

# RESTL: Reinforcement Learning Guided by Multi-Aspect Rewards for Signal Temporal Logic Transformation

Yue Fang<sup>1,2</sup>, Zhi Jin<sup>1,2\*</sup>, Jie An<sup>3,4\*</sup>, Hongshen Chen<sup>5</sup>, Xiaohong Chen<sup>6</sup>, Naijun Zhan<sup>1,2</sup>

<sup>1</sup>School of Computer Science, Peking University, Beijing, China

<sup>2</sup>Key Laboratory of High Confidence Software Technologies (PKU), MOE, China

<sup>3</sup>National Key Laboratory of Space Integrated Information System

<sup>4</sup>Institute of Software, Chinese Academy of Sciences, Beijing, China

<sup>5</sup>JD.com, Beijing, China

<sup>6</sup>East China Normal University, Shanghai, China

y.fang@stu.pku.edu.cn, zhijin@pku.edu.cn, anjie@iscas.ac.cn

## Abstract

Signal Temporal Logic (STL) is a powerful formal language for specifying real-time specifications of Cyber-Physical Systems (CPS). Transforming specifications written in natural language into STL formulas automatically has attracted increasing attention. Existing rule-based methods depend heavily on rigid pattern matching and domain-specific knowledge, limiting their generalizability and scalability. Recently, Supervised Fine-Tuning (SFT) of large language models (LLMs) has been successfully applied to transform natural language into STL. However, the lack of fine-grained supervision on semantic fidelity, atomic proposition correctness, and formula readability often leads SFT-based methods to produce formulas misaligned with the intended meaning. To address these issues, we propose RESTL, a reinforcement learning (RL)-based framework for the transformation from natural language to STL. RESTL introduces multiple independently trained reward models that provide fine-grained, multi-faceted feedback from four perspectives, i.e., atomic proposition consistency, semantic alignment, formula succinctness, and symbol matching. These reward models are trained with a curriculum learning strategy to improve their feedback accuracy, and their outputs are aggregated into a unified signal that guides the optimization of the STL generator via Proximal Policy Optimization (PPO). Experimental results demonstrate that RESTL significantly outperforms state-of-the-art methods in both automatic metrics and human evaluations.

## Introduction

Signal Temporal Logic (STL) (Maler and Ničković 2004), an extension of classical Temporal Logic (TL) (Pnueli 1977), is currently a well-known specification language for formally specifying requirements of cyber-physical systems (CPS) with dense-time real-valued signals. STL has been applied to critical tasks such as model checking and runtime monitoring of CPS in both academia and industry (Maierhofer et al. 2020; Tellex et al. 2020). However, most of the requirements regarding the timing constraints of CPS are typically specified informally in domain documentation

written in natural language by domain experts. The lack of an effective method for transforming informal requirements into corresponding formal STL specifications has become a critical challenge, limiting its broader adoption in real-world CPS design and analysis.

Manually writing accurate STL formulas is a burdensome task for domain experts because it is time-consuming and error-prone. Consequently, many studies have explored automatic methods for transforming natural language descriptions into STL specifications to alleviate this burden and improve accuracy. Among existing methods, rule-based and pattern-based approaches are widely adopted. For example, fixed patterns have been used to transform natural language into intermediate forms in previous work (Lignos et al. 2015; Ghosh et al. 2016). Then, with a set of manually designed rules, the intermediate forms are further transformed into temporal logic formulas. These methods rely on meticulously crafted templates, which require substantial expert effort and steep learning curves (Kulkarni, Fisher, and Myers 2013). Moreover, they are typically limited to highly restrictive and structured natural language expressions that strictly match predefined patterns.

Advancements in natural language processing (NLP), particularly the impressive capabilities demonstrated by large language models (LLMs), have sparked a strong interest in employing these techniques for transforming natural language into STL. Recent techniques have explored different strategies for tackling the Natural Language to STL (NL-to-STL) transformation problem. For example, DeepSTL (He et al. 2022) uses grammar-guided data synthesis to train transformer-based models that learn to map natural language inputs to STL formulas. NL2TL (Chen et al. 2023) is a method based on Supervised Fine-Tuning (SFT) that constructs synthetic NL-STL pairs and performs instruction tuning on LLMs, relying solely on paired data as supervision to allow the generation of STL formulas from natural language instructions. KGST (Fang et al. 2025) is a two-stage method that first fine-tunes an LLM on NL-STL pairs and then refines its outputs using external knowledge.

While current methods have made notable progress in automating the transformation from natural language to STL,

\*Corresponding author

---

**Case 1 – Temporal Semantic Error**

---

*Natural Language Description:*

During 10–150 time units, if signal  $z_1$  is less than 0.2, then signal  $z_2$  remains less than 0.3 from 1 to 3 time units later.

-----  
*LLaMA 3-8B (fine-tuned):*

$\mathbf{G}_{[10,150]}((z_1 < 0.2) \rightarrow \mathbf{F}_{[1,3]}(z_2 < 0.3))$

-----  
*Ground Truth:*

$\mathbf{G}_{[10,150]}((z_1 < 0.2) \rightarrow \mathbf{G}_{[1,3]}(z_2 < 0.3))$

---

**Case 2 – Atomic Proposition Error**

---

*Natural Language Description:*

If the rear radar detects an obstacle and the reverse gear is engaged, then the rear brake signal `brake_rear` should be activated within 2 seconds.

-----  
*LLaMA 3-8B (fine-tuned):*

$\mathbf{G}_{[0,T]}(\text{radar\_rear.detect\_obstacle} \rightarrow$

$\mathbf{F}_{[0,2]}(\text{brake\_rear} = 1))$

-----  
*Ground Truth:*

$\mathbf{G}_{[0,T]}((\text{radar\_rear.detect\_obstacle} \wedge \text{gear\_rev} = 1) \rightarrow \mathbf{F}_{[0,2]}(\text{brake\_rear} = 1))$

---

**Case 3 – Formula Redundancy**

---

*Natural Language Description:*

The temperature  $T$  is consistently above 22°C during the first 2 hours and then rises above 30°C sometime between 2 and 4 hours.

-----  
*LLaMA 3-8B (fine-tuned):*

$(\mathbf{G}_{[0,120]}(T > 22) \wedge \mathbf{F}_{[0,240]}(T > 30) \wedge \mathbf{G}_{[120,240]}(T > 30))$

-----  
*Ground Truth:*

$\mathbf{G}_{[0,2]}(T > 22) \wedge \mathbf{F}_{[2,4]}(T > 30)$

---

Table 1: Examples of common errors in NL-to-STL transformation by the fine-tuned LLM. Underlined parts of formulas indicate incorrect outputs in LLaMA3 and correct ones in the ground truth.

their accuracy in generating correct STL formulas remains insufficient. Table 1 illustrates representative examples of common errors made by a fine-tuned LLaMA 3-8B model compared to the ground-truth. Specifically, Case 1 shows a temporal semantic error where the model fails to capture the meaning of “remains”, leading to the incorrect temporal operator “F”. Case 2 reflects atomic proposition misalignment, as the model omits the condition “reverse gear is engaged”. Case 3 illustrates formula redundancy, as the model generates unnecessary temporal constraints. These issues reflect the limitations of existing SFT-based approaches, which often rely on coarse-grained supervision and static training objectives, providing limited guidance for learning the precise semantic information required for accurate STL generation.

To address the limitations arising from coarse-grained supervision, we propose RESTL, a multi-aspect reward-guided reinforcement learning framework for transforming natural language into STL. RESTL introduces four complementary reward metrics to provide fine-grained supervision from multiple perspectives: (1) Atomic Proposition Alignment, which checks whether all key variables are accurately captured; (2) Templated Natural Language Simi-

larity, which evaluates semantic alignment by focusing on semantic content and logical consistency; (3) Formula Succinctness, which measures the difference between the length of the output formula and its ground truth to improve succinct yet faithful expressions. Normally, a smaller difference is better. and (4) STL-level Similarity, which measures the similarity between the generated and reference formulas, providing global supervision for the formula generation. Each metric corresponds to a lightweight reward model, which is trained via preference learning on multiple generated STL outputs to provide evaluative feedback for guiding the generator. To improve learning accuracy, we employ a curriculum learning strategy that gradually increases task difficulty for each reward model. Finally, the outputs of these reward models are aggregated into a unified scalar signal to optimize the STL generator using Proximal Policy Optimization (PPO) (Schulman et al. 2017). During this optimization, we incorporate a Kullback-Leibler (KL) regularization strategy to constrain drastic policy updates while maximizing the reward and enhancing training stability.

Experimental results show that our RESTL framework significantly outperforms baseline methods in both automatic and human evaluations. It achieves higher accuracy in transforming natural language descriptions into STL formulas, with better semantic alignment and readability.

In summary, our main contributions include:

- We propose RESTL, the first reinforcement learning framework for transforming natural language into STL. RESTL learns from multiple dimensions of feedback, including atomic proposition consistency, semantic alignment, formula succinctness, and symbol matching.
- We introduce a curriculum learning strategy to train each reward model from easier to harder examples, improving the accuracy of the reward and training stability.
- Experimental results on two datasets show that RESTL outperforms baselines in both automatic and human evaluations.

## Related Work

In this section, we present the most relevant related work, more details can be found in the Appendix A. Many efforts have been made to transform natural language (NL) into TL specifications. For example, a catalog of temporal logic formulas that capture common specification patterns in the design of concurrent and reactive systems was proposed in (Dwyer, Avrunin, and Corbett 1999). Controlled English has also been transformed into TL through the use of syntactic and grammatical dependency parsing, together with pre-defined mapping rules (Žilka 2010; Santos, Carvalho, and Sampaio 2018). Although these methods are effective in specific domains, they rely on handcrafted rules and restricted language inputs, which limit their ability to handle more diverse and complex expressions. To overcome these limitations, the nl2spec method (Cosler et al. 2023) integrates human feedback and LLMs to automatically derive TL formulas. However, these TL-focused methods are not readily extended to STL, because STL introduces real-valued signals

and continuous-time constraints, which pose challenges not typically addressed by traditional TL.

As an extension of TL that incorporates real-valued dense-time signals, STL has gained widespread use in both academia and industry to meet the requirements of CPS (Madsen et al. 2018). Consequently, numerous efforts have been made to transform natural language into STL. For example, DeepSTL (He et al. 2022) trains a Transformer model using grammar-based synthetic data. Although this approach ensures formal consistency, it heavily relies on handcrafted rules and artificial data, which fail to capture the diversity in real-world natural language. In addition, NL2TL (Chen et al. 2023) mitigates the reliance on rule design by fine-tuning a T5 model on NL-TL pairs generated by LLMs. KGST (Fang et al. 2025) uses a generate-then-refine approach by first fine-tuning LLMs to generate initial STL formulas, then refining them with external knowledge. However, as supervised fine-tuning methods, both NL2TL and KGST optimize fixed training objectives, lacking fine-grained feedback and struggling to accurately represent the semantics of the input natural language. To address these limitations, we propose RESTL, a reinforcement learning-based framework for NL-to-STL transformation that incorporates multi-aspect supervision and curriculum-guided reward modeling to enhance both the accuracy and readability of generated STL formulas.

## Preliminary

STL is a widely used formalism for specifying the real-time properties of CPS, such as autonomous vehicles, robotic systems, etc (Maierhofer et al. 2020; Tellex et al. 2020).

Let  $\mathbb{R}$  denote the set of real numbers, and let  $\mathbb{R}_{\geq 0}$  and  $\mathbb{R}_+$  denote the sets of non-negative and positive real numbers, respectively. We denote  $\mathbb{N}_{\geq 0}$  and  $\mathbb{N}_+$  the set of non-negative integers and the set of positive integers, respectively.

Given a time horizon  $T \in \mathbb{R}_+$  and a signal dimension  $d \in \mathbb{N}_+$ , a  $d$ -dimensional signal is a function  $\mathbf{v} : [0, T] \rightarrow \mathbb{R}^d$ . For any time  $t \in [0, T]$ ,  $\mathbf{v}(t) \in \mathbb{R}^d$  represents the values of  $d$  signal variables at time  $t$ . Each component may correspond to physical quantities such as velocity, RPM, or acceleration. In this paper, we fix a set  $X$  of such variables and refer to one-dimensional signals as signal variables.

**Definition 1 (STL Syntax)** STL formulas  $\varphi$  are constructed from atomic propositions  $\alpha$  as follows:

$$\begin{aligned} \alpha &::= f(x_1, \dots, x_K) > 0 \\ \varphi &::= \alpha \mid \perp \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{G}_I\varphi \mid \mathbf{F}_I\varphi \mid \varphi_1 \mathcal{U}_I\varphi_2 \end{aligned}$$

Here,  $\alpha$  represents atomic proposition, where  $f$  is a real-valued function over variables  $x_1, \dots, x_K \in X$ .  $I = [l, u] \subseteq \mathbb{R}_{\geq 0}$  is a closed interval with  $l < u$ , and  $l, u \in \mathbb{N}_{\geq 0}$ . The temporal operators  $\mathbf{G}$ ,  $\mathbf{F}$ , and  $\mathcal{U}$  denote “always”, “eventually”, and “until” respectively.

The Boolean semantics of an STL formula are evaluated over a signal  $\mathbf{v}$  at time  $t$  as follows:

$$\begin{aligned} (\mathbf{v}, t) \models \alpha &\Leftrightarrow f(\mathbf{v}(t)) \geq 0 \\ (\mathbf{v}, t) \models \neg\varphi &\Leftrightarrow (\mathbf{v}, t) \not\models \varphi \end{aligned}$$

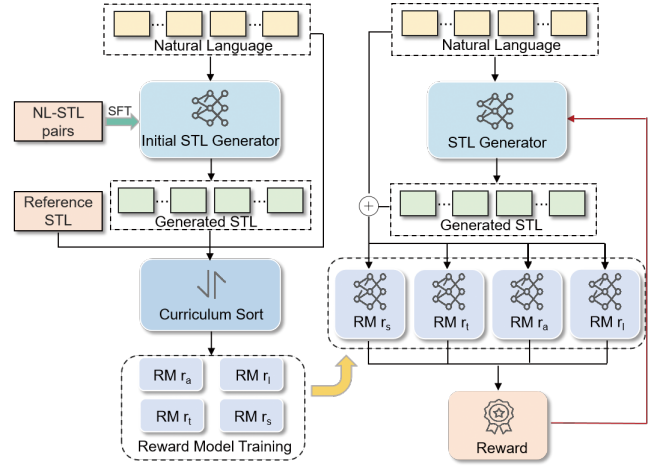


Figure 1: The RESTL framework. First, initialize the STL generator with NL-STL pairs. Then, natural language inputs with both reference and generated STL formulas are used to train multi-aspect reward models based on curriculum learning. Finally, the STL generator is optimized using PPO.

$$\begin{aligned} (\mathbf{v}, t) \models \varphi_1 \wedge \varphi_2 &\Leftrightarrow (\mathbf{v}, t) \models \varphi_1 \wedge (\mathbf{v}, t) \models \varphi_2 \\ (\mathbf{v}, t) \models \mathbf{G}_{[l, u]}\varphi &\Leftrightarrow \forall t' \in [t + l, t + u]. (\mathbf{v}, t') \models \varphi \\ (\mathbf{v}, t) \models \mathbf{F}_{[l, u]}\varphi &\Leftrightarrow \exists t' \in [t + l, t + u]. (\mathbf{v}, t') \models \varphi \\ (\mathbf{v}, t) \models \varphi_1 \mathcal{U}_{[l, u]}\varphi_2 &\Leftrightarrow \exists t' \in [t + l, t + u]. (\mathbf{v}, t') \models \varphi_2 \\ &\quad \wedge \forall t'' \in [t, t']. (\mathbf{v}, t'') \models \varphi_1 \end{aligned}$$

For example, considering one of safety requirements of a vehicle’s transmission controller (Ernst et al. 2022): “During the next 21 time units, whenever the velocity exceeds 40, the RPM must drop below 2500 within four time units.”, it can be expressed as an STL formula over real-valued signals as follows:  $\mathbf{G}_{[0, 21]}(\text{velocity} > 40 \rightarrow \mathbf{F}_{[1, 4]}(\text{RPM} < 2500))$ .

## Method

In this section, we present RESTL, a reinforcement learning framework for transforming natural language into STL. RESTL integrates four distinct reward models trained through curriculum learning to provide multi-aspect feedback, which is then aggregated to guide the optimization of the STL generator using the PPO algorithm, as illustrated in Figure 1. First, we fine-tune a LLaMA 3-8B model on a dataset of NL-STL pairs to obtain an initial STL generator. Second, for each metric, we train a separate reward model using preference-based paired examples and apply a curriculum learning strategy to improve the accuracy of the reward model feedback. Finally, the reward outputs from all models are aggregated into a unified scalar signal that provides feedback for PPO to optimize the STL generator.

### STL Generator Initialization

We fine-tune the LLaMA 3-8B model on a dataset of NL-STL pairs to obtain an initial STL generator capable of transforming natural language into STL. Each pair includes an Instruction guiding the transformation, an Input

NL description, and the target `Output` STL formula. The model learns to generate the STL formula from the input and instruction.

After training, the model serves as the initial STL generator with basic NL-to-STL transformation capabilities, ready for further fine-tuning on downstream tasks.

## Multi-aspect Metric

To address common issues in transforming natural language into STL formulas using LLMs, we design four reward functions as reinforcement learning feedback as shown in Figure 2. These reward functions guide the model toward generating more accurate STL formulas.

**Atomic Proposition Alignment** A common issue in NL-to-STL transformation is the incorrect identification or omission of atomic propositions, which are the fundamental variables in the formula. To evaluate the alignment between the generated formula  $\hat{y}$  and the ground truth  $y^*$ , we extract their atomic propositions  $A(\hat{y})$  and  $A(y^*)$  respectively using LLMs, then compute the precision metric:  $m_a = |A(\hat{y}) \cap A(y^*)|/|A(y^*)|$ .

**Templated NL Similarity** LLMs often produce semantic errors in NL-to-STL transformation, such as incorrect temporal logic operators, thresholds, or time ranges, which cause mismatches with the original intention of natural language. To enforce consistency at the semantic level, we reverse-map the generated STL formula  $\hat{y}$  into a templated natural language  $T(\hat{y})$ . These templated natural language sentences are generated by LLMs based on templates defined by STL syntax rules. We then use a pre-trained language model encoder (e.g., BERT) to convert both  $T(\hat{y})$  and the original input  $x$  into dense vector representations  $v_{T(\hat{y})}$  and  $v_x$ . Their semantic similarity is computed by cosine similarity as the metric  $m_t = \text{CoS}(v_x, v_{T(\hat{y})})$ .

**Formula Succinctness** To encourage the generation of concise and human-readable STL formulas, we design a reward function based on length difference. Let  $|\hat{y}|$  and  $|y^*|$  denote the number of characters in the generated STL formula and the ground truth formula, respectively. We define the normalized length difference metric  $m_l$  as:  $m_l = 1 - ||y^*| - |\hat{y}||/|y^*|$ . This metric rewards formulas close in length to the reference. Formulas that are too long may include redundant or unnecessary components, while formulas that are too short may omit important logical content. Formulas with length near the reference get higher scores.

**STL-level Similarity** To measure the overall similarity between the generated STL formula and the ground truth, we adopt the ROUGE-L (Lin 2004) score as the metric  $m_r$ . ROUGE-L evaluates the longest common subsequence between two sequences, capturing both content overlap and order information. This makes it suitable for assessing the structural and semantic consistency of STL formulas. Given the generated formula  $\hat{y}$  and ground truth  $y^*$ , the metric is defined as:  $m_s = \text{ROUGE-L}(y^*, \hat{y})$ .

## Reward Model Training

We introduce reward models based on LLaMA 3-8B to evaluate generated STL formulas. For a given input  $x$ , the initial STL generator produces multiple candidate formulas  $\hat{y}_1, \dots, \hat{y}_k$  selected by minimizing pairwise ROUGE similarity, where  $k$  is a hyperparameter. All candidate formulas are evaluated with the four reward functions based on the metrics  $m_a$ ,  $m_t$ ,  $m_l$ , and  $m_s$ . For each reward metric, we compute scores for the candidates and compare them to construct ('chosen', 'rejected') pairs, where the higher-scoring candidate is labeled as 'chosen'. For example, for the metric  $m_s$ , if  $m_s(\hat{y}^{(i)}) > m_s(\hat{y}^{(j)})$ , we collect the following pair:

$$\{(\text{chosen} : [x, \hat{y}^{(i)}], \text{rejected} : [x, \hat{y}^{(j)}]) \mid m_s(\hat{y}^{(i)}) > m_s(\hat{y}^{(j)})\}.$$

This process creates preference data that enables reward models to learn fine-grained preferences for STL generation quality.

When training the reward models, we use the Bradley-Terry model (Bradley and Terry 1952) to formulate the preference distribution with the reward model  $r_\psi$  as follows:

$$P_\psi(y_c \succ y_r | x) = \sigma(r_\psi(x, y_c) - r_\psi(x, y_r)),$$

where  $\sigma$  is the logistic function,  $y_c$  denotes the chosen STL, and  $y_r$  represents the rejected STL. This is treated as a binary classification task, yielding the following negative log-likelihood loss function:

$$L_{\text{rm}} = -\mathbb{E}_{D_{\text{rm}}} [\log \sigma(r_\psi(x, y_c) - r_\psi(x, y_r))],$$

where  $D_{\text{rm}}$  is the preference dataset.

In this work, we initialize the reward models using the initial STL generator. Additionally, a linear layer is added on top of the final transformer layer to produce a scalar prediction representing the reward value.

Let  $r_a$ ,  $r_t$ ,  $r_l$ , and  $r_s$  denote the **reward models** for the metrics  $m_a$ ,  $m_t$ ,  $m_l$ , and  $m_s$ , respectively. Given the input natural language  $x$  and the generated STL formula  $\hat{y}$ , the corresponding rewards can be computed as  $r_a(x, \hat{y})$ ,  $r_t(x, \hat{y})$ ,  $r_l(x, \hat{y})$ , and  $r_s(x, \hat{y})$ , abbreviated as  $r_a(\hat{y})$ ,  $r_t(\hat{y})$ ,  $r_l(\hat{y})$ , and  $r_s(\hat{y})$ . To facilitate aggregation of the rewards, the scores from each reward model are scaled to the range  $[0, 1]$ .

## Curriculum Learning for Reward Models

To improve the effectiveness of reward models, we incorporate curriculum learning by organizing training examples from easy to more challenging cases. We define separate curricula based on the four reward metrics, each capturing distinct reward model evaluation criteria.

**Atomic Proposition Curriculum for  $r_a$**  This curriculum is used for training the atomic proposition alignment reward model. We count the number of atomic propositions in the ground truth STL formula and sort the training samples in ascending order of this count. Samples with fewer atomic propositions are considered easier and are prioritized, enabling the model can gradually learn to assess atomic proposition alignment.

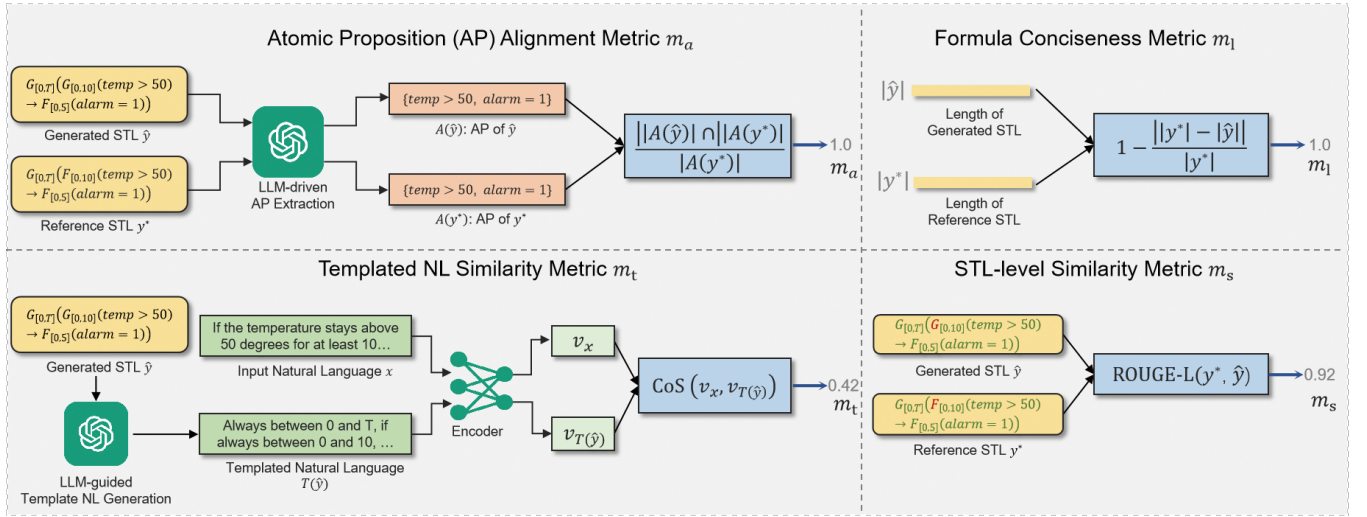


Figure 2: Design of the four reward metrics  $m_a$ ,  $m_l$ ,  $m_t$ , and  $m_s$ . We present the evaluation process of the same natural language input and its corresponding generated STL formula under different reward metrics.

**NL Similarity Curriculum for  $r_t$**  When training the reward model of templated natural language similarity, for each natural language description in the preference dataset, we first convert the corresponding three generated STL formulas into three templated natural language sentences. Then, we compute the cosine similarity between the original natural language sentence and each templated sentence, and use the average similarity of the three pairs to determine the sample order. These training data are sorted in ascending order of difficulty scores.

**STL Formula Length Curriculum for  $r_l$**  For training the formula succinctness reward model, samples are sorted by the length of the ground truth STL formula. Shorter STL formulas are regarded as easier and introduced earlier, with progressively longer formulas introduced as training proceeds.

**STL Similarity Curriculum for  $r_s$**  This curriculum is designed to train the STL-level similarity reward model. For each group of three generated STL formulas in the preference dataset, we first calculate the ROUGE-L score between each generated formula and the ground truth STL formula. The average of these three ROUGE-L scores is then used to compute the difficulty score. Training samples are sorted in ascending order of difficulty scores.

### Reinforcement Learning for STL Generator

Given a natural language instruction  $x$  and the generated STL formula  $\hat{y}$ , the overall reward is computed as:

$$r_{\text{RL}}(\hat{y}) = \lambda_1 r_a(\hat{y}) + \lambda_2 r_l(\hat{y}) + \lambda_3 r_t(\hat{y}) + \lambda_4 r_s(\hat{y}),$$

where the hyperparameters  $\lambda_1, \lambda_2, \lambda_3, \lambda_4$  control the relative importance of each reward. The reward objective  $r_{\text{total}}$  is to maximize expected reward while constraining deviation from the initial policy  $G_{\theta_0}$ .

$$r_{\text{total}} = r_{\text{RL}}(\hat{y}) - \eta \cdot \text{KL}(G_{\theta} \parallel G_{\theta_0}),$$

where  $\eta$  is a KL penalty coefficient. This term stabilizes training by penalizing large shifts from the pre-trained generator.

Finally, we apply the PPO algorithm (Schulman et al. 2017) to optimize the STL generator  $G_{\theta}$  using the KL-regularized reward signal.

## Experiments

In this section, we conduct comprehensive experiments to evaluate our proposed method RESTL.

### Experimental Settings

We first introduce the empirical settings, including datasets, baselines, evaluation measures, and implementation details.

**Datasets** We conduct experiments on two datasets for NL-to-STL transformation, including the DeepSTL dataset (He et al. 2022) and STL-DivEn dataset (Fang et al. 2025). The DeepSTL dataset is synthetically generated using a grammar-based generator that samples STL formulas from predefined templates and operator distributions. STL-DivEn is created through a hybrid approach that integrates GPT-4-based generation and human verification. For a fair comparison, we randomly select 14,000 samples from each dataset for training and 2,000 samples for testing.

**Baselines** We compare RESTL with five baseline methods: DeepSTL (He et al. 2022), GPT-3.5<sup>1</sup>, GPT-4<sup>2</sup>, DeepSeek (Guo et al. 2025), and KGST (Fang et al. 2025). In our implementation, we adopt the “gpt-4-0125-preview” version of GPT-4, the “gpt-3.5-turbo-1106” version of GPT-3.5, and the “DeepSeek-V1” version of DeepSeek.

<sup>1</sup><https://platform.openai.com/docs/models/gpt-3-5-turbo>

<sup>2</sup><https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4>

DeepSTL adopts the Transformer architecture for training and optimizes model parameters using the Adam algorithm (Kingma 2014). KGST is the SOTA model, which adopts a two-stage architecture: it first fine-tunes a LLaMA 3-8B model on an NL-STL dataset to produce preliminary STL formulas. In the second stage, it retrieves the top-5 most similar NL-STL pairs from the training set as reference examples, and leverages GPT-4 to refine the initial outputs, generating the final STL formulas.

**Evaluation Measures** To evaluate STL generation quality, we use both automatic metrics and human evaluation. Following prior work (He et al. 2022), we adopt formula accuracy, template accuracy, and BLEU (Papineni et al. 2002). Formula accuracy measures exact token-level matches, template accuracy assesses structural alignment, and BLEU captures n-gram overlap for lexical similarity. Definitions of formula and template accuracy are provided in the Appendix C.

For human evaluation, we randomly sample 100 NL-STL pairs from testing sets of STL-DivEn and DeepSTL. Five trained annotators, familiar with STL semantics and syntax, assess the output without knowing the model source. The evaluation is based on three criteria: readability (i.e. ease of understanding), syntactic correctness, and consistency with the original semantics. Readability is judged only if the first two criteria are met. This evaluation is designed to complement metric-based methods, which may miss cases where STL formulas differ in form but share the same meaning. Each comparison between RESTL and a baseline is labeled as win, loss, or tie, reflecting overall clarity and correctness.

**Implementation Details** Our experiments run on 8 NVIDIA GeForce RTX 4090 GPUs (24GB each). Each reward model is fine-tuned on LLaMA 3-8B with a linear value head for 5 epochs using Adam (lr=5e-5, batch=16). The STL generator is fine-tuned via PPO for 80,000 steps (batch=32, lr=1.41e-5, KL penalty  $\eta = 0.05$ ). The combined reward uses weighted scores with  $\lambda_1 = 0.2$ ,  $\lambda_2 = 0.25$ ,  $\lambda_3 = 0.35$ , and  $\lambda_4 = 0.2$ . More details, including hyperparameter discussions, are in Appendix D.

## Main Results

We demonstrate results on two datasets to evaluate RESTL.

**Metric-Based Evaluation** As shown in Table 2a, on the STL-DivEn dataset, RESTL reaches a formula accuracy of 68.38%, exceeding the strongest baseline KGST (55.87%). It also achieves a template accuracy of 69.74% (vs. KGST’s 56.27%) and a BLEU score of 0.3347, higher than KGST’s 0.2142. These results demonstrate RESTL’s superior ability in generating more accurate STL formulas. Similarly, as shown in Table 2b, RESTL achieves the best performance on the DeepSTL dataset, with a formula accuracy of 59.85%, a template accuracy of 63.27%, and a BLEU score of 0.6783. Compared to KGST (45.38%, 49.39%, and 0.5686, respectively), RESTL demonstrates clear improvements across all metrics. We conduct a significance test, confirming that RESTL significantly outperforms existing models across datasets and metrics, i.e., p-value < 0.01. Experimental results including measures of variability are provided

(a) STL-DivEn dataset			
Model	Formula Acc.	Template Acc.	BLEU
DeepSTL	0.1986	0.1883	0.0293
GPT-3.5	0.3018	0.3034	0.0424
GPT-4	0.4733	0.4741	0.0831
DeepSeek	0.4790	0.4825	0.0791
KGST	0.5587	0.5627	0.2142
RESTL (Ours)	<b>0.6838</b>	<b>0.6974</b>	<b>0.3347</b>

(b) DeepSTL dataset			
Model	Formula Acc.	Template Acc.	BLEU
DeepSTL	0.2002	0.2916	0.3332
GPT-3.5	0.2145	0.3002	0.2249
GPT-4	0.2262	0.3048	0.2881
DeepSeek	0.2537	0.3254	0.3982
KGST	0.4538	0.4939	0.5686
RESTL (Ours)	<b>0.5985</b>	<b>0.6327</b>	<b>0.6783</b>

Table 2: Performance comparison of RESTL and baselines on STL-DivEn and DeepSTL datasets.

Model	vs. STL-DivEn			vs. DeepSTL		
	Win	Loss	Tie	Win	Loss	Tie
DeepSeek	64.2	12.3	23.5	56.3	17.2	26.5
GPT-4	61.0	13.5	25.5	54.7	18.6	26.7
KGST	58.7	15.9	25.4	52.8	19.7	27.5

Table 3: Human evaluation (%) of RESTL vs. baselines.

in Appendix E.1.

**Human Evaluation** The results shown in Table 3 indicate that annotators consistently prefer formulas generated by RESTL, as they exhibit more concise and readable expressions while maintaining semantic consistency. For example, RESTL achieved win rates of 64.2%, 61.0%, and 58.7% against DeepSeek, GPT-4, and KGST, respectively. We attribute this readability advantage to the introduction of the formula succinctness reward ( $m_l$ ) during reinforcement learning, which explicitly encourages the model to generate STL formulas with lengths close to the reference, thereby improving clarity and readability.

## Ablation Study

We conduct ablation studies on both datasets by removing one reward metric at a time and training the model with the remaining three. The results are shown in Table 4, and more details are provided in the Appendix E.2.

(1) Impact of  $m_a$  for atomic proposition alignment: for STL-DivEn dataset, removing  $m_a$  yields 65.73% formula accuracy, 65.94% template accuracy, and a BLEU score of 0.3117, all outperforming the fine-tuned LLaMA 3-8B baseline. For DeepSTL dataset, removing  $m_a$  still better than baseline. However, these improvements remain limited compared to the full multi-reward combination in RESTL. These results indicate that  $m_a$  as a reward feedback is effective.

(2) Impact of  $m_l$  for templated NL similarity: without



(a) STL-DivEn dataset

Model	Formula Acc.	Template Acc.	BLEU
RESTL	<b>0.6838</b>	<b>0.6974</b>	<b>0.3347</b>
- w/o $m_a$	0.6573	0.6594	0.3117
- w/o $m_t$	0.6424	0.6503	0.3095
- w/o $m_l$	0.6642	0.6709	0.3294
- w/o $m_s$	0.6314	0.6393	0.2913
LLaMA3 (Fine-tuned)	0.4956	0.5007	0.1784

(b) DeepSTL dataset

Model	Formula Acc.	Template Acc.	BLEU
RESTL	<b>0.5985</b>	<b>0.6327</b>	<b>0.6783</b>
- w/o $m_a$	0.5571	0.5642	0.6129
- w/o $m_t$	0.5683	0.5782	0.6293
- w/o $m_l$	0.5832	0.5927	0.6473
- w/o $m_s$	0.5409	0.5503	0.6091
LLaMA3 (Fine-tuned)	0.2850	0.3285	0.5579

Table 4: Ablation study of different reward feedback on STL-DivEn and DeepSTL datasets.

Model	#AP	#Operator	#Value	#Redundancy
LLaMA3-8B (Fine-tuned)	19	39	27	22
KGST	15	27	25	26
RESTL	6	15	18	14

Table 5: Error analysis of RESTL, KGST, and fine-tuned LLaMA3-8B. #AP, #Operator, #Value, and #Redundancy denote counts of atomic proposition, operator, value, and redundancy errors, respectively.

$m_t$ , formula accuracy drops to 64.24% on STL-DivEn and 56.83% on DeepSTL, with template accuracies of 65.03% and 57.82% and BLEU scores of 0.3095 and 0.6293, respectively, showing that  $m_t$  is effective.

(3) Impact of  $m_l$  for formula succinctness: removing  $m_l$  yields the highest automatic scores among all ablation settings, with 66.42% formula accuracy and a BLEU score of 0.3294 on STL-DivEn, and 58.32% formula accuracy and 0.6473 BLEU on DeepSTL, indicating that  $m_l$  has limited metric impact but improves readability.

(4) Impact of  $m_s$  for STL-level similarity: removing  $m_s$  leads to the largest performance drop among all ablations, with 63.14% formula accuracy and a BLEU score of 0.2913 on STL-DivEn, and 54.09% accuracy and 0.6091 BLEU on DeepSTL, showing that  $m_s$  is the most significant reward for improving accuracy.

## Error Analysis

As shown in Table 5, we analyze 100 STL formulas generated by RESTL, KGST, and fine-tuned LLaMA 3-8B, categorizing errors into four types: atomic proposition (AP), operator, numerical value, and redundancy. Compared to the baselines, RESTL shows fewer AP errors due to the AP Alignment reward, and fewer operator and value errors thanks to the Templated NL Similarity metric. The Formula Conciseness reward also helps reduce redundancy, whereas KGST tends to include more irrelevant content, likely due

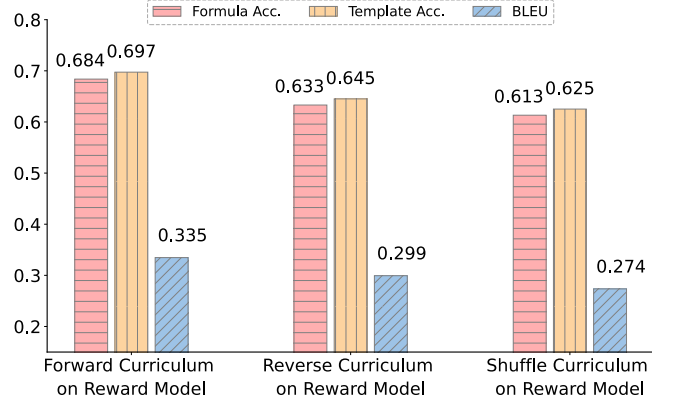


Figure 3: Scheduling strategy impact of different curricula.

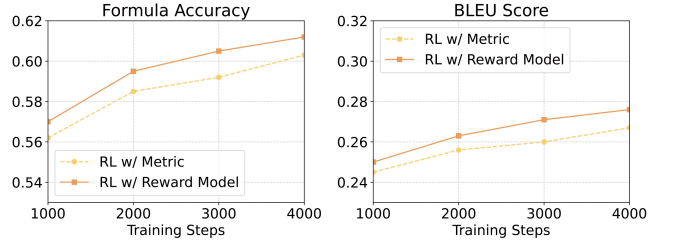


Figure 4: Comparison of reinforcement learning feedback strategies: reward model vs. metric supervision on STL-DivEn.

to its retrieval-augmented design. The STL-level Similarity metric is excluded from this analysis as it serves as a global training signal.

## Impacts of Curriculum Learning

As shown in Figure 3, we compare how different training data orders for reward models affect the performance of the RESTL framework. The results show that reward models trained with curriculum learning more effectively improve formula accuracy, template accuracy, and BLEU scores, significantly enhancing the overall performance of RESTL. Detailed impacts on individual reward models are provided in Appendix E.3.

## Reward Model vs. Direct Metric in RL

To compare reward model feedback with direct metric supervision in reinforcement learning, we evaluate their performance on the STL-DivEn dataset. As shown in Figure 4, models trained with reward models achieve higher gains in formula and template accuracy. Specifically, template accuracy reaches 61.2% with reward models versus 60.3% with metric-based supervision, and BLEU improves from 0.267 to 0.276. This advantage is due to the reward model’s ability to capture deeper NL-STL correspondence through preference learning, while metric supervision may introduce noise. These results suggest reward models provide more effective guidance for RL in this task.

## Conclusion

In this work, we propose RESTL, a reinforcement learning framework for transforming natural language into STL specifications. It employs multi-aspect reward models to ensure semantic correctness and uses curriculum learning to improve training efficiency. Experiments on two benchmarks show that RESTL outperforms existing methods in formula accuracy, template accuracy, and BLEU. By integrating structured rewards with progressive training, RESTL provides an effective solution for formal specification generation of cyber-physical systems.

## Acknowledgement

We sincerely thank the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China under Grant Nos. 62192731, 62192732, 62192730, and 62272166.

## References

- Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48.
- Bradley, R. A.; and Terry, M. E. 1952. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4): 324–345.
- Chen, Y.; Gandhi, R.; Zhang, Y.; and Fan, C. 2023. NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, 15880–15903. Association for Computational Linguistics.
- Christiano, P. F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.
- Cosler, M.; Hahn, C.; Mendoza, D.; Schmitt, F.; and Trippe, C. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In Enea, C.; and Lal, A., eds., *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*, volume 13965 of *Lecture Notes in Computer Science*, 383–396. Springer.
- Dann, C.; Mansour, Y.; and Mohri, M. 2023. Reinforcement learning can be more efficient with multiple rewards. In *International Conference on Machine Learning*, 6948–6967. PMLR.
- Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1999. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, 411–420.
- Ernst, G.; Arcaini, P.; Fainekos, G.; Formica, F.; Inoue, J.; Khandait, T.; Mahboob, M. M.; Menghi, C.; Pedrielli, G.; Waga, M.; Yamagata, Y.; and Zhang, Z. 2022. ARCH-COMP 2022 Category Report: Falsification with Unbounded Resources. In Frehse, G.; Althoff, M.; Schoitsch, E.; and Guiochet, J., eds., *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90 of *EPiC Series in Computing*, 204–221. EasyChair.
- Fang, Y.; Jin, Z.; An, J.; Chen, H.; Chen, X.; and Zhan, N. 2025. Enhancing Transformation from Natural Language to Signal Temporal Logic Using LLMs with Diverse External Knowledge. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, 10446–10458. Association for Computational Linguistics.
- Ghosh, S.; Elenius, D.; Li, W.; Lincoln, P.; Shankar, N.; and Steiner, W. 2016. ARSENAL: automatic requirements specification extraction from natural language. In *NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings 8*, 41–46. Springer.
- Graves, A.; Bellemare, M. G.; Menick, J.; Munos, R.; and Kavukcuoglu, K. 2017. Automated curriculum learning for neural networks. In *international conference on machine learning*, 1311–1320. Pmlr.
- Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- He, J.; Bartocci, E.; Nickovic, D.; Isakovic, H.; and Grosu, R. 2022. DeepSTL - From English Requirements to Signal Temporal Logic. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, 610–622. ACM.
- Justesen, N.; Rodriguez Torrado, R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation. In *NeurIPS Workshop on Deep Reinforcement Learning*.
- Kingma, D. P. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kulkarni, D.; Fisher, A. N.; and Myers, C. J. 2013. A new assertion property language for analog/mixed-signal circuits. In *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, 1–8. IEEE.
- Li, R.; Jabri, A.; Darrell, T.; and Agrawal, P. 2020. Towards practical multi-object manipulation using relational reinforcement learning. In *2020 IEEE international conference on robotics and automation (icra)*, 4051–4058. IEEE.
- Lignos, C.; Raman, V.; Finucane, C.; Marcus, M. P.; and Kress-Gazit, H. 2015. Provably correct reactive control from natural language. *Auton. Robots*, 38(1): 89–105.
- Lin, C.-Y. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, 74–81.
- Madsen, C.; Vaidyanathan, P.; Sadraddini, S.; Vasile, C. I.; DeLateur, N. A.; Weiss, R.; Densmore, D.; and Belta, C. 2018. Metrics for Signal Temporal Logic Formulae. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, 1542–1547. IEEE.



- Maierhofer, S.; Rettinger, A.-K.; Mayer, E. C.; and Althoff, M. 2020. Formalization of interstate traffic rules in temporal logic. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, 752–759. IEEE.
- Maler, O.; and Ničković, D. 2004. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT 2004*, volume 3253 of *LNCS*, 152–166. Springer.
- Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PmLR.
- Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 311–318.
- Paszke, A. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.
- Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS 1977*, 46–57. IEEE.
- Ryu, S.; Do, H.; Kim, Y.; Lee, G.; and Ok, J. 2024. Multi-Dimensional Optimization for Text Summarization via Reinforcement Learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 5858–5871.
- Santos, T.; Carvalho, G.; and Sampaio, A. 2018. Formal modelling of environment restrictions from natural-language requirements. In *Formal Methods: Foundations and Applications: 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26–30, 2018, Proceedings 21*, 252–270. Springer.
- Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Team, K.; Du, A.; Gao, B.; Xing, B.; Jiang, C.; Chen, C.; Li, C.; Xiao, C.; Du, C.; Liao, C.; et al. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.
- Tellex, S.; Gopalan, N.; Kress-Gazit, H.; and Matuszek, C. 2020. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1): 25–55.
- Wang, R.; Lehman, J.; Clune, J.; and Stanley, K. O. 2019. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*.
- Wang, Y.; Zhang, H.; Pang, L.; Guo, B.; Zheng, H.; and Zheng, Z. 2025. MaFeRw: Query rewriting with multi-aspect feedbacks for retrieval-augmented large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 25434–25442.
- Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.
- Xie, T.; Gao, Z.; Ren, Q.; Luo, H.; Hong, Y.; Dai, B.; Zhou, J.; Qiu, K.; Wu, Z.; and Luo, C. 2025. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*.
- Zheng, R.; Dou, S.; Gao, S.; Hua, Y.; Shen, W.; Wang, B.; Liu, Y.; Jin, S.; Liu, Q.; Zhou, Y.; et al. 2023. Secrets of rlhf in large language models part i: Ppo. *arXiv preprint arXiv:2307.04964*.
- Zheng, Y.; Zhang, R.; Zhang, J.; YeYanhan, Y.; and Luo, Z. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 400–410.
- Ziegler, D. M.; Stiennon, N.; Wu, J.; Brown, T. B.; Radford, A.; Amodei, D.; Christiano, P.; and Irving, G. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.
- Žilka, L. 2010. *Temporal logic for man*. Ph.D. thesis, Master’s thesis, Brno University of Technology.

## A The Appendix of Related Work

### Reinforcement Learning for LLMs

With the rapid development of LLMs, reinforcement learning (RL) has been widely adopted to further improve their capabilities (Ziegler et al. 2019; Christiano et al. 2017). One representative approach is reinforcement learning from human feedback (RLHF) (Christiano et al. 2017), which optimizes model performance using reward scores generated by reward models, combined with policy gradient algorithms such as PPO (Schulman et al. 2017). To provide more fine-grained supervision signals, prior work has introduced critic models to compute intermediate-stage rewards (Mnih et al. 2016; Zheng et al. 2023). Recent studies have shown that reinforcement learning with multiple reward signals can lead to more efficient training (Dann, Mansour, and Mohri 2023). For example, multi-dimensional optimization has been used to balance various quality aspects such as consistency and relevance (Ryu et al. 2024). In the RAG domain, MaFeRw (Wang et al. 2025) integrates multi-aspect feedback from both retrieval and generation stages to improve output quality and training stability. In this study, we apply reinforcement learning with multi-aspect dense reward signals to enhance the model’s ability to transform natural language descriptions into accurate STL specifications.

### Curriculum Learning for LLMs

Curriculum Learning (CL) (Bengio et al. 2009; Graves et al. 2017) is a training paradigm that organizes training samples in order, guiding the model to learn from easier examples before gradually progressing to more complex ones. In the context of Reinforcement Learning (RL), curricula are typically designed for specific tasks by incrementally increasing the complexity of the environment (Justesen et al. 2018; Wang et al. 2019; Li et al. 2020), thereby improving the generalization and transferability of the learned policies. Recently, with the growing application of RL in the post-training phase of LLMs, curriculum learning has shown great potential for improving both training efficiency and model performance. For instance, Kimi k1.5 (Team et al. 2025) and LogicRL (Xie et al. 2025) train models on “easy” samples for a fixed number of steps before switching to more “difficult” examples. In this work, we apply curriculum learning to train the reward models by ranking NL-to-STL transformation difficulty across multiple dimensions, enabling them to start from simple samples and progressively adapt to accurately evaluate more difficult tasks.

## B The Appendix of Method

### Details of Reinforcement Learning in RESTL

The RL training process of RESTL is shown in Algorithm 1. It starts with the prepared initial STL generator  $G_{\theta_0}$ .

In each training epoch, we compute the aggregated multi-aspect reward for each sample  $x^{(i)} \in D$  (Line 3). First, for each NL description  $x^{(i)}$ , we generate a corresponding STL formula  $\hat{y}^{(i)}$  (Line 4). Then, we compute its scores using the four reward models and aggregate them into an overall reward (Line 5 to 10). Formally, given a natural language

---

#### Algorithm 1: Training STL Generator with Multi-Reward PPO

---

**Input:** Training data  $D = \{x^{(i)}\}_{i=1}^N$ , pretrained generator  $G_{\theta_0}$ , reward models  $r_a, r_t, r_l, r_s$ , reward weights  $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ , PPO parameters, KL coefficient  $\eta$

**Output:** Optimized STL generator  $G_\theta$

- 1 Initialize generator  $G_\theta \leftarrow G_{\theta_0}$ ;
- 2 **for each training epoch do**
- 3     **foreach**  $x^{(i)} \in D$  **do**
- 4         // Sample a generated STL formula
- 4         Sample  $\hat{y}^{(i)} \sim G_\theta(x^{(i)})$ ;
- 4         // Compute multi-aspect rewards
- 5         Compute atomic proposition alignment:  
 $r_a^{(i)} \leftarrow r_a(x^{(i)}, \hat{y}^{(i)})$ ;
- 6         Compute templated NL similarity:  
 $r_t^{(i)} \leftarrow r_t(x^{(i)}, \hat{y}^{(i)})$ ;
- 7         Compute formula succinctness:  
 $r_l^{(i)} \leftarrow r_l(x^{(i)}, \hat{y}^{(i)})$ ;
- 8         Compute STL-level similarity:  
 $r_s^{(i)} \leftarrow r_s(x^{(i)}, \hat{y}^{(i)})$ ;
- 8         // Aggregate overall reward
- 9          $r_{\text{RL}}^{(i)} \leftarrow \lambda_1 r_a^{(i)} + \lambda_2 r_t^{(i)} + \lambda_3 r_l^{(i)} + \lambda_4 r_s^{(i)}$ ;
- 9         // Apply KL-regularized objective
- 10          $r_{\text{total}}^{(i)} \leftarrow r_{\text{RL}}^{(i)} - \eta \cdot \text{KL}(G_\theta \parallel G_{\theta_0})$ ;
- 10         // Update generator using PPO
- 11         Compute PPO loss:  
 $\mathcal{L} \leftarrow \text{PPO\_Loss}(r_{\text{total}}^{(i)}, G_\theta)$ ;
- 12         Update policy parameters:  $\theta \leftarrow \theta - \nabla_\theta \mathcal{L}$ ;
- 13 **return**  $G_\theta$ ;

---

instruction  $x$  and the generated STL formula  $\hat{y}$ , the overall reward is computed as:

$$r_{\text{RL}}(\hat{y}) = \lambda_1 r_a(\hat{y}) + \lambda_2 r_t(\hat{y}) + \lambda_3 r_l(\hat{y}) + \lambda_4 r_s(\hat{y}),$$

where  $r_a, r_t, r_l$ , and  $r_s$  are the reward scores corresponding to the atomic proposition alignment  $m_a$ , templated NL similarity  $m_t$ , formula conciseness  $m_l$ , and STL-level similarity  $m_s$ , proposed in Section , respectively. The hyperparameters  $\lambda_1, \lambda_2, \lambda_3, \lambda_4$  control the relative importance of each reward. The reward objective  $r_{\text{total}}$  is to maximize expected reward while constraining deviation from the initial policy  $G_{\theta_0}$ .

$$r_{\text{total}} = r_{\text{RL}}(\hat{y}) - \eta \cdot \text{KL}(G_\theta \parallel G_{\theta_0}),$$

where  $\eta$  is a KL penalty coefficient. This term stabilizes training by penalizing large shifts from the pre-trained generator.

Finally, we apply the PPO algorithm (Schulman et al. 2017) to optimize the STL generator  $G_\theta$  using the KL-regularized reward signal (Line 11 to 12).

Model	STL-DivEn			DeepSTL		
	Formula Acc.	Template Acc.	BLEU	Formula Acc.	Template Acc.	BLEU
DeepSTL	0.1986 $\pm$ 0.0182	0.1883 $\pm$ 0.0109	0.0293 $\pm$ 0.0043	0.2002 $\pm$ 0.0146	0.2916 $\pm$ 0.0172	0.3332 $\pm$ 0.0192
GPT-3.5	0.3018 $\pm$ 0.0208	0.3034 $\pm$ 0.0223	0.0424 $\pm$ 0.0182	0.2145 $\pm$ 0.0137	0.3002 $\pm$ 0.0162	0.2249 $\pm$ 0.0203
GPT-4	0.4733 $\pm$ 0.0284	0.4741 $\pm$ 0.0293	0.0831 $\pm$ 0.0196	0.2262 $\pm$ 0.0172	0.3048 $\pm$ 0.0192	0.2881 $\pm$ 0.0217
DeepSeek	0.4790 $\pm$ 0.0195	0.4825 $\pm$ 0.0206	0.0791 $\pm$ 0.0175	0.2537 $\pm$ 0.0182	0.3254 $\pm$ 0.0193	0.3982 $\pm$ 0.0233
KGST	0.5587 $\pm$ 0.0263	0.5627 $\pm$ 0.0276	0.2142 $\pm$ 0.0187	0.4538 $\pm$ 0.0207	0.4939 $\pm$ 0.0264	0.5686 $\pm$ 0.0301
<b>RESTL (Ours)</b>	<b>0.6838 <math>\pm</math> 0.0243</b>	<b>0.6974 <math>\pm</math> 0.0255</b>	<b>0.3347 <math>\pm</math> 0.0177</b>	<b>0.5985 <math>\pm</math> 0.0183</b>	<b>0.6327 <math>\pm</math> 0.0192</b>	<b>0.6783 <math>\pm</math> 0.0177</b>

Table 6: Experimental results on performance and variability measures of RESTL and baselines on STL-DivEn and DeepSTL datasets.

## C The Appendix of Evaluation Metric

**STL Formula Accuracy** Measure the alignment accuracy between the reference and prediction sequences at the string level.

**Template Accuracy** First transform the reference and prediction instances into STL templates, then calculate the alignment accuracy of the resulting template sequences.

The following example shows both the formula and its corresponding template for the reference and predicted outputs.

Formula:  $\mathbf{G}(x > 8) \rightarrow \mathbf{F}(y < 3) \Rightarrow$  Template:  $\mathbf{G}(\phi) \rightarrow \mathbf{F}(\phi)$

Formula:  $\mathbf{G}(x > 8) \rightarrow \mathbf{F}(z < 3) \Rightarrow$  Template:  $\mathbf{G}(\phi) \rightarrow \mathbf{F}(\phi)$

The formula contains 13 tokens. All tokens coincide except for the variable “y” vs “z” in the atomic proposition, resulting in  $A_F = \frac{12}{13}$ . When both formulas are converted to templates by replacing atomic propositions with placeholders (e.g.,  $\phi$ ), all tokens align perfectly:  $A_T = 1$ .

## D The Appendix of Implementation Details

Our experiments are carried out on 8 NVIDIA GeForce RTX 4090 GPUs (24GB VRAM each). We implement our framework using PyTorch (Paszke 2019) and Huggingface Transformers (Wolf et al. 2020), with LLaMA-Factory (Zheng et al. 2024) as the base for model customization. Each reward model is fine-tuned in LLaMA 3-8B with a linear value head and trained for 5 epochs using Adam optimizer (Kingma 2014), with a learning rate of 5e-5 and a batch size of 16. The STL generator is fine-tuned using PPO for 80,000 steps, with a batch size of 32, a learning rate of 1.41e-5, and a KL penalty coefficient  $\eta$  set to 0.05. The combined reward signal is computed using weighted scores with  $\lambda_1 = 0.2$ ,  $\lambda_2 = 0.25$ ,  $\lambda_3 = 0.35$ , and  $\lambda_4 = 0.2$ , where these hyperparameters control the weights of different reward components. Their initial values were assigned based on the proportional contribution of each individual metric to the model’s performance improvement. Subsequently, grid search was performed to fine-tune these weights, resulting in the final values above. We found that this combination achieves superior performance across different datasets.

## E The Appendix of Experimental Results

### E.1 Experimental Results on Variability Measures

Table 6 presents experimental results on performance and variability measures of RESTL and baseline models on the

(a) STL-DivEn dataset

Model	Formula Acc.	Template Acc.	BLEU
RESTL	<b>0.6838</b>	<b>0.6974</b>	<b>0.3347</b>
$m_a$	0.6027	0.6152	0.2824
$m_t$	0.6172	0.6137	0.2652
$m_l$	0.5952	0.6072	0.2636
$m_s$	0.6282	0.6299	0.2892
LLaMA3 (Fine-tuned)	0.4956	0.5007	0.1784

(b) DeepSTL dataset

Model	Formula Acc.	Template Acc.	BLEU
RESTL	<b>0.5985</b>	<b>0.6327</b>	<b>0.6783</b>
$m_a$	0.5203	0.5182	0.5872
$m_t$	0.5182	0.5191	0.5925
$m_l$	0.4952	0.5001	0.5782
$m_s$	0.5373	0.5407	0.6093
LLaMA3 (Fine-tuned)	0.2850	0.3285	0.5579

Table 7: Ablation analysis of different reward feedback mechanisms on each metric for STL-DivEn and DeepSTL datasets.

STL-DivEn and DeepSTL datasets. It is evident from the table that RESTL outperforms all other baselines across both datasets.

### E.2 Ablation Study Details

From Table 7, we can see that each individual reward metric contributes positively to model performance, outperforming LLaMA3 (Fine-tuned) but falling short of RESTL.

### E.3 Accuracy of Reward Models

We analyze the impact of curriculum learning on training reward models. As shown in Table 8, we evaluate the accuracy of four reward models:  $r_a$ ,  $r_t$ ,  $r_l$ , and  $r_s$  in distinguishing between “chosen” and “rejected” outputs across two datasets, and compare three curriculum ordering strategies, which are the easy-to-hard (Forward), hard-to-easy (Reverse), and random shuffle (Shuffle).

The results demonstrate that the forward ordering achieves the highest accuracy. For instance,  $r_a$  achieves 80.1% accuracy on the STL-DivEn dataset, outperforming Reverse (74.6%) and Shuffle (76.9%). This trend is observed consistently across all reward models and both datasets, sug-

Reward Model	Order	STL-DivEn	DeepSTL	Avg.
$r_a$ (AP Align)	Forward	<b>80.1</b>	<b>78.4</b>	<b>79.3</b>
	Reverse	74.6	73.1	73.9
	Shuffle	76.9	75.5	76.2
$r_t$ (NL Sim.)	Forward	<b>82.3</b>	<b>80.2</b>	<b>81.3</b>
	Reverse	76.2	74.4	75.3
	Shuffle	78.5	76.7	77.6
$r_l$ (Length)	Forward	<b>81.7</b>	<b>79.1</b>	<b>80.4</b>
	Reverse	74.1	72.5	73.3
	Shuffle	77.2	74.9	76.1
$r_s$ (STL Sim.)	Forward	<b>84.0</b>	<b>82.1</b>	<b>83.1</b>
	Reverse	77.6	75.9	76.8
	Shuffle	79.3	77.4	78.4

Table 8: Accuracy (%) of reward models under different curriculum learning orderings on STL-DivEn and DeepSTL datasets.

Prompts for STL Transformation
<p>Please transform the natural language description into an STL specification. Let <math>a</math> and <math>b</math> be two variables, and let <math>\phi</math> be the specification. The rules are as follows:</p> <ol style="list-style-type: none"> <li>1. <math>\phi 1 \ U[a,b] \ \phi 2</math> indicates that there exists a moment <math>t'</math> such that <math>\phi 1</math> holds continuously before <math>t'</math>, and <math>\phi 2</math> holds at <math>t'</math>, where <math>t' \in [a, b]</math>.</li> <li>2. <math>F[a,b]\phi</math> indicates that <math>\phi</math> holds at some point within the interval <math>[a, b]</math>.</li> <li>3. <math>G[a,b]\phi</math> indicates that <math>\phi</math> holds at every point within the interval <math>[a, b]</math>.</li> </ol> <p>Additionally, assume signals <math>x1[t], x2[t], \dots, xn[t]</math>, the atomic predicates are of the form: <math>f(x1[t], \dots, xn[t]) &gt; 0</math>.</p> <p>The STL formula should only contain atomic propositions, Boolean operators <math>\wedge, \neg, \rightarrow, \leftrightarrow</math>, and temporal operators <math>U[a,b], G[a,b], F[a,b]</math>.</p>

Figure 5: The prompts for STL Transformation.

gesting that progressively increasing task difficulty facilitates more effective stepwise learning and better capture of reward characteristics.

### F LLM Prompts for STL Transformation

We provide the prompt to LLMs to guide the transformation of natural language into STL, as shown in Figure 5.