Teaching LTL$_f$ Satisfiability Checking to Neural Networks

Weilin Luo[1] , **Hai Wan**[1,2∗] , **Jianfeng Du**[3,4∗] , **Xiaoda Li**[1] ,
Yuze Fu[1] , **Rongzhen Ye**[1] and **Delong Zhang**[1]
[1]School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2]Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China
[3]Guangdong University of Foreign Studies, Guangzhou, China
[4]Pazhou Lab, Guangzhou, China
{luowlin3,lixd36,fuyz,yerzh,zhangdlong3}@mail2.sysu.edu.cn, wanhai@mail.sysu.edu.cn, jfdu@gdufs.edu.cn

Abstract

Linear temporal logic over finite traces (LTL$_f$) satisfiability checking is a fundamental and hard (PSPACE-complete) problem in the artificial intelligence community. We explore teaching end-to-end neural networks to check satisfiability in polynomial time. It is a challenge to characterize the syntactic and semantic features of LTL$_f$ via neural networks. To tackle this challenge, we propose `LTLfNet`, a recursive neural network that captures syntactic features of LTL$_f$ by recursively combining the embeddings of sub-formulae. `LTLfNet` models permutation invariance and sequentiality in the semantics of LTL$_f$ through different aggregation mechanisms of sub-formulae. Experimental results demonstrate that `LTLfNet` achieves good performance in synthetic datasets and generalizes across large-scale datasets. They also show that `LTLfNet` is competitive with state-of-the-art symbolic approaches such as nuXmv and CDLSC.

1 Introduction

Linear temporal logic over finite traces (LTL$_f$) [Giacomo and Vardi, 2013] is one of the popular modal logics in artificial intelligence (AI). Since LTL$_f$ is well-defined and unambiguous, it has been widely applied to formalize and validate system behaviors. *LTL$_f$ satisfiability checking* is a fundamental problem of LTL$_f$, *i.e.*, checking whether a given LTL$_f$ formula is satisfiable or unsatisfiable. Compared with *Linear temporal logic (LTL)* [Pnueli, 1977], LTL$_f$ is interested in finite traces [Giacomo and Vardi, 2013]. Therefore, LTL$_f$ is more attractive in AI focusing on finite behaviors, such as reinforcement learning [Xie *et al.*, 2021], program synthesis [Xiao *et al.*, 2021], and explainable AI [Kim *et al.*, 2019].

The complexity of LTL$_f$ satisfiability checking is PSPACE-complete [Giacomo and Vardi, 2013]. Lots of approaches have tried to efficiently solve this problem. They are all based on crafted symbolic reasoning, *e.g.*, based on tableau [Li *et al.*, 2014] and based on SAT [Fionda and Greco, 2016; Li *et al.*, 2020]. We refer to these approaches as *symbolic approaches*. A symbolic approach needs to be sound and complete, *i.e.*, it must correctly answer whether a given formula is satisfiable or not. Given the inherent intractability of LTL$_f$ satisfiability checking, no symbolic approach scales to all datasets [Li *et al.*, 2020]. Therefore, developing new methods for all different paradigms remains to be an interesting research direction [Li *et al.*, 2020].

Recently, some approaches tried to introduce end-to-end neural networks to solve Boolean satisfiability (SAT) problems [Selsam *et al.*, 2019; Cameron *et al.*, 2020]. Their success stems from the fact that neural networks are able to capture *permutation invariance* of the semantics of Boolean formulae. A task is permutation invariant if permutations of its input do not change the output. Although these works are not competitive with state-of-the-art (SOTA) symbolic approaches, they show the potential of neural networks in solving hard computational problems. This poses an interesting question: *whether LTL$_f$ satisfiability checking can be effectively tackled by end-to-end neural networks?*

To answer this question, we explore teaching end-to-end neural networks to check LTL$_f$ satisfiability. It is a challenge to integrate symbolic reasoning of LTL$_f$ with continuous neural reasoning, as it requires neural networks to capture the permutation invariance of LTL$_f$. Some logical operators (*e.g.*, $\wedge$ operators) of LTL$_f$ are permutation invariant, *i.e.*, arbitrarily changing the position of the sub-formula, the satisfiability of the formula remains unchanged. For example, both $(north \vee west)$ U $door$ and $(west \vee north)$ U $door$ are satisfiable. Besides, neural networks need to model the sequentiality of LTL$_f$. Some logical operators (*e.g.*, U operator) of LTL$_f$ is sequential, *i.e.*, changing the position of the sub-formula, the satisfiability of the formula can be changed. For example, $(door$ U $west) \wedge \mathsf{G}\neg door$ is satisfiable while $(west$ U $door) \wedge \mathsf{G}\neg door$ is unsatisfiable.

In this paper, we propose `LTLfNet`, a recursive neural network, to predict the satisfiability of a given LTL$_f$ formula end-to-end. While symbolic approaches scales exponentially with the formula size, `LTLfNet` spends only polynomial time to predict the result. Although `LTLfNet` does not guarantee

[∗]Corresponding author.

soundness, its efficient framework and highly confident prediction are still worthwhile for some LTL$_f$-SAT-heavy tasks. To bridge the gap between symbolic reasoning of LTL$_f$ and continuous neural reasoning, `LTLfNet` not only captures the syntactic features of LTL$_f$ by combining and aggregating features recursively over the syntax tree, but also models the permutation invariance and sequentiality of LTL$_f$ (semantic features). Specifically, to characterize the different semantics of logical operators, `LTLfNet` learns independent combination function for each logical operator as well as exploits aggregation functions fulfilling the permutation invariance or sequentiality for each logical operator.

Empirical evaluation on synthetic datasets shows that `LTLfNet` outperforms existing architectures that have been used to embed LTL or LTL$_f$ formulae. Besides, we also evaluate `LTLfNet` on some large-scale datasets. Results demonstrate strong generalization of `LTLfNet` and competitive results with the SOTA symbolic approaches such as nuXmv [Cavada *et al.*, 2014] and CDLSC [Li *et al.*, 2020].

2 Related Work

Symbolic approaches to LTL$_f$ satisfiability checking. The classical solution to LTL$_f$ satisfiability checking is reducing it to LTL satisfiability checking, which has been studied for decades and many tools are available, *e.g.*, nuXmv [Cavada *et al.*, 2014]. However, an extra cost has to be paid when checking LTL formulae, because the traces in LTL satisfiability checking are infinite while those are finite in LTL$_f$ satisfiability checking. Therefore, Li *et al.* (2014) presented a tableau-style algorithm for LTL$_f$ satisfiability checking so as to avoid this cost. Inspired by recent dramatic improvements in propositional SAT solving, Li *et al.* (2020) proposed an LTL$_f$ satisfiability checker via SAT-based model checking and demonstrated outstanding performance.

Permutation invariance. Permutation invariance is a key property in many symbolic logical reasoning mechanisms. A task is permutation invariant if permutations of its input leave the output unchanged. Take SAT problem for example, satisfiability status is unaffected by the permutation of variables and clauses. A number of recent works have studied neural network based methods for permutation invariance, such as sets [Zaheer *et al.*, 2017; Lee *et al.*, 2019], matrices and tensors [Hartford *et al.*, 2018; Cameron *et al.*, 2020], pooling [Murphy *et al.*, 2019; Zhang *et al.*, 2020], and graph structured data [Cameron *et al.*, 2020]. All these methods build neural network layers respecting the permutation invariance, but they differ in how to aggregate the embeddings and which permutation-invariant function will be used in aggregation.

End-to-end neural networks for SAT. End-to-end neural networks for the SAT problem are often designed delicately to keep the property of permutation invariance so as to improve performance. Amizadeh *et al.* (2019) used an aggregation function in the model, which is invariant to the permutation of its input, to keep the property. Selsam *et al.* (2019) built a graph of the propositional formula and enforced permutation invariance by managing nodes and edges according to the topology of the graph without any additional ordering. Cameron *et al.* (2020) demonstrated that two different architectural approaches, *i.e.*, exchangeable architecture [Hartford *et al.*, 2018] and message passing [Selsam *et al.*, 2019], can outperform previous approaches on random 3-SAT problems. Inspired by their work, we explore efficient end-to-end neural networks for LTL$_f$ satisfiability checking that can capture permutation invariance and sequentiality of LTL$_f$.

Continuous neural representation of logical expressions. It is hard to embed logical expressions because of the gap between the continuous neural representation and the discrete semantic definition of logical expressions. Nevertheless, there is work attempting to bridge this gap. Allamanis *et al.* (2017) proposed a TreeNN [Socher *et al.*, 2013] based approach to learning the identical representation of the syntactic parse tree for semantically equivalent Boolean expressions. Evans *et al.* (2018) compared the abilities of different approaches in capturing and exploiting the structure of logical expressions against an entailment prediction task. They also proposed a model class named PossibleWorldNets, which computes entailment as a "convolution over possible worlds". Paliwal *et al.* (2020) used message passing graph neural networks to learn the representation of the modified abstract syntax tree of higher-order logic expressions. They demonstrated the benefits of modeling syntactic features of formulae to embed logical expressions.

Embedding LTL/LTL$_f$ formulae. In recent years, some works have tried to find a good way to embed LTL/LTL$_f$ formulae and explored the applications of LTL/LTL$_f$ formula embeddings. Xie *et al.* (2021) sought to incorporate temporal knowledge into deep sequential learning. They transferred an LTL$_f$ formula to a semantically equivalent deterministic finite-state automaton (DFA) and used the graph embeddings of the DFA as the representation of the LTL$_f$ formula. Empirical results confirmed that their approach improved deep models for sequential human action recognition and imitation learning. However, their approach is hard to handle long formulae because the size of the DFA grows exponentially with the formula size. Since the size of most formulae in industrial datasets is more than 1k, we do not compare their approach with ours in our experiments. Vaezipoor *et al.* (2021) taught a deep reinforcement learning agent to follow instructions expressed in LTL in multi-task environments. They used relational graph convolutional network (R-GCN) [Schlichtkrull *et al.*, 2018] to embed the parse tree for LTL formulae. Hahn *et al.* (2021) explored the ability of Transformer [Vaswani *et al.*, 2017] to predict a trace satisfying the given LTL formula. They directly employed Transformer to encode the LTL$_f$ formulae in Polish notation and decode a satisfying trace. Since experimental results show the potential of neural networks in predicting a satisfying trace of LTL$_f$ formulae, they suggested that deep learning can already augment combinatorial approaches in automatic verification and the broader formal methods community [Hahn *et al.*, 2021]. In comparison with Transformer and R-GCN in LTL$_f$ satisfiability checking, our proposed approach is shown to have more advantages in our experiments. Our superiority is probably due to the fact that our approach explicitly models permutation invariance and sequentiality, but Transformer and R-GCN do not.

3 Preliminaries

The syntax of LTL$_f$ for a finite set of atomic propositions $\mathbb{P}$ includes *true* ($\top$), standard logical operators (*conjunction* ($\wedge$) and *negation* ($\neg$)), and temporal logical operators (*next* (X) and *until* (U)), described as follows:

$$\phi := \top \mid p \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \phi_1 \mathsf{U} \phi_2,$$

where $p \in \mathbb{P} \cup \{\top\}$ and ϕ, ϕ_1, and ϕ_2 are LTL$_f$ formulae. For brevity, we only consider the above fundamental operators in this paper. Operator *disjunction* ($\vee$), *weak next* (N), *release* (R), *eventually* (F), and *always* (G) are commonly used, and can be defined as $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\mathsf{N}\phi_1 := \neg\mathsf{X}\top \vee \mathsf{X}\phi_1$, $\phi_1\mathsf{R}\phi_2 := \neg(\neg\phi_1 \ \mathsf{U} \ \neg\phi_2)$, $\mathsf{F}\phi_1 := \top \ \mathsf{U} \ \phi_1$, and $\mathsf{G}\phi_1 := \neg(\top \ \mathsf{U} \ \neg\phi_1)$ respectively. The *sub-formula* of an LTL$_f$ formula ϕ is defined as follows, where $p \in \mathbb{P}\cup\{\top\}$ and ϕ_1, ϕ_2 are LTL$_f$ formulae: (1) if $\phi = p$, then p is sub-formula of ϕ; (2) if $\phi = \neg\phi_1$, then $\neg\phi_1$ and the sub-formulae of ϕ_1 are sub-formulae of ϕ; (3) if $\phi = \phi_1 \wedge \phi_2$, then $\phi_1 \wedge \phi_2$, the sub-formulae of ϕ_1, and the sub-formulae of ϕ_2 are sub-formulae of ϕ; (4) if $\phi = \mathsf{X}\phi_1$, then $\mathsf{X}\phi_1$ and the sub-formulae of ϕ_1 are sub-formulae of ϕ; (5) if $\phi = \phi_1 \ \mathsf{U} \ \phi_2$, then $\phi_1 \ \mathsf{U} \ \phi_2$, the sub-formulae of ϕ_1, and the sub-formulae of ϕ_2 are sub-formulae of ϕ. The set of sub-formulae of ϕ is denoted by $\mathtt{sub}(\phi)$. The *size* of an LTL$_f$ formula ϕ is the number of logical operators and atomic propositions in ϕ, denoted by $|\phi|$.

LTL$_f$ is interpreted over finite *traces*. An finite trace is represented in the form $\pi = s_0, s_1, \ldots, s_n$, where $s_t \in 2^{\mathbb{P}}$ is a state at time t. For every state s_i of π and every $p \in \mathbb{P}$, p holds if $p \in s_i$ or $\neg p$ holds otherwise. The traces mentioned in this paper are finite. The size of π is the number of states of π, denoted by $|\pi|$. π_i denotes a *sub-trace* of π beginning from the state s_i. Let π is a trace and $|\pi| = n$. The *satisfaction relation* $\models$ is defined as follows:

$$\begin{array}{lll}
\pi_i \models p & \text{iff} & p \in s_i \\
\pi_i \models \neg\phi & \text{iff} & \pi_i \not\models \phi \\
\pi_i \models \phi_1 \wedge \phi_2 & \text{iff} & \pi_i \models \phi_1 \text{ and } \pi_i \models \phi_2 \\
\pi_i \models \mathsf{X}\phi & \text{iff} & i < n \text{ and } \pi_{i+1} \models \phi \\
\pi_i \models \phi_1 \ \mathsf{U} \ \phi_2 & \text{iff} & \exists i \leq k \leq n, \pi_k \models \phi_2 \text{ and} \\
 & & \forall i \leq j < k, \pi_j \models \phi_1.
\end{array}$$

where ϕ, ϕ_1, ϕ_2 are LTL$_f$ formulae, and $p \in \mathbb{P} \cup \{\top\}$. An LTL$_f$ formula ϕ is *satisfiable* if and only if there is a trace π such that $\pi_0 \models \varphi$; otherwise it is *satisfiable*.

In this paper, we focus on the task of LTL$_f$ satisfiability checking. Its input is an LTL$_f$ formula ϕ and the output is a binary classification predicting whether ϕ is satisfiable or unsatisfiable. For example, given an LTL$_f$ formula $(west \vee north) \ \mathsf{U} \ door$, it is satisfiable because there is a trace such that $west \vee north$ is true until the $door$ is true. In contrast, $((west \vee north) \ \mathsf{U} \ door) \wedge \mathsf{G}\neg door$ is unsatisfiable because $door$ being always false ($\mathsf{G}\neg door$) contradicts $door$ being true in the future ($(west \vee north) \ \mathsf{U} \ door$)).

4 Approach

The core neural network of our approach is named `LTLfNet`. Overall, `LTLfNet` first embeds a given LTL$_f$ formula, then uses the embedding to perform a binary classification (satisfiable or unsatisfiable). In order to capture the syntactic features and handle formulae of any size, we employ the general framework of recursive neural networks (TreeNN) [Socher *et al.*, 2011] to embed the *syntax tree* of a given LTL$_f$ formula. The syntax tree of LTL$_f$ formula is defined as follows. An example of syntax tree is shown in Figure 1.

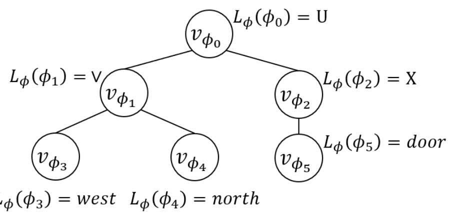

Figure 1: The syntax tree of $(west \vee north) \ \mathsf{U} \ \mathsf{X}door$.

Definition 1. *Let ϕ be an LTL$_f$ formula. Its* syntax tree *G_ϕ is a four-tuple $(V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi)$ defined as follows, where V_ϕ is a set of vertices, $r_\phi \in V_\phi$ is the* root vertex, *$E_\phi \subseteq V_\phi \times V_\phi$ is a set of undirected edges, and $\mathtt{L}_\phi$: $V_\phi \to \{\top\} \cup \mathbb{P} \cup \{\neg, \wedge, \mathsf{X}, \mathsf{U}\}$. V_ϕ and E_ϕ are initialized as $\{v_\phi\}$ and $\emptyset$, respectively. $r_\phi = v_\phi$. For each sub-formula $\phi_i \in \mathtt{sub}(\phi)$, V_ϕ, E_ϕ, and $\mathtt{L}_\phi$ are constructed as follows:*

- *if $\phi_i = p$, then $\mathtt{L}_\phi(\phi_i) = p$;*
- *if $\phi_i = \mathtt{o}_1 \ \phi_j$, then $V_\phi = V_\phi \cup \{v_{\phi_j}\}$, $E_\phi = E_\phi \cup \{(v_{\phi_i}, v_{\phi_j})\}$, and $\mathtt{L}_\phi(\phi_i) = \mathtt{o}_1$;*
- *if $\phi_i = \phi_j \ \mathtt{o}_2 \ \phi_k$, then $V_\phi = V_\phi \cup \{v_{\phi_j}, v_{\phi_k}\}$, $E_\phi = E_\phi \cup \{(v_{\phi_i}, v_{\phi_j}), (v_{\phi_i}, v_{\phi_k})\}$, and $\mathtt{L}_\phi(\phi_i) = \mathtt{o}_2$,*

where $p \in \mathbb{P} \cup \{\top\}$, $\mathtt{o}_1 \in \{\neg, \mathsf{X}\}$, $\mathtt{o}_2 \in \{\wedge, \mathsf{U}\}$, and ϕ_j, ϕ_k are LTL$_f$ formulae.

Algorithm 1: LTLFEMBEDDING

```
Input  : a syntax tree $(V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi)$ of $\phi$ and
         current vertex $r_{\phi_i}$.
Output : the embedding $\mathbf{r}_{\phi_i}$ of $\phi_i$.
1  if $\mathtt{L}_\phi(r_{\phi_i}) = p$, where $p \in \mathbb{P} \cup \{\top\}$ then
2      $\mathbf{r}_{\phi_i}$ ←ONE-HOTEMBEDDING($p$)
3  else if $\mathtt{L}_\phi(r_{\phi_i}) = op_1$, where $op_1 \in \{\neg, \mathsf{X}\}$ then
4      get $r_{\phi_j} \in V_\phi$ s.t. $(r_{\phi_i}, r_{\phi_j}) \in E_\phi$
5      $\mathbf{r}_{\phi_j}$ ←LTLFEMBEDDING($(V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi)$,$r_{\phi_j}$)
6      $\mathbf{r}_{\phi_i}$ ←COMBINE($\mathbf{r}_{\phi_j}$,$op_1$)
7  else
8      get $r_{\phi_j}, r_{\phi_k} \in V_\phi$ s.t. $(r_{\phi_i}, r_{\phi_j}), (r_{\phi_i}, r_{\phi_k}) \in E_\phi$
9      $\mathbf{r}_{\phi_j}$ ←LTLFEMBEDDING($(V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi)$,$r_{\phi_j}$)
10     $\mathbf{r}_{\phi_k}$ ←LTLFEMBEDDING($(V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi)$,$r_{\phi_k}$)
11     if $\mathtt{L}_\phi(r_\phi) = \wedge$ then
12         $\mathbf{r}_{\phi_i}$ ←COMBINE(MP($\mathbf{r}_{\phi_j}$,$\mathbf{r}_{\phi_k}$),$\wedge$)
13     else /* $\phi = \phi_i \ \mathsf{U} \ \phi_k$ */
14         $\mathbf{r}_{\phi_i}$ ←COMBINE([$\mathbf{r}_{\phi_i}, \mathbf{r}_{\phi_j}$],U)
15 return $\mathbf{r}_\phi$
```

Algorithm 1 shows the pseudo-code of embedding LTL$_f$ formulae. LTLFEMBEDDING learns embeddings of sub-formula for each vertex in the syntax tree by recursively combining and aggregating the embeddings of its sub-formulae using a neural network. If the sub-formula is an atomic proposition or $\top$, we use a one-hot vector $\mathbf{r}_p \in \mathbb{R}^{d_m}$ (line 2 of Algorithm 1). Otherwise, we perform the function COMBINE with different learnable parameters for different logical operators. The pseudo-code of COMBINE is shown in Algorithm 2. We use a two-layer multilayer perceptron (MLP) with a residual-like connection to compute the combination representation of sub-formulae, which is the same as the work [Allamanis *et al.*, 2017]. Note that the dimensions of $\mathbf{W}_{0,op}, \mathbf{W}_{1,op}, \mathbf{W}_{2,op}$ are different from each other and related to the array of the operators.

```
Algorithm 2: COMBINE
Input  : an aggregated representation r of
         sub-formulae and the logical operator op.
Output : the combination representation $\mathbf{r}_{out}$.
1 $\mathbf{r}' \leftarrow \mathsf{ReLU}(\mathbf{W}_{0,op} \cdot \mathbf{r})$
2 $\mathbf{r}_{out} \leftarrow \mathbf{W}_{1,op} \cdot \mathbf{r}' + \mathbf{W}_{2,op} \cdot \mathbf{r}$
3 return $\mathbf{r}_{out}/\|\mathbf{r}_{out}\|_2$
```

We define aggregation function for each operator fulfilling permutation invariance or sequentiality as follows.

- **Permutation invariance.** If a formula and its sub-formulae are connected by an unary operator (line 6 of Algorithm1) or the $\wedge$ operator (line 12 of Algorithm 1), we aggregate the embeddings of sub-formulae by performing a mean pooling (MP).
- **Sequentiality.** If they are connected by the U operator (line 14 of Algorithm 1), we aggregate the embeddings of sub-formulae by concatenating them.

Given an LTL$_f$ formula ϕ, `LTLfNet` computes the probability $\hat{y_\phi}$ for ϕ being satisfiable as follow:

$$\mathbf{r}_\phi = \text{LTLfEmbedding}((V_\phi, E_\phi, r_\phi, \mathtt{L}_\phi), \phi),$$
$$\hat{y_\phi} = \mathtt{MLP}_{sat}(\mathbf{r}_\phi),$$

where $\mathtt{MLP}_{sat}$ is an MLP. We train `LTLfNet` to minimize the sigmoid cross-entropy loss between $\hat{y_\phi}$ and the ground truth y_ϕ, where $y_\phi = 1$ if ϕ is satisfiable; $y_\phi = 0$ otherwise.

5 Evaluation and Analysis

In this section, we conduct a comprehensive evaluation among different approaches for LTL$_f$ satisfiability checking on a large amount of datasets[1].

5.1 Dataset

Synthetic dataset. In order to generate synthetic datasets, we used the *randltl* tool in the SPOT framework[2] to generate random formulae. The tool can generate unique formulae following a specified symbol distribution ($\mathbb{P}$, $\top$, $\bot$, standard logical operators, and temporal logical operators) in a specified size interval. Our symbol distribution sets weights to: $\mathbb{P}$ 2.5, $\top$ 1, $\bot$ 1, $\neg$ 1, $\vee$ 1, X 1, and U 1. The propositions in $\mathbb{P}$ obey the uniform distribution.

We generate synthetic datasets where the formula size is in the interval $[20, 100)$ and the number of different atomic propositions is less than 100 for every formula. The synthetic datasets include a training set with 16K formulae and a validation set with 2K formulae. Besides, we generate formulae and divide them into 6 test sets according to their size intervals: $[20, 100)$, $[100, 120)$, $[120, 140)$, $[140, 160)$, $[160, 180)$, and $[180, 200)$. For each test set, we randomly generate 2K formulae. We keep the balance of the number of satisfiable formulae and that of unsatisfiable formulae.

Following the evaluation of the work [Li *et al.*, 2020], we also evaluate our approaches on large-scale datasets that are available so far, which have been used to evaluate symbolic approaches. These datasets are summarized as follows, where the latter two come from industries and generally have larger sizes than the former two.

***LTL-as-LTL$_f$*.** It consists of 4668 formulae coming from LTL satisfiability checking.

***LTL$_f$-Specific*.** It consists of 1700 formulae generated by common LTL$_f$ patterns.

***NASA-Boeing*.** It consists of 63 real-world LTL$_f$ specifications used in the Boeing AIR 6110 wheelbraking system and the NASA NextGen air traffic control (ATC) system.

***DECLARE*.** It consists of 112 LTL$_f$ patterns widely used in the business process management.

The distribution of *NASA-Boeing* and *DECLARE* and that of the synthetic datasets are different because they are real-world LTL$_f$ specifications.

5.2 Competitor

Our competitors include some neural approaches and some symbolic approaches. The details are shown as follows.

`Transformer`. We follow the work [Hahn *et al.*, 2021]. Specifically, we use Transformer to encode an LTL$_f$ formula and take the embedding of '[CLS]' as the representation of the formula, where '[CLS]' is a special token meaning the beginning of the sentence in Transformer. Then we apply an MLP to the embedding of '[CLS]' to obtain the classification result. We use one-hot embeddings for atomic propositions and trainable embeddings for other tokens.

`RGCN`. We follow the work [Vaezipoor *et al.*, 2021] to exploit R-GCN [Schlichtkrull *et al.*, 2018] to encode LTL$_f$ formulae and then to apply an MLP to obtain the classification result. Every LTL$_f$ formula is represented as a directed graph by first creating the parse tree and then adding self-loops for all the nodes, where each sub-formula is connected to its parent operator via a directed edge. The edges in graph are divided into four types: 1. Self-loops, 2. Unary: the edges from the sub-formula of a unary operator to its parent node, 3. Binary_left (*resp.* 4. Binary_right): the edges from the left (*resp.* right) sub-formula of a binary operator to its parent node. We use one-hot (*resp.* trainable) embeddings for atomic propositions (*resp.* other tokens) as initialization. The graphs are fed into R-GCN to get the embeddings of the root node as the embeddings of LTL$_f$ formulae.

`TreeNN`. We follow TreeNN described in [Allamanis *et al.*, 2017] to obtain the representation of LTL$_f$ formulae, where

[1]Our code and benchmarks are publicly available at https://github.com/wanderer0205/LTLfNet.

[2]SPOT is available in https://spot.lrde.epita.fr/.

the "LOOKUPLEAFEMBEDDING" is set as one-hot embeddings. Different from our method, TreeNN does not apply different 'COMBINE' functions to different operators, thus it does not guarantee permutation invariance or ordering of operators to be kept.

CDLSC. It is a SOTA symbolic approach to LTL$_f$ satisfiability checking [Li *et al.*, 2020]. CDLSC conducts LTL$_f$ satisfiability checking by reducing the problem to a path-search problem over the transition system constructed by iteratively solving certain Boolean SAT problems. Besides, it leverages information produced by SAT solving from both satisfiable and unsatisfiable results to speed up the checking.

nuXmv. It is one of the SOTA approaches to model checking [Cavada *et al.*, 2014]. As LTL satisfiability checking is reducible to model checking, we can compare with nuXmv using the LTL$_f$-to-LTL satisfiability-preserving reduction [Giacomo and Vardi, 2013].

5.3 Setup

We train all compared neural approaches with the Adam optimizer. The hyperparameters of these approaches are as follows. For Transformer, the number of heads is 4, the number of layers is 3, and the dimension of hidden layers is 256 in the encoder layer. The dimension of embeddings d_m is 1024 and the classification layer is an MLP. The batch size is 512, the learning rate is $1e-5$, and the number of training epochs is 1000 with early stopping when the loss does not decrease for 30 epochs. For LTLfNet and TreeNN, the dimension of embeddings d_m is 1024 and the classification layer is an MLP. The batch size is 128, the learning rate is $1e-3$, and the number of training epochs is 256 with early stopping when the loss does not decrease for 30 epochs. For RGCN, the dimension of embeddings d_m is 1024, the dimension of the hidden layer in R-GCN is 32, and the classification layer is a two-layer MLP with the dimension of the hidden layer 128. The batch size is 128, the learning rate is $1e-3$, and the number of training epochs is 256 with early stopping when the loss does not decrease for 30 epochs. The loss function of all these approaches is CrossEntropyLoss and the activation functions for all MLPs are ReLU. All experiments were conducted on a single GPU (NVIDIA A100).

We train all compared approaches on the synthetic datasets and directly test them on datasets with different formula sizes and different distributions. We use the accuracy (acc.), precision (pre.), recall (rec.) and F1 score (F1) of binary classification as the metrics to evaluate the performance.

5.4 Result

Evaluation on Synthetic Datasets

We omit the results of symbolic approaches in the synthetic datasets because they can solve all formulae very quickly. In Table 1, LTLfNet works best. TreeNN and LTLfNet significantly outperform Transformer and RGCN, because the model structures of TreeNN and LTLfNet are the same recursive structures as the parse tree of LTL$_f$ formulae, while Transformer only explicitly represents an ordered sequence of symbols and RGCN considers redundant additional edges. Compared with TreeNN, LTLfNet performs better especially on longer formulae as shown in Figure 2 because LTLfNet keeps the sequentiality of the U operator. It is worth to note that Transformer runs very quickly. Similar results are also reported by Table 2. This is due to the fact that Transformer is implemented by efficient parallel matrix operations.

Model	acc. (%)	pre. (%)	rec. (%)	F1 (%)	time (s)
Transformer	83.95	93.58	72.90	81.96	**5.63**
RGCN	73.85	72.31	77.30	74.72	97.01
TreeNN	94.05	94.54	93.50	94.02	13.83
LTLfNet (our)	**99.25**	**98.91**	**99.60**	**99.25**	14.72

Table 1: Evaluation on the synthetic datasets as the same size of training formulae ([20, 100)), where **boldface** numbers are the best results and the column "time (s)" records the total time (s) to solve all formulae.

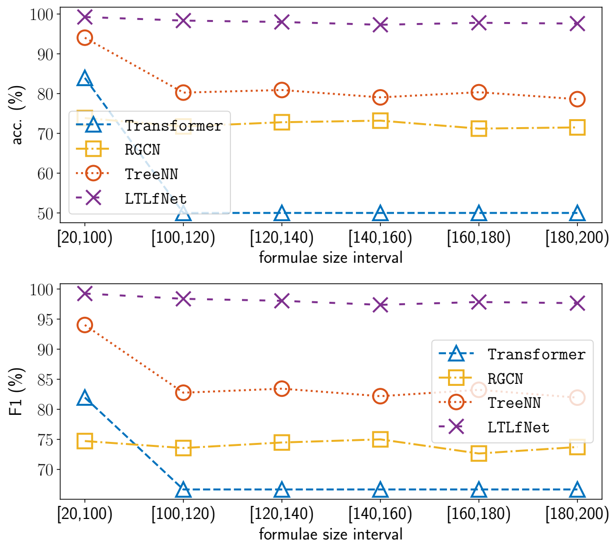

Figure 2: Results of different approaches on test sets with formulae of different sizes.

As shown in Figure 2, the performance of Transformer and TreeNN drops sharply as the formulae become larger while that of RGCN and LTLfNet keeps steady. LTLfNet outperforms other approaches and keeps the high performance even when formulae become larger. It suggests that our approach has scale generalizability to a certain extent.

Evaluation on Large Scale Datasets

The results are shown in Table 2. It is occasional that some neural networks including ours have the same accuracy or F1. These occasional cases occur on relatively small datasets. In these cases, all compared neural approaches result in the same numbers of true positives, false positives, true negatives, and false negatives.

The neural approaches are much faster than the symbolic approaches generally. Particularly, the running time of the symbolic approaches is unacceptable on *LTL-as-LTL$_f$*, *NASA-Boeing*, and *DECLARE*, although they guarantee the correctness. The results on the performance of neural approaches are surprising, such as the results on *NASA-Boeing* and *DECLARE*, which contain only satisfiable formulae. This

Model	*LTL-as-LTL$_f$*			*LTL$_f$-Specific*			*NASA-Boeing*			*DECLARE*		
	acc. (%)	F1 (%)	time (s)	acc. (%)	F1 (%)	time (s)	acc. (%)	F1 (%)	time (s)	acc. (%)	F1 (%)	time (s)
`CDLSC`	100.00	100.00	75,979.65	100.00	100.00	27.47	100.00	100.00	3,604.41	100.00	100.00	60,905.95
`nuXmv`	100.00	100.00	75,560.60	100.00	100.00	2,483.38	100.00	100.00	3,695.72	100.00	100.00	18,096.68
`Transformer`	47.33	62.63	12.98	61.71	61.27	4.25	98.39	99.19	1.34	100.00	100.00	3.71
`RGCN`	39.17	55.16	4,412.20	54.18	70.28	3,309.81	100.00	100.00	216.92	100.00	100.00	6,854.66
`TreeNN`	88.65	93.94	311.08	54.18	70.28	101.42	100.00	100.00	27.50	100.00	100.00	170.78
`LTLfNet` (our)	89.77	94.61	327.25	54.18	70.28	130.93	100.00	100.00	28.70	100.00	100.00	177.36

Table 2: Evaluation on the large-scale datasets.

shows that neural approaches are able to learn biases that are widely present in industrial datasets. `LTLfNet` achieves the best accuracy and F1 score on almost all the four datasets among all compared neural approaches while keeping a decent running time. These results confirm that `LTLfNet` achieves highly confident prediction for LTL$_f$ satisfiability checking in relatively short time.

All neural approaches were trained on the synthetic dataset and then tested in the industrial datasets. This way aims to evaluate the generalization ability of neural networks across distributions, which is useful for industrial instances where the distribution cannot be clarified and the number of data is small. However, we also notice that the generalization ability of neural networks is limited for different distributions of data, *e.g.*, on the *LTL$_f$-Specific* dataset. The development of an effective way to perform LTL$_f$ satisfiability checking across distributions is left in our future work.

6 Discussion

The scale of the sampling formulae. We check the satisfiability of formulae to label the ground truth using exact symbolic reasoning. Due to the high complexity (PSPACE-complete) of LTL$_f$ satisfiability checking, the SOTA symbolic approach is time-consuming to check the formula with a large number of atomic propositions and long sizes, which is confirmed by the experimental results of `CDLSC` on *DECLARE*. In order to generate a large amount of data to train neural networks in an acceptable time, we sample the formulae whose size and the number of atomic propositions are less than 100 in the synthetic datasets.

Sparsity of unsatisfiable formulae. We discover that unsatisfiable formulae are sparse when generating random formulae via the technique described in Section 5.1. Specifically, we sample 50M formulae, of which there are only about 0.4M unsatisfiable formulae. In order to deal with long-tailed distributions, we balance the satisfiable and unsatisfiable instances in the synthetic datasets. Balanced and sufficient sampling of the minority class (unsatisfiable formulae) allows our approach to precisely characterize the boundary of unsatisfiable formulae. Making the boundary of unsatisfiable formulae precise also improves the performance for satisfiable formulae since our problem is a binary classification problem.

LTL satisfiability checking. Although LTL and LTL$_f$ have different semantics, they have the same set of logical constructs, including permutation invariance and sequentiality. Hence, as an approximate approach to LTL$_f$ satisfiability checking, our proposed approach can naturally be applied to LTL satisfiability checking.

Applicability. Our approach is practical as it is able to obtain a highly confident result of LTL$_f$ satisfiability checking in polynomial time. This makes our approach potentially applicable to some LTL$_f$-SAT-heavy tasks, *e.g.*, goal-conflict identification [Degiovanni *et al.*, 2018; Luo *et al.*, 2021]. Our approach can act as a highly confident and efficient pre-identifier, which can not only increase the number of traversed solutions (thanks to the polynomial time complexity) but can also pre-screen candidates that are obviously not valid (thanks to the high confidence). Symbolic approaches can be used in post-processing to ensure soundness.

7 Conclusion and Future Work

This paper is the first to teach an end-to-end neural network to check LTL$_f$ satisfiability. Our main motivation is to explore a new solving paradigm for LTL$_f$ satisfiability checking, a symbolic reasoning problem in PSPACE-complete, to improve the SOTA. Combining symbolic reasoning of LTL$_f$ with continuous neural reasoning is a grand challenge. Our work has established that, by designing the neural architecture (`LTLfNet`) to characterize syntactic features and semantic features (permutation invariance and sequentiality) of LTL$_f$, neural networks can learn to distinguish the satisfiable and unsatisfiable instances in LTL$_f$ and can generalize across large-scale datasets. Moreover, the experimental results show the competitive results of `LTLfNet` compared with the SOTA symbolic approaches. Our work makes it possible to obtain a highly confident result of LTL$_f$ satisfiability checking in polynomial time, which has great implications for some LTL$_f$-SAT-heavy tasks.

Our future work will improve our approach to generalize across distributions, evaluate our approach in LTL satisfiability checking, and extend our approach to generate a trace as evidence of satisfiability.

Acknowledgments

We thank anonymous referees for their helpful comments. This paper was supported by the National Natural Science Foundation of China (No. 61976232 and 61876204), Guangdong Basic and Applied Basic Research Foundation (No.2022A1515011355 and 2020A1515010642), Guizhou Science Support Project (No. 2022-259), Humanities and Social Science Research Project of Ministry of Education (18YJCZH006).

References

[Allamanis *et al.*, 2017] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In *ICML*, volume 70, pages 80–88, 2017.

[Amizadeh *et al.*, 2019] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *ICLR*, 2019.

[Cameron *et al.*, 2020] Chris Cameron, Rex Chen, Jason S. Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *AAAI*, pages 3324–3331, 2020.

[Cavada *et al.*, 2014] 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.

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

[Evans *et al.*, 2018] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *ICLR*, 2018.

[Fionda and Greco, 2016] Valeria Fionda and Gianluigi Greco. The complexity of LTL on finite traces: Hard and easy fragments. In *AAAI*, pages 971–977, 2016.

[Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

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

[Hartford *et al.*, 2018] Jason S. Hartford, Devon R. Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *ICML*, volume 80, pages 1914–1923, 2018.

[Kim *et al.*, 2019] Joseph Kim, Christian Muise, Ankit Shah, Shubham Agarwal, and Julie Shah. Bayesian inference of linear temporal logic specifications for contrastive explanations. In *IJCAI*, pages 5591–5598, 2019.

[Lee *et al.*, 2019] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, volume 97, pages 3744–3753, 2019.

[Li *et al.*, 2014] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. Ltlf satisfiability checking. In *ECAI*, volume 263, pages 513–518, 2014.

[Li *et al.*, 2020] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. Sat-based explicit ltlf satisfiability checking. *Artif. Intell.*, 289:103369, 2020.

[Luo *et al.*, 2021] 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.

[Murphy *et al.*, 2019] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *ICLR*, 2019.

[Paliwal *et al.*, 2020] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *AAAI*, pages 2967–2974, 2020.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[Schlichtkrull *et al.*, 2018] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, volume 10843, pages 593–607, 2018.

[Selsam *et al.*, 2019] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *ICLR*, pages 1–11, 2019.

[Socher *et al.*, 2011] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, pages 151–161, 2011.

[Socher *et al.*, 2013] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642, 2013.

[Vaezipoor *et al.*, 2021] Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. Ltl2action: Generalizing LTL instructions for multi-task RL. In *ICML*, volume 139, pages 10497–10508, 2021.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Xiao *et al.*, 2021] Shengping Xiao, Jianwen Li, Shufang Zhu, Yingying Shi, Geguang Pu, and Moshe Y. Vardi. On-the-fly synthesis for LTL over finite traces. In *AAAI*, pages 6530–6537, 2021.

[Xie *et al.*, 2021] Yaqi Xie, Fan Zhou, and Harold Soh. Embedding symbolic temporal knowledge into deep sequential models. In *ICRA*, pages 4267–4273, 2021.

[Zaheer *et al.*, 2017] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *NeurIPS*, pages 3391–3401, 2017.

[Zhang *et al.*, 2020] Yan Zhang, Jonathon S. Hare, and Adam Prügel-Bennett. Fspool: Learning set representations with featurewise sort pooling. In *ICLR*, 2020.