# End-to-End Learning of LTL$_f$ Formulae by Faithful LTL$_f$ Encoding

**Hai Wan**[1], **Pingjia Liang**[1], **Jianfeng Du**[2,3*], **Weilin Luo**[1], **Rongzhen Ye**[1], **Bo Peng**[1]
[1]School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, P.R.China
[2]Guangdong University of Foreign Studies, Guangzhou 510006, P.R.China
[3]Bigmath Technology, Shenzhen 518063, P.R.China
wanhai@mail.sysu.edu.cn, liangpj3@mail2.sysu.edu.cn, jfdu@gdufs.edu.cn, luowlin3@mail2.sysu.edu.cn, yerzh@mail2.sysu.edu.cn, pengb8@mail2.sysu.edu.cn

## Abstract

It is important to automatically discover the underlying tree-structured formulae from large amounts of data. In this paper, we examine learning linear temporal logic on finite traces (LTL$_f$) formulae, which is a tree structure syntactically and characterizes temporal properties semantically. Its core challenge is to bridge the gap between the concise tree-structured syntax and the complex LTL$_f$ semantics. Besides, the learning quality is endangered by explosion of the search space and wrong search bias guided by imperfect data. We tackle these challenges by proposing an LTL$_f$ encoding method to parameterize a neural network so that the neural computation is able to simulate the inference of LTL$_f$ formulae. We first identify *faithful LTL$_f$ encoding*, a subclass of LTL$_f$ encoding, which has a one-to-one correspondence to LTL$_f$ formulae. Faithful encoding guarantees that the learned parameter assignment of the neural network can directly be interpreted to an LTL$_f$ formula. With such an encoding method, we then propose an end-to-end approach, `TLTLf`, to learn LTL$_f$ formulae through neural networks parameterized by our LTL$_f$ encoding method. Experimental results demonstrate that our approach achieves state-of-the-art performance with up to 7% improvement in accuracy, highlighting the benefits of introducing the faithful LTL$_f$ encoding.

## Introduction

Formulae with tree structures widely exist in data mining and knowledge management, *e.g.*, linear temporal logic (LTL) (Luo et al. 2022) and regular expression (Ye et al. 2023). Therefore, it is important to automatically discover the underlying tree structured formulae from large amounts of data. In this paper, we focus on *linear temporal logic on finite traces (LTL$_f$)* formulae (Baier and McIlraith 2006; Giacomo and Vardi 2013), which are tree-structured formulae characterizing temporal properties. LTL$_f$ has been successfully used in many applications, typically, characterizing the high-level behaviors of a system based on observed traces in planning (Neider and Gavran 2018). For example, LTL$_f$ can specify that an agent keeps walking forward until ($\mathsf{U}$) it encounters a block ($p_1 \mathsf{U} p_2$), where $p_1$ and $p_2$ express "agent walking forward" and "agent encountering a block", respectively. Besides planning, learning LTL$_f$ formulae can also be applied in apprenticeship learning (Kasenberg and Scheutz 2017), behavior classification (Camacho and McIlraith 2019), and explainable artificial intelligence (Kim et al. 2019).

Learning LTL$_f$ formulae is a challenging problem. The core challenge is to bridge the gap between the concise tree-structured syntax and the complex LTL$_f$ semantics. Besides, a formula with *arbitrary form* has only semantic constraints and no syntactic constraints. Finding such a formula probably leads to explosion of the search space. On the other hand, *imperfect data*, *e.g.*, incorrect labels (Kim et al. 2019), tend to introduce wrong search bias far away from the target formula.

To tackle the above challenges, some approaches (Camacho and McIlraith 2019; Luo et al. 2022) modeled the relation between syntax and semantics by introducing an intermediate representation, but they need to pay additional costs from learning the over-expressive intermediate representation. There are also some approaches (Neider and Gavran 2018; Gaglione et al. 2021) that uniformly encoded syntactic and semantic constraints as Boolean satisfiability problem (SAT) or maximum satisfiability problem (MaxSAT), but they suffer from the high computational complexity of SAT or MaxSAT. Other approaches (Lemieux, Park, and Beschastnikh 2015; Shah et al. 2018; Kim et al. 2019) directly encoded syntactic constraints, exploiting well-designed heuristics to search for formulae satisfying semantic constraints. However, the heuristics are limited in scope and are time-consuming. In confront of imperfect data, some approaches (Neider and Gavran 2018; Camacho and McIlraith 2019) directly ignored them, and others (Lemieux, Park, and Beschastnikh 2015; Shah et al. 2018; Kim et al. 2019) limited the hypothesis space by templates.

In this paper, we propose an LTL$_f$ encoding method for faithfully bridging the neural network inference and the LTL$_f$ inference. An LTL$_f$ encoding is a parameter assignment of the target neural network ensuring that, under certain simple conditions, an LTL$_f$ formula is able to be directly interpreted from the neural network. We identify the conditions where LTL$_f$ encodings and LTL$_f$ formulae are one-to-one corresponding, and refer to the encoding under these conditions as *faithful LTL$_f$ encoding*, which provides a compact encoding of the LTL$_f$ syntax. We model the LTL$_f$ semantics via designing an inference process with LTL$_f$ encoding; in other words, inference with LTL$_f$ encoding simulates the inference of LTL$_f$ formulae. Based on this theoretical re-

*Corresponding author

sult, we propose an end-to-end approach, `TLTLf`, to learn LTL$_f$ formulae. Specifically, `TLTLf` learns a neural network that is parameterized by the LTL$_f$ encoding method, and subsequently interprets an LTL$_f$ formula from the learned parameter assignment of the neural network.

Following the assessment of the state-of-the-art (SOTA) approach (Luo et al. 2022), we evaluate `TLTLf` across datasets with and without imperfect data. Experimental results show that `TLTLf` achieves SOTA performance and improves the accuracy by up to 7% compared with the SOTA approach. Besides, the LTL$_f$ formula learnt by `TLTLf` is highly faithful to the network from which it is extracted, giving the same satisfiability results over 95% traces. Following the work (Kim et al. 2019; Luo et al. 2022), we also evaluate `TLTLf` on imperfect data generated by assigning incorrect labels to a subset of the original data. Results demonstrate that `TLTLf` still outperforms other approaches on imperfect data.

## Related Work

**Learning LTL formulae from positive and negative traces.** Neider and Gavran (2018) provided two methods to learn LTL formulae, one based on SAT and the other based on the decision tree. Besides, Camacho and McIlraith (2019) designed a SAT-based method to learn formulae from the symbolic state representation of LTL formulae. However, both studies are limited by an assumption that the training data are precise. Kim et al. (2019) handled imperfect data based on Bayesian inference. Their use of templates limits the hypothetical space of formulae, preventing the proposed method from learning arbitrary formulae. Overall, these approaches cannot handle imperfect data and learn arbitrary formulae simultaneously.

Recently, several studies have realized the importance of learning arbitrary LTL$_f$ formulae from imperfect data. Gaglione et al. (2021) proposed a MaxSAT-based approach to tolerate imperfect data, but the scalability is limited by the computational complexity of the MaxSAT solver. The work (Luo et al. 2022) is the closest to this work. Based on a theoretical result that the graph neural network (GNN) inference is able to simulate the LTL$_f$ inference in checking if a trace is satisfied, Luo et al. (2022) first trained a GNN classifier for the trace satisfiability problem and then interpreted LTL$_f$ formulae from the model parameters. However, they do not guarantee the *faithfulness* between the GNN inference and the LTL$_f$ inference, *i.e.*, the result of the interpreted LTL$_f$ formula is probably different from that of the GNN classifier in checking if a trace is satisfied. This is because they only proved that, given a formula, a GNN classifier can always be constructed to simulate LTL$_f$ inference, but not vice versa, which means that the expressive power of a GNN classifier is stronger than an LTL$_f$ formula. It results in additional costs from learning the over-expressive GNN classifier. In contrast, the interpreted LTL$_f$ formula in this work is more faithful with the learnt neural network.

**Learning temporal logic formulae from other forms of data.** Some work learns temporal logic formulae from other forms of data, such as Markov decision process (Kasenberg and Scheutz 2017) and only positive traces (la Rosa and McIlraith 2011; Lemieux, Park, and Beschastnikh 2015; Le and Lo 2015; Shah et al. 2018; Cao et al. 2018). In the business process modeling, some studies (Maggi, Bose, and van der Aalst 2012; Ciccio, Mecella, and Mendling 2013; Maggi et al. 2018; Leno et al. 2020) use temporal logic to model the declarative process model learned from the event log, which is viewed as a kind of positive traces.

**Learning other formal languages.** Our work is also relevant to learning logical representations, such as regular language (Angluin 1987; Giantamidis and Tripakis 2016; Bollig et al. 2009; Angluin, Eisenstat, and Fisman 2015; Fiterau-Brostean et al. 2017; Gabel and Su 2008, 2010; Ye et al. 2023) and other forms of temporal formulae (Asarin et al. 2011; Jin et al. 2015; Bombara et al. 2016; Xu et al. 2016; Xu and Julius 2016; Kong, Jones, and Belta 2016; Arif et al. 2020; Brunello, Sciavicco, and Stan 2020; Roy, Fisman, and Neider 2020; Xu et al. 2019; Maggi, Montali, and Peñaloza 2020). Although the expressive power of temporal formulae is weaker than that of regular languages, learning temporal formulae is more compact and easy to understand (Camacho and McIlraith 2019). Our approach will potentially be extended to learn formulae in other formal languages.

**Learning tree structure.** There have been various studies on learning tree structure. They can be divided into two main categories. One is to extract a latent tree structure for each example to improve the performance. For example, in nature language processing (NLP), the syntactic dependency tree of the sentence is extracted to improve the performance on tasks like semantic role labeling (Shi, Malioutov, and Irsoy 2020; Zhang et al. 2022) and sentiment analysis (Li et al. 2021; Hou et al. 2021). Syntactic dependency parsing is a classical problem in NLP and there have been some mature tools like the natural language toolkit (Bird, Klein, and Loper 2009). For the neural approach, Chen and Manning (2014) used three types of operations of a stack and a queue to construct the syntactic dependency tree and used a neural network to predict the operation in each construction step. The other category is to design an approach to learning the tree structure of specific semantics. Fargier, Gimenez, and Mengin (2018) proposed a greedy learning algorithm to learn lexicographic preference trees from positive examples. Sun et al. (2020) and Xie et al. (2021) used a tree-structured decoder to generate the abstract syntax tree of code from the natural language description. Ye et al. (2023) used a differentiable approach to learn the syntax tree of regular expressions from both positive and negative examples. Compared with the above studies, our approach learns the tree-structured LTL$_f$ formulae with a completely different tree structure from both positive and negative examples.

## Preliminaries

**Linear temporal logic.** *Linear temporal logic (LTL)* (Pnueli 1977) is a modal logic typically used to express temporally extended constraints over state trajectories. Derived from LTL, LTL interpreted over finite traces (Baier and McIlraith 2006; Giacomo and Vardi 2013) is referred to as *finite LTL$_f$*. The syntax of LTL$_f$ for a finite set of atomic propositions $\mathbb{P}$ is built up of $\top$ (true) and $\bot$ (false), standard logical operators ($\wedge$ and $\neg$), and temporal logical operators *next* ($\mathsf{X}$) and *until*

(U), described as follows:

$$\phi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2,$$

where $p \in \mathbb{P} \cup \{\top, \bot\}$ and $\varphi$, $\varphi_1$, and $\varphi_2$ are LTL$_f$ formulae.

The other operators *or* ($\vee$), *weak next* (N), *release* (R), *eventually* (F), and *always* (G) are commonly used, but they can be defined using the above fundamental operators, *i.e.*, $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathsf{N}\varphi := \neg\mathsf{X}\top \vee \mathsf{X}\varphi$, $\varphi_1 \mathsf{R} \varphi_2 := \neg(\neg\varphi_1 \mathsf{U} \neg\varphi_2)$, $\mathsf{F}\varphi := \top\mathsf{U}\varphi$, and $\mathsf{G}\varphi := \neg(\top\mathsf{U}\neg\varphi)$. For brevity, we only consider the fundamental operators in this paper, but our approach can be extended to handle all operators of LTL$_f$.

The prefix form of an LTL$_f$ formula $\phi$, denoted by $\mathrm{pre}(\phi)$, is defined recursively as follows:

$$\mathrm{pre}(\phi) = \begin{cases} p, & \phi = p \in \mathbb{P}, \\ \alpha\mathrm{pre}(\phi_1), & \phi = \alpha\phi_1, \alpha \in \{\neg, \mathsf{X}\}, \\ \beta\mathrm{pre}(\phi_1)\mathrm{pre}(\phi_2), & \phi = \phi_1\beta\phi_2, \beta \in \{\wedge, \mathsf{U}\}. \end{cases} \tag{1}$$

The syntax tree of an LTL$_f$ formula $\phi$, denoted by $T(\phi)$, is defined as follows:

- if $\phi = p$, then $T(\phi)$ is a tree with a single node $v_p$.
- if $\phi = \beta\phi_i$, where $\beta \in \{\neg, \mathsf{X}\}$, then the root node of $T(\phi)$ is $v_\beta$ and it has a unique subtree $T(\phi_i)$.
- if $\phi = \phi_i\beta\phi_j$, where $\beta \in \{\wedge, \mathsf{U}\}$, then the root node of $T(\phi)$ is $v_\beta$ and it has two subtrees, where the left subtree is $T(\phi_i)$ and the right subtree is $T(\phi_j)$.

The node sequence obtained by preorder traversal of all nodes in an LTL$_f$ syntax tree $T(\phi)$, denoted by $\mathrm{pretravel}(T(\phi))$, is defined as follows:

$$\mathrm{pretravel}(T(\phi)) = \begin{cases} v_p, & \phi = p \in \mathbb{P}. \\ v_\alpha, \mathrm{pretravel}(T(\phi_1)), & \phi = \alpha\phi_1, \alpha \in \{\neg, \mathsf{X}\}. \\ v_\beta, \mathrm{pretravel}(T(\phi_1)), & \\ \mathrm{pretravel}(T(\phi_2)), & \phi = \phi_1\beta\phi_2, \beta \in \{\wedge, \mathsf{U}\}. \end{cases} \tag{2}$$

LTL$_f$ formulae are interpreted over finite *traces* of propositional states. A finite trace is represented by $\pi = s_0, s_1, \ldots, s_n$, where $s_t \in 2^{\mathbb{P}}$ is a state at time $t$. For every state $s_i$ of $\pi$ 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 $\pi$ is the number of states of $\pi$, denoted by $|\pi|$. By $\pi_i$ we denote a *sub-trace* of $\pi$ beginning from the state $s_i$. Let $\pi$ be 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 $\phi, \phi_1, \phi_2$ are LTL$_f$ formulae, and $p \in \mathbb{P} \cup \{\top, \bot\}$.

**Learning LTL$_f$ formulae.** We focus on the problem of learning a *target LTL$_f$ formula* to characterize the behaviors observed in a set of *positive traces* ($\Pi^+$), while to exclude behaviors observed in a set of *negative traces* ($\Pi^-$). The target formula is an *LTL$_f$ classifier over traces* aiming to separate the provided positive and negative traces. We denote $\Pi = \Pi^+ \cup \Pi^-$. We define $\mathsf{lab}\colon \Pi \to \{0, 1\}$: for any $e \in \Pi$, $\mathsf{lab}(e) = 1$ if $e \in \Pi^+$, or $\mathsf{lab}(e) = 0$ if $e \in \Pi^-$. The *accuracy* of an LTL$_f$ formula $\phi$ for $\Pi$ is defined as:

$$\mathsf{acc}(\phi, \Pi) = \frac{|\{\pi \in \Pi^+ | \pi \models \phi\}| + |\{\pi \in \Pi^- | \pi \not\models \phi\}|}{|\Pi|}. \tag{3}$$

## LTL$_f$ Encoding and Inference

The key part of `TLTLf` is a neural network parameterized by the LTL$_f$ encoding method. So in this section, we first show the definition of LTL$_f$ encoding to parameterize `TLTLf`, as well as the model structure of `TLTLf` for inference of the satisfaction relation over traces. Afterwards, we introduce a subclass of LTL$_f$ encoding, namely *faithful LTL$_f$ encoding*, and show a one-to-one correspondence between LTL$_f$ formulae and faithful LTL$_f$ encodings.

### Model Structure of `TLTLf`

The input of `TLTLf` is a trace and its output is the satisfaction relation between `TLTLf` and a trace. The trainable parameters of `TLTLf` are given by Definition 1.

**Definition 1** *Let $\mathbb{P}$ be a set of atomic propositions and $L \in \mathbb{N}$. The parameter set of `TLTLf` of size $L$ is defined as $\Gamma = \{(\Gamma_{\mathrm{right}})_{i,j} \in \mathbb{R} | 1 \leq i \leq L-2, i+2 \leq j \leq L\} \cup \{(\Gamma_{\mathrm{atom}})_{i,j} \in \mathbb{R} | 1 \leq i \leq L, 1 \leq j \leq |\mathbb{P}|\} \cup \{(\Gamma_\neg)_i, (\Gamma_\wedge)_i, (\Gamma_\mathsf{X})_i, (\Gamma_\mathsf{U})_i, (\Gamma_{\mathrm{none}})_i \in \mathbb{R} | 1 \leq i \leq L\}$. For brevity, we also reuse $\Gamma$ to denote an assignment of the parameter set $\Gamma$ of `TLTLf`.*

To establish a relationship between parameters of `TLTLf` and an LTL$_f$ syntax tree, we confine that every parameter in $\Gamma$ be assigned a value between 0 and 1. We call a parameter assignment with this restriction an LTL$_f$ encoding, formally defined below.

**Definition 2** *An LTL$_f$ encoding of `TLTLf` of size $L$ is defined as $\theta = \{(\theta_{\mathrm{right}})_{i,j} \in \mathbb{R}^{(0,1)} | 1 \leq i \leq L-2, i+2 \leq j \leq L\} \cup \{(\theta_{\mathrm{atom}})_{i,j} \in \mathbb{R}^{(0,1)} | 1 \leq i \leq L, 1 \leq j \leq |\mathbb{P}|\} \cup \{(\theta_\neg)_i, (\theta_\wedge)_i, (\theta_\mathsf{X})_i, (\theta_\mathsf{U})_i, (\theta_{\mathrm{none}})_i \in \mathbb{R}^{(0,1)} | 1 \leq i \leq L\}$, where $\mathbb{R}^{(0,1)}$ denotes the real value range from 0 to 1.*

An LTL$_f$ encoding is able to represent an LTL$_f$ syntax tree. Let $T(\phi)$ be an LTL$_f$ syntax tree and $\mathrm{pretravel}(T(\phi)) = v_1, v_2, ..., v_L$. For all $i \in [1, L]$, the parameters $\theta_\neg$, $\theta_\wedge$, $\theta_\mathsf{X}$, $\theta_\mathsf{U}$ and $\theta_{\mathrm{atom}}$ determine whether $v_i$ is $v_\neg, v_\wedge, v_\mathsf{X}, v_\mathsf{U}$ and $v_p$ for $p \in \mathbb{P}$, respectively. We introduce $\theta_{\mathrm{none}}$ to encode a null node so as to represent a smaller syntax tree using an LTL$_f$ encoding of a greater size. For example, if we want to use an LTL$_f$ encoding of size $L+1$ to represent $T(\phi)$, we should set $(\theta_{\mathrm{none}})_{L+1}$ to 1. The parameter $(\theta_{\mathrm{right}})_{i,j}$ is used to represent whether the right child of $v_i$ is $v_j$. Since $\mathrm{pretravel}(T(\phi))$ is the prefix form of $\phi$, the left or the only child of $v_i$ should be $v_{i+1}$ if $v_i$ has children. We use the following Example 1 to illustrate an LTL$_f$ encoding.

**Example 1** *Let $\mathbb{P} = \{p_1, p_2\}$ and $\theta$ be an LTL$_f$ encoding of `TLTLf` of size 3 where $(\theta_\neg)_1 = 0.8, (\theta_\mathsf{X})_1 = 0.3, (\theta_{\mathrm{atom}})_{2,1} = (\theta_{\mathrm{none}})_3 = 1$ and other parameters are* *assigned* 0. *The LTL$_f$ formula that $\theta$ represents is the most likely to be $\neg p_1$ while it may also be $\mathsf{X}p_1$ since $(\theta_\mathsf{X})_1 = 0.3$.*

In fact, an arbitrary parameter assignment of `TLTLf` can be converted to an LTL$_f$ encoding of `TLTLf` of the same size, as shown in the following equation.

$$\begin{aligned}
&(\theta_{\text{right}})_{i,j} = \frac{e^{(\Gamma_{\text{right}})_{i,j}}}{(\eta_{\text{right}})_i}, (\theta_{\text{atom}})_{i,j} = \frac{e^{(\Gamma_{\text{atom}})_{i,j}}}{(\eta_{\text{op}})_i},\\
&(\theta_\neg)_i = \frac{e^{(\Gamma_\neg)_i}}{(\eta_{\text{op}})_i}, (\theta_\wedge)_i = \frac{e^{(\Gamma_\wedge)_i}}{(\eta_{\text{op}})_i},\\
&(\theta_\mathsf{X})_i = \frac{e^{(\Gamma_\mathsf{X})_i}}{(\eta_{\text{op}})_i}, (\theta_\mathsf{U})_i = \frac{e^{(\Gamma_\mathsf{U})_i}}{(\eta_{\text{op}})_i}, (\theta_{\text{none}})_i = \frac{e^{(\Gamma_{\text{none}})_i}}{(\eta_{\text{op}})_i},
\end{aligned} \tag{4}$$

where

$$\begin{aligned}
(\eta_{\text{right}})_i \quad &= \textstyle\sum_{j=i+2}^{L} e^{(\Gamma_{\text{right}})_{i,j}} + \sum_{j=1}^{|\mathbb{P}|} e^{(\Gamma_{\text{atom}})_{i,j}}\\
&\quad + e^{(\Gamma_{\text{none}})_i} + e^{(\Gamma_\neg)_i} + e^{(\Gamma_\mathsf{X})_i},\\
(\eta_{\text{op}})_i \quad &= \textstyle\sum_{j=1}^{|\mathbb{P}|} e^{(\Gamma_{\text{atom}})_{i,j}} + e^{(\Gamma_{\text{none}})_i}\\
&\quad + e^{(\Gamma_\neg)_i} + e^{(\Gamma_\wedge)_i} + e^{(\Gamma_\mathsf{X})_i} + e^{(\Gamma_\mathsf{U})_i}.
\end{aligned} \tag{5}$$

Thus, `TLTLf` is defined as a neural network parameterized by Definition 1, which is then transformed to an LTL$_f$ encoding by Equation (4). Given a trace $\pi$, `TLTLf` computes a satisfaction relation between the LTL$_f$ encoding of `TLTLf` and the trace. Such an inference process of `TLTLf` is formalized in Definition 3.

**Definition 3** *Let $\mathbb{P}$ be a set of atomic propositions and $\Gamma$ a parameter assignment of `TLTLf` of size $L \in \mathbb{N}$. $\theta$ is constructed from $\Gamma$ by Equation* (4). *Given a trace $\pi = s_0, s_1, ..., s_n$ over $\mathbb{P}$, `TLTLf` computes satisfaction vectors $x_i \in \mathbb{R}^L$ (where $0 \le i \le n$) defined as follows:*

$$\begin{aligned}
&(x_i)_j = \sigma(\textstyle\sum_{k=1}^{|\mathbb{P}|}(\theta_{\text{atom}})_{j,k} I(p_k \in s_i) + (\theta_\mathsf{X})_j (x_{i+1})_{j+1}\\
&+(\theta_\neg)_j \sigma(1-(x_i)_{j+1}) + (\theta_\wedge)_j \sigma((x_i)_{j+1} + (r_i)_j - 1)\\
&+(\theta_\mathsf{U})_j((r_i)_j + \sigma((x_i)_{j+1} + (x_{i+1})_j - 1))),
\end{aligned} \tag{6}$$

*where*

$$\begin{aligned}
(r_i)_j \quad &= \textstyle\sum_{k=L}^{j+2}(\theta_{\text{right}})_{j,k}(x_i)_k,\\
\sigma(x) \quad &= \min(1, \max(0, x)),
\end{aligned} \tag{7}$$

*and $(x_i)_{L+1} = 0$, $(x_{n+1})_k = 0$ for all $1 \le k \le L+1$, and $I(C)$ returns 1 if $C$ is satisfied or 0 otherwise. By* $\mathrm{ESat}(\theta, \pi)$ *we denote the satisfaction relation between $\theta$ and $\pi$. Finally `TLTLf` outputs* $\mathrm{ESat}(\theta, \pi)$ *as* $(x_0)_1$.

Intuitively, $(x_i)_j$ represents whether $\pi_i = s_i, s_{i+1}, ..., s_n$ satisfies the $j$-th sub-formula of the formula that $\theta$ represents. The computation of $(x_i)_j$ is based on the definition of the satisfaction relation of LTL$_f$. Specifically, we first calculate $(r_i)_j$, which denotes whether the right sub-formula of the $j$-th sub-formula satisfies $\pi_i$. Here $\theta_{\text{right}}$ plays a role of right-child selection. In Equation (6), $\theta_\neg, \theta_\wedge, \theta_\mathsf{X}, \theta_\mathsf{U}$ are used to select operators while $\theta_{\text{atom}}$ is used to select atomic propositions. For example, if the $j$-th sub-formula is of the form of $\neg\phi_{j+1}$, then the equation can be simplified to $(x_i)_j = \sigma(1-(x_i)_{j+1})$, which means that the $j$-th sub-formula satisfies $\pi_i$ if its only child sub-formula does not satisfy $\pi_i$. We use the following Example 2 to illustrate Definition 3.

**Example 2** *Let $\pi = \{p_1, p_2\}, \{p_2\}$ and $\theta$ be the LTL$_f$ encoding of `TLTLf` given in Example 1. Then the satisfaction vector is $x_0 = [0, 1, 0], x_1 = [0.8, 0, 0]$. The inference output is* $\mathrm{ESat}(\theta, \pi) = (x_0)_1 = 0$.

## The Faithful Subclass of LTL$_f$ Encoding

As shown in Example 1, not every LTL$_f$ encoding corresponds to an LTL$_f$ formula. So in the following we introduce the notion of *faithful LTL$_f$ encoding* by adding additional constraints to ensure the correspondence.

**Definition 4** *Let $\theta$ be an LTL$_f$ encoding of `TLTLf` of size $L$. $\theta$ is said to be* faithful *if it satisfies the following conditions:*

1. $\forall\gamma \in \theta : \gamma = 0 \vee \gamma = 1$.
2. $\forall i \in [1, L] : (\theta_{\text{none}})_i + \sum_{j=1}^{|\mathbb{P}|}(\theta_{\text{atom}})_{i,j} + (\theta_\neg)_i + (\theta_\wedge)_i + (\theta_\mathsf{X})_i + (\theta_\mathsf{U})_i = 1$.
3. $\sum_{j=1}^{|\mathbb{P}|}(\theta_{\text{atom}})_{L,j} + (\theta_{\text{none}})_L = 1 \wedge \forall i \in [1, L-1] : \sum_{j=i+2}^{L}(\theta_{\text{right}})_{i,j} + (\theta_{\text{none}})_i + \sum_{j=1}^{|\mathbb{P}|}(\theta_{\text{atom}})_{i,j} + (\theta_\neg)_i + (\theta_\mathsf{X})_i = 1$.
4. $(\theta_{\text{none}})_1 = 0 \wedge \forall i \in [2, L] : \sum_{j=1}^{i-2}(\theta_{\text{right}})_{j,i} + (\theta_\neg)_{i-1} + (\theta_\wedge)_{i-1} + (\theta_\mathsf{X})_{i-1} + (\theta_\mathsf{U})_{i-1} + (\theta_{\text{none}})_i = 1$.
5. $\forall i \in [1, L-1] : (\theta_{\text{none}})_{i+1} \ge (\theta_{\text{none}})_i$.
6. $\forall i \in [1, L), \forall j \in (i, L], \forall t \in (i, j), \forall t' \in (j, L] : (\theta_{\text{right}})_{i,j} + (\theta_{\text{right}})_{t,t'} \le 1$.

Intuitively, Condition 1 ensures all the parameters are assigned Boolean values. Condition 2 forces each node on the syntax tree to represent only one operator or one atomic proposition. Condition 3 restricts the number of children of a node according to the operator. If this node represents a binary operator, it will not represent other operators or atomic propositions because of Condition 2. Thus this node will have exactly one right child according to Condition 3. Besides, the node $v_L$ has no child node so it must represent an atomic proposition or must not be used. Condition 4 restricts the number of fathers of a node. Except for the root node that has no father, every valid node $v_i$ (where $(\theta_{\text{none}})_i = 0$) has exactly one father node. Condition 5 is used to restrict that null nodes $v_i$ (where $(\theta_{\text{none}})_i = 1$) are placed continuously at the end of the node sequence obtained by preorder traversal of all nodes. Condition 6 restricts that, if $v_j$ is the right child of $v_i$, then every node $v_t$ between them does not have a right child $v'_t$ such that $t' > j$.

The faithful LTL$_f$ encoding ensures the LTL$_f$ formula that it represents to be unique, as shown in the following example.

**Example 3** *Consider the LTL$_f$ encoding $\theta$ of `TLTLf` given in Example 1 again. A faithful LTL$_f$ encoding closed to $\theta$ is $\hat{\theta}$, where $(\hat{\theta}_\neg)_1 = 1, (\hat{\theta}_\mathsf{X})_1 = 0, (\hat{\theta}_{\text{atom}})_{2,1} = 1, (\hat{\theta}_{\text{none}})_3 = 1$ and other parameters are assigned* 0. *The LTL$_f$ formula that $\hat{\theta}$ represents is unique, which is $\neg p_1$.*

## Faithful LTL$_f$ Encoding vs LTL$_f$ Formula

For an arbitrary LTL$_f$ formula $\phi$, we introduce a function to encode $\phi$ into a parameter assignment of `TLTLf` of an equal or greater size, formalized in the following Definition 5.

**Definition 5** *Let $\phi$ be an LTL$_f$ formula, $T(\phi)$ its syntax tree, and* $\mathrm{pretravel}(T(\phi)) = v_1, v_2, ..., v_L$. *The function for encoding $\phi$ into a parameter assignment of `TLTLf` of size $L' \ge L$, denoted by $\theta_{\phi(L')}$, is defined as follows:*

- $\forall 1 \le i \le L : (\theta_{\text{right}})_{i,j} = 1$ *if $v_j$ is the right child of $v_i$ and $(\theta_{\text{right}})_{i,j} = 0$ otherwise.*

- $\forall 1 \leq i \leq L$ : $(\theta_{\text{atom}})_{i,j} = 1$ *if* $v_i = v_{p_j}$ *and* $(\theta_{\text{atom}})_{i,j} = 0$ *otherwise.*
- $\forall 1 \leq i \leq L$ : $(\theta_\beta)_i = 1$ *if* $v_i = v_\beta$ *and* $(\theta_\beta)_i = 0$ *otherwise, where* $\beta \in \{\neg, \wedge, \mathsf{X}, \mathsf{U}\}$.
- $\forall L < i \leq L'$ : $(\theta_{\text{none}})_i = 1, (\theta_{\text{right}})_{i,j} = 0, (\theta_{\text{atom}})_{i,j} = 0, (\theta_\beta)_i = 0$, *where* $\beta \in \{\neg, \wedge, \mathsf{X}, \mathsf{U}\}$.

**Example 4** *Let* $\phi$ *be* $p_1 \mathsf{U} \mathsf{X} p_2$. *We have* $\text{pretravel}(T(\phi)) = v_{\mathsf{U}}, v_{p_1}, v_{\mathsf{X}}, v_{p_2}$. *Then in the encoding* $\theta_{\phi(5)}$, *we have* $(\theta_{\mathsf{U}})_1 = 1$, $(\theta_{\text{atom}})_{2,1} = 1$, $(\theta_{\mathsf{X}})_3 = 1$, $(\theta_{\text{atom}})_{4,2} = 1$, $(\theta_{\text{right}})_{1,3} = 1$, $(\theta_{\text{none}})_5 = 1$, *and other parameters are assigned 0.*

The following lemma shows that the parameter assignment of `TLTLf` encoded from an LTL$_f$ formula must be a faithful LTL$_f$ encoding.[1]

**Lemma 1** *Let* $\phi$ *be an LTL$_f$ formula, then* $\theta_{\phi(L')}$ *is a faithful LTL$_f$ encoding of* `TLTLf` *of size* $L'$.

The following Definition 6 shows how to decode a faithful LTL$_f$ encoding to a symbol sequence. Then Example 5 illustrates the decoding method.

**Definition 6** *Let* $\theta$ *be a faithful LTL$_f$ encoding of* `TLTLf` *of size L. The decoding function* $\text{decode}(\theta) = o_1 \dots o_L$ *is defined as follows:*

$$o_i = \begin{cases} p_j, & (\theta_{\text{atom}})_{i,j} = 1, \\ \beta, & (\theta_\beta)_i = 1, \beta \in \{\neg, \wedge, \mathsf{X}, \mathsf{U}\}, \\ \epsilon, & (\theta_{\text{none}})_i = 1. \end{cases} \tag{8}$$

**Example 5** *Consider* $\phi$ *and* $\theta_{\phi(5)}$ *in Example 4 again. Since* $(\theta_{\mathsf{U}})_1 = 1, (\theta_{\text{atom}})_{2,1} = 1, (\theta_{\mathsf{X}})_3 = 1, (\theta_{\text{atom}})_{4,2} = 1$ *and* $(\theta_{\text{none}})_5 = 1$, *We have* $\text{decode}(\theta_{\phi(5)}) = \mathsf{U} p_1 \mathsf{X} p_2$, *which is exactly the prefix form of* $\phi$.

The following Theorem 1 shows that the decoding method always results in the prefix form of an LTL$_f$ formula.

**Theorem 1** *For every faithful LTL$_f$ encoding* $\theta$ *of* `TLTLf`, $\text{decode}(\theta)$ *is the prefix form of a certain LTL$_f$ formula.*

Now we can prove that faithful LTL$_f$ encodings and the prefix forms of LTL$_f$ formulae have one-to-one correspondence, by showing that the decoding method is both subjective (Theorem 2) and injective (Theorem 3).

**Theorem 2** *For any LTL$_f$ formula* $\phi$ *with* $\text{pretravel}(T(\phi)) = v_1, v_2, ..., v_L$ *and any* $L' \geq L$, *there exists a faithful LTL$_f$ encoding* $\theta$ *of size* $L'$ *such that* $\text{decode}(\theta) = \text{pre}(\phi)$.

**Theorem 3** *Given two different faithful LTL$_f$ encodings of the same size, namely* $\theta_1$ *and* $\theta_2$, $\text{decode}(\theta_1) \neq \text{decode}(\theta_2)$.

## Inference for the Satisfaction Relation

Now we achieve Theorem 4 to guarantee an equivalence between the inference of `TLTLf` and the inference of LTL$_f$.

**Theorem 4** *For any LTL$_f$ formula* $\phi$ *with* $\text{pretravel}(T(\phi)) = v_1, v_2, ..., v_L$, *any* $L' \geq L$ *and any trace* $\pi = s_0, s_1, ..., s_n$, *it holds that* $\text{ESat}(\theta_{\phi(L')}, \pi) = 1$ *if* $\pi \models \phi$, *or* $\text{ESat}(\theta_{\phi(L')}, \pi) = 0$ *otherwise.*

[1]All the proofs of lemmas/theorems are provided in the technical report available at https://github.com/a79461378945/TLTLf.git.

## Learning LTL$_f$ Formulae by LTL$_f$ Encoding

For learning LTL$_f$ formulae, we first build `TLTLf` parameterized by an LTL$_f$ encoding and then train it to distinguish positive traces from negative traces. Afterwards, we give an algorithm to extract the formula from `TLTLf`.

The faithful conditions are used in both the network structure and the optimization objective. The construction process of $\theta$ is similar to using the softmax function. We use it to make $\theta$ satisfy Condition 2 and Condition 3 as much as possible. Notice that $\theta$ satisfies Condition 2 because $\forall i \in [1, L] (\theta_{\text{none}})_i + \sum_{j=1}^{|\mathbb{P}|} (\theta_{\text{atom}})_{i,j} + (\theta_\neg)_i + (\theta_\wedge)_i + (\theta_{\mathsf{X}})_i + (\theta_{\mathsf{U}})_i = \frac{e^{(\Gamma_{\text{none}})_i}}{(\eta_{\text{op}})_i} + \sum_{j=1}^{|\mathbb{P}|} \frac{e^{(\Gamma_{\text{atom}})_{i,j}}}{(\eta_{\text{op}})_i} + \frac{e^{(\Gamma_\neg)_i}}{(\eta_{\text{op}})_i} + \frac{e^{(\Gamma_\wedge)_i}}{(\eta_{\text{op}})_i} + \frac{e^{(\Gamma_{\mathsf{X}})_i}}{(\eta_{\text{op}})_i} + \frac{e^{(\Gamma_{\mathsf{U}})_i}}{(\eta_{\text{op}})_i} = 1$. Similarly, Condition 3 in Definition 4 is approximately satisfied.

For each trace $\pi$ in the set of positive traces $\Pi^+$ and the set of negative traces $\Pi^-$, we use the LTL$_f$ encoding $\theta$ to infer the satisfaction relation. The classification objective is:

$$\zeta_1 = \sum_{\pi \in \Pi} (\text{ESat}(\theta, \pi) - \mathsf{lab}(\pi))^2, \tag{9}$$

where $\forall \pi \in \Pi^+, \mathsf{lab}(\pi) = 1$ and $\forall \pi \in \Pi^-, \mathsf{lab}(\pi) = 0$.

We additionally use regularization loss to make $\theta$ approximately satisfy Condition 4, 5, and 6 in Definition 4. The regularization terms are formulated as:

$$\begin{aligned} \zeta_2 = &\sum_{i=2}^{L} (\sum_{j=1}^{i-2} (\theta_{\text{right}})_{j,i} + (\theta_\neg)_{i-1} + (\theta_\wedge)_{i-1} + (\theta_{\mathsf{X}})_{i-1} \\ &+ (\theta_{\mathsf{U}})_{i-1} + (\theta_{\text{none}})_i - 1)^2, \\ \zeta_3 = &\sum_{i=1}^{L-1} \mathsf{Relu}((\theta_{\text{none}})_i - (\theta_{\text{none}})_{i+1}), \\ \zeta_4 = &\sum_{i=1}^{L-2} \sum_{j=i+2}^{L} \sum_{t=i+1}^{j-1} \sum_{t'=j+1}^{L} \mathsf{Relu}((\theta_{\text{right}})_{i,j} \\ &+ (\theta_{\text{right}})_{t,t'} - 1). \end{aligned} \tag{10}$$

They are obtained from the corresponding conditions by converting constraints like $x = y$ to $(x - y)^2$ and $x > y$ to $\mathsf{Relu}(y - x)$. The final objective to be minimized is:

$$\zeta = \zeta_1 + \alpha_1 \zeta_2 + \alpha_2 \zeta_3 + \alpha_3 \zeta_4, \tag{11}$$

where $\alpha_1, \alpha_2, \alpha_3$ are coefficients for regularization terms.

## LTL$_f$ Encoding Interpretation

We interpret an LTL$_f$ formula from an LTL$_f$ encoding by Algorithm 1. The algorithm interprets the LTL$_f$ encoding from bottom to top and calculates the score of the interpretations for each sub-formula ($s_j$ in line 12) according to the LTL$_f$ encoding. The score of an interpretation is obtained by multiplying all related parameters in the LTL$_f$ encoding. The following Example 6 illustrates how Algorithm 1 works.

**Example 6** *Suppose an LTL$_f$ encoding* $\theta$ *of size* 3 *shown in Table 1 and the beam width 2 are input to Algorithm 1. The outcomes of execution steps of Algorithm 1 are shown in*

**Algorithm 1:** Interpreting LTL$_f$ Formula

**Input** : LTL$_f$ encoding $\theta$ of size $L$, the beam width $\omega$, the training set $\Pi$
**Output** : An LTL$_f$ formula $\phi$ interpreted from $\theta$

1 $\theta_{\text{right}}, \theta_{\text{none}}, \theta_{\text{atom}}$, and $\theta_{\text{op}} \leftarrow$ computed by Equation (7) on $\theta$, where $op \in \{\neg, \wedge, \mathsf{X}, \mathsf{U}\}$;
2 $i \leftarrow L$;
3 **while** $i \geq 1$ **do**
4     $f_i \leftarrow \emptyset$;
5     **for** $p_k \in \mathbb{P}$ **do**
6         $f_i \leftarrow f_i \cup \{(p_k, (\theta_{\text{atom}})_{i,k})\}$;
7     **for** $(\phi_k, s_k) \in f_{i+1}$ **do**
8         $f_i \leftarrow f_i \cup \{((\neg, \phi_k), (\theta_\neg)_i s_k), ((\mathsf{X}, \phi_k), (\theta_{\mathsf{X}})_i s_k)\}$;
9     **for** $i + 2 \leq j \leq L$ **do**
10         **for** $((\phi_l, s_l), (\phi_r, s_r)) \in f_{i+1} \times f_j$ **do**
11             $f_i \leftarrow f_i \cup \{((\wedge, \phi_l, \phi_r), (\theta_\wedge)_i (\theta_{\text{right}})_{i,r} s_l s_r)\} \cup \{((\mathsf{U}, \phi_l, \phi_r), (\theta_{\mathsf{U}})_i (\theta_{\text{right}})_{i,r} s_l s_r)\}$;
12     Sort $f_i = \{(\phi_j, s_j)\}$ according to $s_j$, keep top $\omega$ elements and remove the others;
13     $i \leftarrow i - 1$;
14 $\phi \leftarrow$ the best formula obtained from $f_1$ according to the classification accuracy on $\Pi$;
15 **return** $\phi$;

| i | $(\theta_{\text{none}})_i$ | $(\theta_\neg)_i$ | $(\theta_\wedge)_i$ | $(\theta_{\mathsf{X}})_i$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 0.8 | 0 |
| 2 | 0 | 0 | 0 | 0.1 |
| 3 | 0 | 0 | 0 | 0 |
| i | $(\theta_{\mathsf{U}})_i$ | $(\theta_{\text{atom}})_{i,1}$ | $(\theta_{\text{atom}})_{i,2}$ | $(\theta_{\text{right}})_{i,3}$ |
| 1 | 0 | 0 | 0.2 | 1 |
| 2 | 0 | 0.3 | 0.6 | absent |
| 3 | 0 | 0.7 | 0.3 | absent |

Table 1: The parameters in Example 6.

| i | $\phi_i$ | related parameter | $f_i$ | score |
|---|---|---|---|---|
| 3 | $p_1$ | $(\theta_{\text{atom}})_{3,1} = 0.7$ | $\boldsymbol{p_1}$ | **0.7** |
| | $p_2$ | $(\theta_{\text{atom}})_{3,2} = 0.3$ | $\boldsymbol{p_2}$ | **0.3** |
| 2 | $p_1$ | $(\theta_{\text{atom}})_{2,1} = 0.3$ | $\boldsymbol{p_1}$ | **0.3** |
| | $p_2$ | $(\theta_{\text{atom}})_{2,2} = 0.6$ | $\boldsymbol{p_2}$ | **0.6** |
| | $\mathsf{X}\phi_3$ | $(\theta_{\mathsf{X}})_3 = 0.1$ | $\mathsf{X}p_1$ | 0.07 |
| | | | $\mathsf{X}p_2$ | 0.03 |
| 1 | $p_2$ | $(\theta_{\text{atom}})_{1,2} = 0.2$ | $\boldsymbol{p_2}$ | **0.2** |
| | $\phi_2 \wedge \phi_3$ | $(\theta_\wedge)_1 = 0.8$, $(\theta_{\text{right}})_{1,3} = 1$ | $p_1 \wedge p_1$ | 0.168 |
| | | | $\boldsymbol{p_1 \wedge p_2}$ | **0.336** |
| | | | $p_2 \wedge p_1$ | 0.072 |
| | | | $p_2 \wedge p_2$ | 0.144 |

Table 2: Executing Algorithm 1 for Example 6.

*Table 2, where the top 2 interpretations and their scores for each $f_i$ are displayed in bold. For $i = 3$, the interpretations of $f_3$ can only be an atomic proposition. The score of $f_3 = p_1$ is $(\theta_{\text{atom}})_{3,1}$ according to Algorithm 1 (line 5). We keep $p_1$ and $p_2$ as the top 2 interpretations for $f_3$. The score of $f_2 = \mathsf{X}p_1$ is obtained by multiplying $(\theta_{\mathsf{X}})_2$ and the score of $f_3 = p_1$. Continuing this way we finally obtain the top 2 interpretations for $f_1$, namely $p_2$ and $p_1 \wedge p_2$.*

## Evaluation

**Competitors.** We compared `TLTLf` with four SOTA approaches, including `C.&M.` (Camacho and McIlraith 2019), `BayesLTL` (Kim et al. 2019), `MaxSAT-DT` (Gaglione et al. 2021) and `GLTLf` (Luo et al. 2022). `C.&M.` cannot learn from imperfect data but can learn arbitrary formulae. `BayesLTL` can learn from imperfect data but cannot learn arbitrary formulae. `MaxSAT-DT`, `GLTLf` and our proposed `TLTLf` can learn arbitrary formulae from imperfect data.

**Datasets.** We reused the datasets that are provided by (Luo et al. 2022). There are 5 domains for $k_f \in \{3, 6, 9, 12, 15\}$ and each domain has 50 datasets. For each dataset, there is a formula with $k_f$ operators, and there are $250/250$ positive/negative traces for this formula constituting the training set and $500/500$ positive/negative traces for this formula constituting the test set. For generating the imperfect data, a portion of traces from the original data were randomly chosen to reverse labels; *i.e.*, original positive labels are turned to be negative, and vice versa. The percentage $\delta$ of traces with wrong labels is determined by the imperfect rate drawn from $\{10\%, 20\%, 30\%, 40\%\}$.

**Settings.** All experiments were conducted on a Linux system equipped with an Intel(R) Xeon(R) Gold 6248R processor with 3.0 GHz and 126 GB RAM. The time limit is set to 1 hour and the memory limit set to 10 GB for each instance. We used grid search to find optimal hyperparameters of `TLTLf`. We used Adam (Kingma and Ba 2015) to optimize the parameters in our model. Detailed settings for all approaches can be found in our technical report. In our experiments, all approaches first learn an LTL$_f$ formula and then are evaluated to estimate the classification performance on the test set.

**Comparison across datasets.** As shown in Table 3, `TLTLf` obviously outperforms `BayesLTL` and `GLTLf`. Although `MaxSAT-DT` and `C.&M.` seem to have the best performance, they run out of time in more cases when $k_f$ increases; in contrast, `TLTLf` keeps successful in all cases and it keeps a high accuracy when $k_f$ increases. If we treat the accuracy of failed cases as 0, then the accuracies of `MaxSAT-DT` and `C.&M.` on datasets with $k_f = 9$ drop to 16% and 68%, respectively, obviously lower than that of `TLTLf`.

**Comparison on imperfect data.** `TLTLf` also outperforms other approaches on imperfect data, as shown in Figure 1(a). `MaxSAT-DT` and `C.&M.` fail to solve any formula on imper-

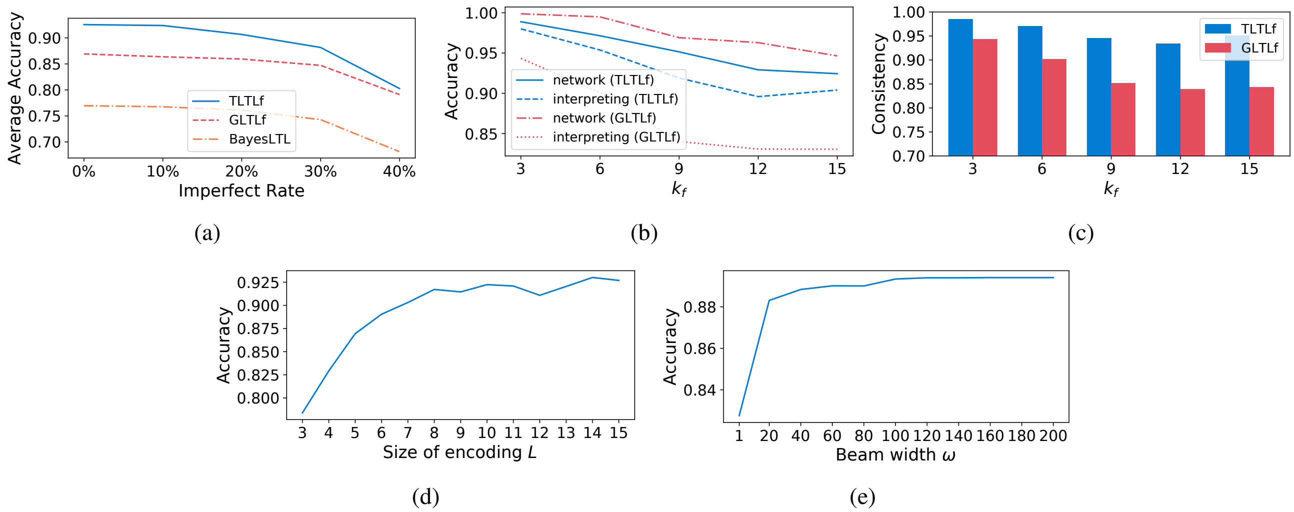


Figure 1: (a) Accuracy achieved by different imperfect rates. The results are averaged by 5 datasets with $k_f \in \{3, 6, 9, 12, 15\}$. (b) Network accuracy and the accuracy of formula interpreted from the network. (c) Consistency. (d) Accuracy achieved by different encoding sizes $L$. (e) Accuracy achieved by different beam widths $\omega$ for Algorithm 1.

| | $k_f = 3$ | | | $k_f = 6$ | | | $k_f = 9$ | | | $k_f = 12$ | | | $k_f = 15$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | $\mathrm{F}_1$ | $N_\mathrm{s}$ | Acc | $\mathrm{F}_1$ | $N_\mathrm{s}$ | Acc | $\mathrm{F}_1$ | $N_\mathrm{s}$ | Acc | $\mathrm{F}_1$ | $N_\mathrm{s}$ | Acc | $\mathrm{F}_1$ | $N_\mathrm{s}$ |
| MaxSAT-DT | 100 | 100 | 49 | 100 | 100 | 19 | 100 | 100 | 8 | 100 | 100 | 5 | 100 | 100 | 5 |
| C.&M. | 99.7 | 99.7 | 50 | 97.9 | 96.7 | 47 | 97.1 | 95.5 | 35 | 95.1 | 91.9 | 20 | 93.7 | 87.4 | 8 |
| BayesLTL | 85.1 | 85.9 | 50 | 77.9 | 76.7 | 50 | 74.0 | 75.7 | 50 | 72.7 | 73.4 | 50 | 74.8 | 77.3 | 50 |
| GLTLf | 94.3 | 94.2 | 50 | 90.0 | 90.3 | 50 | 84.0 | 83.2 | 50 | 83.0 | 83.2 | 50 | 83.0 | 83.5 | 50 |
| TLTLf | 98.0 | 97.9 | 50 | 95.3 | 95.4 | 50 | 91.9 | 91.3 | 50 | 89.5 | 88.9 | 50 | 90.4 | 90.2 | 50 |

Table 3: Experimental results for $L = 10$ across different approaches. Acc stands for the average accuracy (%) for successful cases. $\mathrm{F}_1$ stands for the average $\mathrm{F}_1$ score (%) for successful cases. $N_\mathrm{s}$ stands for the number of cases out of total 50 cases that are successfully solved within the time limit.

fect training data. In contrast, TLTLf still has a high average accuracy $88.47\%$ even when being trained on datasets with a moderately high imperfect rate $\delta = 0.3$.

**Comparison on the performance of interpreting.** Both TLTLf and GLTLf involve network training and interpreting, so we compare the performance gap between the two parts for TLTLf with that for GLTLf. Figure 1(b) shows that TLTLf has a smaller performance gap than GLTLf. This result suggests that the neural model underpinned TLTLf is more interpretable. We further assess consistency between the inference of neural model and that of the interpreted formula, indicating their agreement on the test set. For example, if the neural model and the interpreted formula give the same classification results on 95 out of 100 test traces, then the consistency between them is 95%. TLTLf achieves higher consistency, aligning with its reduced performance gap between the neural network and interpreted formula.

**Hyper-parameters analysis.** To analyze the impact of different hyper-parameters, we conducted experiments with various sizes of encoding and beam widths, on the datasets with $k_f = 9$. During the analysis of one hyper-parameter, the other hyper-parameter was set as default. From Figure 1(d), it can be seen that with the increase of the size of encoding, the accuracy increases to a certain extent until $L = 8$. This may be caused by that $k_f$ was set to 9 for all experimental datasets. Figure 1(e) shows that the accuracy first increases rapidly as the beam width increases and then remains stable. This implies that $\omega = 100$ is sufficient to guarantee a high accuracy for the formula interpreted by Algorithm 1.

## Conclusion and Future Work

Learning tree-structured $\mathrm{LTL}_f$ formulae from imperfect data is important and challenging. In this paper we have proposed TLTLf parameterized by the $\mathrm{LTL}_f$ encoding to simulate $\mathrm{LTL}_f$ inference. TLTLf bridges the gap between the concise tree-structured syntax and the complex $\mathrm{LTL}_f$ semantics. Besides, we have identified the faithful $\mathrm{LTL}_f$ encoding, which has a one-to-one correspondence to the prefix form of $\mathrm{LTL}_f$ formulae. Experiment results demonstrate that TLTLf achieves the SOTA performance and yields $\mathrm{LTL}_f$ formulae more consistent with the learnt neural network than existing approaches do. Future work will extend our approach to LTL or other formal languages.

## Acknowledgements

This paper was supported by National Natural Science Foundation of China (No. 62276284, 61976232, 61876204, 51978675), Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515011470, 2022A1515011355), Guangzhou Science and Technology Project (No. 202201011699), Shenzhen Science and Technology Program (KJZD20230923114059020), Guizhou Provincial Science and Technology Projects (No. 2022-259), 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

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2): 87–106.

Angluin, D.; Eisenstat, S.; and Fisman, D. 2015. Learning Regular Languages via Alternating Automata. In *IJCAI*, 3308–3314.

Arif, M. F.; Larraz, D.; Echeverria, M.; Reynolds, A.; Chowdhury, O.; and Tinelli, C. 2020. SYSLITE: syntax-guided synthesis of PLTL formulas from finite traces. In *FMCAD*, 93–103.

Asarin, E.; Donzé, A.; Maler, O.; and Nickovic, D. 2011. Parametric identification of temporal properties. In *RV*, 147–160.

Baier, J. A.; and McIlraith, S. A. 2006. Planning with Temporally Extended Goals Using Heuristic Search. In *ICAPS*, 342–345.

Bird, S.; Klein, E.; and Loper, E. 2009. *Natural Language Processing with Python*. O'Reilly.

Bollig, B.; Habermehl, P.; Kern, C.; and Leucker, M. 2009. Angluin-Style Learning of NFA. In *IJCAI*, 1004–1009.

Bombara, G.; Vasile, C. I.; Penedo, F.; Yasuoka, H.; and Belta, C. 2016. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *HSCC*, 1–10.

Brunello, A.; Sciavicco, G.; and Stan, I. E. 2020. Interval Temporal Logic Decision Tree Learning. *CoRR*, abs/2003.04952.

Camacho, A.; and McIlraith, S. A. 2019. Learning Interpretable Models Expressed in Linear Temporal Logic. In *ICAPS*, 621–630.

Cao, Z.; Tian, Y.; Le, T. B.; and Lo, D. 2018. Rule-based specification mining leveraging learning to rank. *Automated Software Engineering*, 25(3): 501–530.

Chen, D.; and Manning, C. D. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *EMNLP*, 740–750.

Ciccio, C. D.; Mecella, M.; and Mendling, J. 2013. The Effect of Noise on Mined Declarative Constraints. In *SIMPDA*, 1–24.

Fargier, H.; Gimenez, P.; and Mengin, J. 2018. Learning Lexicographic Preference Trees From Positive Examples. In *AAAI*, 2959–2966.

Fiterau-Brostean, P.; Lenaerts, T.; Poll, E.; de Ruiter, J.; Vaandrager, F. W.; and Verleg, P. 2017. Model learning and model checking of SSH implementations. In *SPIN*, 142–151.

Gabel, M.; and Su, Z. 2008. Symbolic mining of temporal specifications. In *ICSE*, 51–60.

Gabel, M.; and Su, Z. 2010. Online inference and enforcement of temporal properties. In *ICSE*, 15–24.

Gaglione, J.; Neider, D.; Roy, R.; Topcu, U.; and Xu, Z. 2021. Learning Linear Temporal Properties from Noisy Data: A MaxSAT-Based Approach. In *ATVA*, volume 12971, 74–90.

Giacomo, G. D.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860.

Giantamidis, G.; and Tripakis, S. 2016. Learning Moore Machines from Input-Output Traces. In *FM*, 291–309.

Hou, X.; Qi, P.; Wang, G.; Ying, R.; Huang, J.; He, X.; and Zhou, B. 2021. Graph Ensemble Learning over Multiple Dependency Trees for Aspect-level Sentiment Classification. In *NAACL-HLT*, 2884–2894.

Jin, X.; Donzé, A.; Deshmukh, J. V.; and Seshia, S. A. 2015. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11): 1704–1717.

Kasenberg, D.; and Scheutz, M. 2017. Interpretable apprenticeship learning with temporal logic specifications. In *CDC*, 4914–4921.

Kim, J.; Muise, C.; Shah, A.; Agarwal, S.; and Shah, J. 2019. Bayesian Inference of Linear Temporal Logic Specifications for Contrastive Explanations. In *IJCAI*, 5591–5598.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*, 1–15.

Kong, Z.; Jones, A.; and Belta, C. 2016. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3): 1210–1222.

la Rosa, T. D.; and McIlraith, S. A. 2011. Learning Domain Control Knowledge for TLPlan and Beyond. In *ICAPS-11 workshop PAL*, 1–8.

Le, T. B.; and Lo, D. 2015. Beyond support and confidence: Exploring interestingness measures for rule-based specification mining. In *SANER*, 331–340.

Lemieux, C.; Park, D.; and Beschastnikh, I. 2015. General LTL Specification Mining (T). In *ASE*, 81–92.

Leno, V.; Dumas, M.; Maggi, F. M.; Rosa, M. L.; and Polyvyanyy, A. 2020. Automated discovery of declarative process models with correlated data conditions. *Information Systems*, 89: 101482.

Li, R.; Chen, H.; Feng, F.; Ma, Z.; Wang, X.; and Hovy, E. H. 2021. Dual Graph Convolutional Networks for Aspect-based Sentiment Analysis. In *ACL/IJCNLP*, 6319–6329.

Luo, W.; Liang, P.; Du, J.; Wan, H.; Peng, B.; and Zhang, D. 2022. Bridging LTLf Inference to GNN Inference for Learning LTLf Formulae. In *AAAI*, 9849–9857.

Maggi, F. M.; Bose, R. P. J. C.; and van der Aalst, W. M. P. 2012. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In *CAiSE*, 270–285.

Maggi, F. M.; Ciccio, C. D.; Francescomarino, C. D.; and Kala, T. 2018. Parallel algorithms for the automated discovery of declarative process models. *Information Systems*, 74: 136–152.

Maggi, F. M.; Montali, M.; and Peñaloza, R. 2020. Temporal Logics Over Finite Traces with Uncertainty. In *AAAI*, 10218–10225.

Neider, D.; and Gavran, I. 2018. Learning Linear Temporal Properties. In *FMCAD*, 1–10.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57.

Roy, R.; Fisman, D.; and Neider, D. 2020. Learning Interpretable Models in the Property Specification Language. In *IJCAI*, 2213–2219.

Shah, A.; Kamath, P.; Shah, J. A.; and Li, S. 2018. Bayesian Inference of Temporal Task Specifications from Demonstrations. In *NeurIPS*, 3808–3817.

Shi, T.; Malioutov, I.; and Irsoy, O. 2020. Semantic Role Labeling as Syntactic Dependency Parsing. In *EMNLP*, 7551–7571.

Sun, Z.; Zhu, Q.; Xiong, Y.; Sun, Y.; Mou, L.; and Zhang, L. 2020. TreeGen: A Tree-Based Transformer Architecture for Code Generation. In *AAAI*, 8984–8991.

Xie, B.; Su, J.; Ge, Y.; Li, X.; Cui, J.; Yao, J.; and Wang, B. 2021. Improving Tree-Structured Decoder Training for Code Generation via Mutual Learning. In *AAAI*, 14121–14128.

Xu, Z.; Birtwistle, M.; Belta, C.; and Julius, A. 2016. A temporal logic inference approach for model discrimination. *IEEE life sciences letters*, 2(3): 19–22.

Xu, Z.; and Julius, A. A. 2016. Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Transactions on Automation Science and Engineering*, 15(1): 264–277.

Xu, Z.; Nettekoven, A. J.; Julius, A. A.; and Topcu, U. 2019. Graph Temporal Logic Inference for Classification and Identification. In *CDC*, 4761–4768.

Ye, R.; Zhuang, T.; Wan, H.; Du, J.; Luo, W.; and Liang, P. 2023. A Noise-Tolerant Differentiable Learning Approach for Single Occurrence Regular Expression with Interleaving. In *AAAI*, 4809–4817.

Zhang, Y.; Xia, Q.; Zhou, S.; Jiang, Y.; Fu, G.; and Zhang, M. 2022. Semantic Role Labeling as Dependency Parsing: Exploring Latent Tree Structures inside Arguments. In *COLING*, 4212–4227.