j \models \phi'. \end{array}$$

Luo et al. (2022) [19] conclude four propositions of LTL. We show the definition of the permutation invariance of atomic propositions to demonstrate the advantage of our approach.

Proposition 1 (permutation invariance of atomic propositions [19]): Let $\mathbb{P}$ be a set of atomic propositions and ϕ an LTL formula over $\mathbb{P}$. For any $p, q \in \mathbb{P} \cup \{\top\}$, if ϕ is satisfiable, then $\phi[p/\diamond][q/p][\diamond/q]$ is satisfiable, where $\diamond \notin \mathbb{P} \cup \{\top\}$ and $\varphi[\varphi_j/\varphi_i]$ means replacing the sub-formula φ_j of φ with φ_i.

III. Model Architecture

In this section, we first introduce the OSUG and the OSUG-based extractor, then present how to train OSUG-based extractor for satisfiability checking and trace generating.

A. One-step Unfolded Graph

We define the OSUG of LTL shown in Definition 1.

Definition 1: Let ϕ be an LTL formula. Its *one-step unfolded graph (OSUG)* is a two-tuple (V, E), where V is a set of vertices and $E \subseteq V \times V$ is a set of undirected edges. V and E are initialized as $\{v_\phi\}$ and $\emptyset$ respectively. For every sub-formula ϕ_i of ϕ, V and E are computed as follows:

- if $\phi_i = \neg\phi_j$ or $\phi_i = \bigcirc\phi_j$, then $V = V \cup \{v_{\phi_j}\}$, $E = E \cup \{(v_{\phi_i}, v_{\phi_j})\}$;
- if $\phi_i = \phi_j \wedge \phi_k$, then $V = V \cup \{v_{\phi_j}, v_{\phi_k}\}$, $E = E \cup \{(v_{\phi_i}, v_{\phi_j}), (v_{\phi_i}, v_{\phi_k})\}$;
- if $\phi_i = \phi_j \,\mathcal{U}\, \phi_k$, then $V = V \cup \{v_{\phi_j}, v_{\phi_k}, v_{\phi_i'}, v_{\bigcirc\phi_i}\}$, $E = E \cup \{(v_{\phi_i}, v_{\phi_k}), (v_{\phi_i}, v_{\phi_i'}), (v_{\phi_i'}, v_{\phi_j}), (v_{\phi_i'}, v_{\bigcirc\phi_i}), (v_{\bigcirc\phi_i}, v_{\phi_i})\}$,

where ϕ_j, ϕ_k are LTL formulae.

Intuitively, an OSUG incorporates the syntax and semantics of LTL. The core idea behind it is to unfold the sub-formula in one step according to the semantics of LTL, that is, it is equivalently represented as constraints that need to be satisfied at this time or at the next time. For a negation sub-formula $\phi_i = \neg\phi_j$, its equivalent representation by unfolding in one step is $\phi_i \equiv \neg\phi_j \wedge \bigcirc\top$. By equivalent simplification ($\bigcirc\top \equiv \top$), its OSUG and syntax tree are the same. It allows the neural network to capture the syntax features of LTL, and similar ideas also appear in the work [19]. The OSUG of next sub-formula ($\bigcirc\phi_j \equiv \top \wedge \bigcirc\phi_j$) and conjunction sub-formula ($\phi_j \wedge \phi_k \equiv \phi_j \wedge \phi_k \wedge \bigcirc\top$) are similar with the negation sub-formula. For an until sub-formula, its OSUG corresponds to the syntax tree of the equivalent representation of the until sub-formula, *i.e.*, $\phi_i = \phi_j \,\mathcal{U}\, \phi_k \equiv \phi_k \vee (\phi_j \wedge \bigcirc\phi_i)$. Its intuitive understanding decomposing an abstract until relationship into a series of concrete relationship combinations. It is beneficial for the neural network in capturing the semantic features of the until sub-formulae.

Proposition 2: The cost of the conversion from an LTL formula to its OSUG is linear.

Proposition 2 demonstrates that there is no performance burden using OSUGs to perform graph representation learning.

B. Automatic Feature Extractor

Inspired by the wide application of GNNs in graph-structured feature extraction, we extract features of OSUG via sample and aggregate graph convolutional network (SAGE), *e.g.*, GraphSAGE [23]. A SAGE takes the adjacency matrix of an OSUG as the input and outputs the features of each vertex. We add self-loops to each vertex in an OSUG to increase the diversity of features. Considering the permutation invariance of atomic propositions, for an input formula, we initialize each vertex i as a learnable type vector $\mathbf{v}_i^{(0)} \in \mathbb{R}^d$, where d is the embedding size and d is set to 256. We consider two types of vectors: vertices corresponding to atomic propositions and other vertices. These vectors are further fed to SAGE for T iterations, where we set T to 10. Formally, at the t-th iteration, where $t \in [1, T]$, the detailed computation is represented by

$$\mathbf{v}_i^{(t)} = \mathbf{W}_1 \mathbf{v}_i^{(t-1)} + \mathbf{W}_2 \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j^{(t-1)}, \qquad (1)$$

where $\mathbf{v}_i^{(t)}$ denotes the vector of vertex i in t-th iteration, $\mathbf{W}_1$ and $\mathbf{W}_2$ are two learnable weight matrices, and $\mathcal{N}(i)$ is set of neighbors of vertex i. We obtain $\mathbf{v}_i^T$ for $i \in V$ as vertex features, where V is the set of vertices. To obtain a graph feature ($\mathbf{v}_g^{(T)}$), mean pooling is applied on all vertex features. We denote our feature extraction as

$$\left(\{\mathbf{v}_i^{(T)} | i \in V\}, \mathbf{v}_g^{(T)}\right) = \textsc{extract}(\phi, \{\mathbf{v}_i^{(0)} | i \in V\}). \qquad (2)$$

C. LTL Satisfiability Checking

For satisfiability checking, we first utilize our extractor to obtain a graph feature. Then, a two-layer multi-layer perceptron (MLP) with ReLU activation (softmax activation in the final layer) and hidden size 512 is applied to the graph feature to get the probability for the satisfiability of the formula. If the satisfiability is larger than 0.5, then the formula is checked as satisfiable, and unsatisfiable otherwise. Given an LTL formula ϕ, we denote the computation of probability for the satisfiability as $\textsc{check}(\phi)$. Our model is trained by minimizing the cross-entropy loss against the ground truth.

Proposition 3: Let $\mathbb{P}$ be a set of atomic propositions and ϕ an LTL formula over $\mathbb{P}$. For any $p, q \in \mathbb{P} \cup \{\top\}$, $\textsc{check}(\phi) \equiv \textsc{check}(\phi[p/\diamond][q/p][\diamond/q])$.

It is straightforward to prove Proposition 3 via showing that ϕ and $\phi[p/\diamond][q/p][\diamond/q]$ have the same OSUG and the same result of our extractor. Proposition 3 shows that our approach satisfies the permutation invariance of atomic propositions.

D. Trace Generating

For trace generating, we formulate it as a reinforcement learning task and utilize a deep Q-learning network (DQN) to solve it. We define the four ingredients of a DQN namely states, actions, transitions, and reward as follows.

1) State: Let ϕ be an LTL formula over $\mathbb{P}$. A state o is a tuple $(\phi, w, loop)$, where $w = \langle w[1], \ldots \rangle$, $w[i] \in 2^{\mathbb{P}}$ is a state of a trace, $loop = \langle b_0, \ldots \rangle$, and $b_i \in \{0, 1\}$ represents whether $w[i]$ is in the loop. $(w, loop)$ corresponds to a trace. $\mathbf{w} \in \{0,1\}^{|w| \times |\mathbb{P}|}$ and $\mathbf{loop} \in \{0,1\}^{|w|}$ are tensor representations of w and $loop$ respectively. At the start of an episode, we sample a formula and set w and $loop$ to an empty set. The episode terminates after L steps and we set L to 5.

2) Action: An action $a = (a_0, a_1) \in 2^{\mathbb{P}} \times \{0, 1\}$, where a_0 is a state of a trace and a_1 is whether a_0 is in the loop.

3) Transition: A transition at step t is a tuple $\left(o^{(t)}, a^{(t)}, o^{(t+1)}\right)$, where $o^{(t)} = \left(\phi, w^{(t)}, loop^{(t)}\right)$ and $a^{(t)}$ are a state and an action at time t, respectively, and $o^{(t+1)} = (\phi, w^{(t+1)}, loop^{(t+1)}) = (\phi, \langle w^{(t)}, a_0^{(t)} \rangle, \langle loop^{(t)}, a_1^{(t)} \rangle)$.

4) Reward: The reward for a transition includes three parts: (1) sequence constraint: 1 for the prefix $w^{(t+1)}$ not conflict with the ϕ, specially 0 for $t = 0$; (2) loop constraint: 1 for $a_1^{(t-1)} \leq a_1^{(t)}$ and 0 otherwise, specially 0 for $t = 0$; (3) integrity constraint: 10 for $a_1^{(L-1)} = 1$ and 10 for generated trace π satisfies ϕ, respectively.

With our extractor, we design a controller, denoted by $\textsc{generate}$. It takes as inputs $(\phi, \mathbf{w}^{(t)}, \mathbf{loop}^{(t)})$ and computes $(a_0^{(t+1)}, a_1^{(t+1)})$ as outputs, shown in Algorithm 1, where V is the set of vertices of OSUG of ϕ, V_a is the subset of V corresponding to atomic propositions, $x_i^{(j)}$ is the truth value of vertex i at the j-th iteration, specially, 1 for initializing the vertices corresponding to atomic propositions; 2 for others, $l^{(j)}$ means whether the j-th state is in the loop, specially, 3 for initializing, $\texttt{MLP}_i$ is a two-layer MLP with ReLU activation and hidden size 512, the output size of $\texttt{MLP}_1$ and $\texttt{MLP}_2$ are 2 to represent the truth value, and EMB is an embedding layer [24] with hidden size 512. Overall, $\textsc{generate}$ iteratively encodes prefixes of traces utilizing $\textsc{extract}$ (lines 4-7).

Algorithm 1: GENERATE

```
Input   : An LTL formula $\phi$, $\mathbf{w}^{(t)}$, and $\mathbf{loop}^{(t)}$.
Output  : $a_0^{(t+1)}$ and $a_1^{(t+1)}$.
1 $x_i^{(0)} = 1$, for $i \in V_a$; $x_i^{(0)} = 2$, for $i \in V/V_a$; $l^{(0)} = 3$
2 $\mathbf{h}_i^{(0)} = \texttt{MLP}_0(\texttt{CAT}(\mathbf{v}_i^{(0)}, \texttt{EMB}(x_i^{(0)}) + \texttt{EMB}(l^{(0)})))$, for $i \in V$
3 $\{\mathbf{z}_i^{(0)} | i \in V\}, \mathbf{z}_g^{(0)} = \textsc{extract}(\phi, \{\mathbf{h}_i^{(0)} | i \in V\})$
4 foreach $u \in [1, |w|]$ do
5     $x_i^{(u)} = (\mathbf{w}^{(t)})_{u,i}$, for $i \in V_a$; $x_i^{(u)} = 2$, for $i \in V/V_a$;
       $l^{(u)} = (\mathbf{loop}^{(t)})_u$
6     $\mathbf{h}_i^{(u)} = \texttt{MLP}_0(\texttt{CAT}(\mathbf{z}_i^{(u-1)}, \texttt{EMB}(x_i^{(u)}) + \texttt{EMB}(l^{(u)})))$, for $i \in V$
7     $\{\mathbf{z}_i^{(u)} | i \in V\}, \mathbf{z}_g^{(u)} = \textsc{extract}(\phi, \{\mathbf{h}_i^{(u)} | i \in V\})$
8 return $\{i \in V | \arg\max(\texttt{MLP}_1(\mathbf{z}_i^{|w|})) = 1\}, \arg\max(\texttt{MLP}_2(\mathbf{z}_g^{|w|}))$
```

IV. Preliminary Experiment

A. Setting

Competitor. We denote our approach as `OSUG-SAGE`. We compare `OSUG-SAGE` with two SOTA neural approaches in checking LTL satisfiability: `TreeNN-inv` [19] and `TreeNN-con` [19], and a SOTA neural approach in trace generating: `Transformer` [20].

Dataset. We use synthetic datasets [19] to train and test neural networks, denoted by *SPOT*. Since `Transformer` requires a satisfiable trace as supervision, we use nuXmv [25] to generate a satisfiable trace for each satisfiable formula in datasets.

Setup. For satisfiability checking, we train all neural networks on the training set of *SPOT*-$[100, 200)$, and test them on the test set of *SPOT*-$[100, 200)$ and *SPOT* with larger formulae. For trace generating, we only use the satisfiable formulae in *SPOT*-$[100, 200)$, which follows the work [20].

The evaluation metrics of the performance of satisfiability checking include accuracy (acc.), precision (pre.), recall (rec.), and F1 score (F1). The evaluation metric of performance in generating traces is semantic accuracy (sem. acc.). If a generated trace satisfies the given formula, then it is semantically

TABLE I
EVALUATION RESULTS ON *SPOT*-$[100, 200)$

approach	satisfiability checking					trace generating	
	acc.	pre.	rec.	F1	time	sem. acc.	time
random	50.00	50.00	50.00	50.00	-	51.73	-
`TreeNN-con`	93.61	97.70	89.32	93.32	5,371.56	-	-
`TreeNN-inv`	93.42	97.45	89.18	93.13	4,968.67	-	-
`Transformer`	-	-	-	-	-	50.52	12,880.00
`OSUG-SAGE`	**98.48**	**99.58**	**98.89**	**99.23**	**68.11**	**54.95**	**2,676.12**

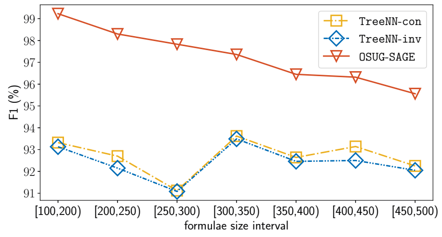

Fig. 1. Evaluation results on the larger formulae. The accuracy, precision, and recall for neural networks have the same rankings and trends as F1 score.

accurate. We use the algorithm [21] to check if a generated trace is semantically accurate.

Both training and testing of all neural networks are conducted on a single GPU (NVIDIA A100). We train all neural networks using the Adam optimizer. For `TreeNN-inv`, `TreeNN-con`, and `Transformer`, we use the hyperparameter settings reported in their papers. For our approach, we use grid search to find optimal hyperparameters.

B. Experimental Result

1) Comparison with SOTA Neural Approaches: Table I shows the results (%) on the test set of *SPOT*-$[100, 200)$, where "time" indicates the sum of the running time (s) for all formulae, "-" marks that the approach cannot solve, and **Bold** numbers mark better results. We show the results of random guesses in checking satisfiability and generating traces ("random"). The performance of `OSUG-SAGE` in checking LTL satisfiability exceeds that of `TreeNN-con` and `TreeNN-inv` in all the evaluation metrics and has a significant improvement in running speed (70 times faster). These results suggest that our approach significantly outperforms SOTA approaches.

In trace generating, `Transformer` fails since its performance is close to that of random guessing. It is due to that the search space of traces is exponentially related to the number of atomic propositions. The work [20] only considered formulae with 5 atomic propositions, while the average number of atomic propositions in our formulae is 34.27. The performance of `OSUG-SAGE` is slightly higher than random guessing, indicating that our approach captures some features suitable for generating traces. However, it is still challenging to generate traces.

Fig. 1 demonstrates the generalization ability of approaches across formula sizes. Note that the three approaches are only trained on small formulae. Like `TreeNN-con` and `TreeNN-inv`, the performance of `OSUG-SAGE` only slightly decreases when the formula sizes get larger. This suggests that our approach has a strong generalization to the different distributions of formula sizes. Meanwhile, the performance of `OSUG-SAGE` exceeds that of `TreeNN-con` and `TreeNN-inv` by a large gap across all distributions, which also demonstrates the advantage of our graph representation and neural architecture.

TABLE II
EVALUATION RESULTS ABOUT VARIANTS OF `OSUG-SAGE`

approach		$[100, 200)$	$[200, 250)$	$[250, 300)$	$[300, 350)$	$[350, 400)$	$[400, 450)$	$[450, 500)$	average
GCN	OSUG	98.66	97.46	96.91	95.75	94.85	94.21	93.36	95.89
	ST	89.14	88.64	88.80	87.42	87.26	88.73	87.23	88.17
GT	OSUG	79.09	78.93	78.05	78.97	79.23	77.86	77.94	78.58
	ST	74.83	74.17	74.33	72.77	72.12	73.66	70.97	73.27
RGGC	OSUG	97.66	95.64	95.40	94.32	91.60	88.58	71.96	90.74
	ST	91.64	90.77	91.19	90.53	90.68	89.53	88.77	90.44
GAT	OSUG	77.88	77.68	78.64	78.31	78.75	78.75	76.35	78.05
	ST	79.72	78.82	79.92	77.34	78.60	79.41	79.52	79.05
SAGE	OSUG	99.23	98.30	97.83	97.36	96.45	96.32	95.56	97.29
	ST	89.76	89.23	89.01	86.94	87.36	87.51	85.78	87.94

2) Effectiveness of OSUG: We compare variants of `OSUG-SAGE` powered by different GNNs and different graph representations. The GNNs include graph convolutional network (GCN) [26], graph Transformer (GT) [27], residual gated graph convolutional network (RGGC) [28], and graph attention network (GAT) [29]. The graph representations include syntax tree (ST) and OSUG. Table II shows the F1 score in checking satisfiability. All five different GNNs achieve better performance when learning on OSUG, except GAT. We argue that the special case of GAT is due to its powerful attention mechanism. This suggests that learning on OSUG is able to capture better features in almost all types of GNNs.

V. CONCLUSION AND FUTURE WORK

Neural approaches are promising to fast obtain highly confident results for LTL satisfiability checking. In this paper, we have proposed a new graph representation, OSUG, to achieve SOTA performance, which further improves the speed and confidence of neural approaches. Furthermore, we have made SAT-verifiable checking by additionally generating a satisfiability trace. However, it is still a challenging problem to generate traces. Future work will extend our approach to validate the effectiveness of neural networks on intractable industrial instances, improve the performance of trace generating, and propose SAT-UNSAT-verifiable end-to-end neural networks for checking LTL satisfiability.

ACKNOWLEDGMENT

This paper was supported by the National Natural Science Foundation of China (No. 62276284, 61976232, 61876204), the National Key Research and Development Program of China (No. 2021YFA1000504), the Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515011470, 2022A1515011355, 2020A1515010642), the Guangzhou Science and Technology Project (No. 202201011699), the Guizhou Provincial Science and Technology Projects (No. 2022-259), the Humanities and Social Science Research Project of Ministry of Education (No. 18YJCZH006), as well as the Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No. 23ptpy31).

References

[1] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 - A framework for LTL and ω -automata manipulation," in *ATVA*, 2016, pp. 122–129.

[2] R. Degiovanni, F. Molina, G. Regis, and N. Aguirre, "A genetic algorithm for goal-conflict identification," in *ASE*, 2018, pp. 520–531.

[3] W. Luo, H. Wan, X. Song, B. Yang, H. Zhong, and Y. Chen, "How to identify boundary conditions with contrasty metric?" in *ICSE*, 2021, pp. 1473–1484.

[4] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *BPM*, vol. 8094, 2013, pp. 81–96.

[5] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[6] K. Y. Rozier and M. Y. Vardi, "LTL satisfiability checking," in *SPIN*, vol. 4595, 2007, pp. 149–167.

[7] ——, "LTL satisfiability checking," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 2, pp. 123–137, 2010.

[8] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli, "A decision algorithm for full propositional temporal logic," in *CAV*, vol. 697, 1993, pp. 97–109.

[9] P. Wolper, "The tableau method for temporal logic: An overview," *Logique et Analyse*, pp. 119–136, 1985.

[10] S. Schwendimann, "A new one-pass tableau calculus for PLTL," in *TABLEAUX*, vol. 1397, 1998, pp. 277–292.

[11] M. Bertello, N. Gigante, A. Montanari, and M. Reynolds, "Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau," in *IJCAI*, 2016, pp. 950–956.

[12] M. Fisher, C. Dixon, and M. Peim, "Clausal temporal resolution," *ACM Trans. Comput. Log.*, vol. 2, no. 1, pp. 12–56, 2001.

[13] M. D. Wulf, L. Doyen, N. Maquet, and J. Raskin, "Antichains: Alternative algorithms for LTL satisfiability and model-checking," in *TACAS*, vol. 4963, 2008, pp. 63–77.

[14] J. Li, L. Zhang, G. Pu, M. Y. Vardi, and J. He, "LTL satisfiability checking revisited," in *TIME*, 2013, pp. 91–98.

[15] J. Li, Y. Yao, G. Pu, L. Zhang, and J. He, "Aalta: an LTL satisfiability checker over infinite/finite traces," in *FSE*, 2014, pp. 731–734.

[16] J. Li, S. Zhu, G. Pu, and M. Y. Vardi, "Sat-based explicit LTL reasoning," in *HVC*, vol. 9434, 2015, pp. 209–224.

[17] J. Li, G. Pu, L. Zhang, M. Y. Vardi, and J. He, "Accelerating LTL satisfiability checking by SAT solvers," *J. Log. Comput.*, vol. 28, no. 6, pp. 1011–1030, 2018.

[18] J. Li, S. Zhu, G. Pu, L. Zhang, and M. Y. Vardi, "Sat-based explicit LTL reasoning and its application to satisfiability checking," *Formal Methods in System Design*, vol. 54, no. 2, pp. 164–190, 2019.

[19] W. Luo, H. Wan, D. Zhang, J. Du, and H. Su, "Checking LTL satisfiability via end-to-end learning," in *ASE*, 2022, pp. 21:1–21:13.

[20] C. Hahn, F. Schmitt, J. U. Kreber, M. N. Rabe, and B. Finkbeiner, "Teaching temporal logics to neural networks," in *ICLR*, 2021.

[21] N. Markey and P. Schnoebelen, "Model checking a path," in *CONCUR*, vol. 2761, 2003, pp. 248–262.

[22] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[23] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

[25] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *CAV*, 2014, pp. 334–342.

[26] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[27] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *IJCAI*, 2021, pp. 1548–1554.

[28] X. Bresson and T. Laurent, "Residual gated graph convnets," *CoRR*, vol. abs/1711.07553, 2017.

[29] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.