Enhancing Transformation from Natural Language to Signal Temporal Logic Using LLMs with Diverse External Knowledge

Anonymous ACL submission

Abstract

Temporal Logic (TL), especially Signal Temporal Logic (STL), enables precise formal specification, making it widely used in cyber-physical systems such as autonomous driving and robotics. Automatically transforming NL into STL is an attractive approach to overcome the limitations of manual transformation, which is time-consuming and error-prone. However, due to the lack of datasets, automatic transformation currently faces significant challenges and has not been fully explored. In this paper, we propose a NL-STL dataset named STL-Diversity-Enhanced (STL-DivEn), comprising 16,000 samples enriched with diverse patterns. To develop the dataset, we first manually create a small-scale seed set of NL-STL pairs. Next, representative examples are identified through clustering and used to guide large language models (LLMs) in generating additional NL-STL pairs. Finally, diversity and accuracy are ensured through rigorous rule-based filters and human validation. Furthermore, we introduce the Knowledge-Guided STL Transformation (KGST) framework, a novel approach for transforming natural language into STL, involving a generate-then-refine process based on external knowledge. Statistical analysis shows that the STL-DivEn dataset exhibits more diversity than the existing NL-STL dataset. Moreover, both metric-based and human evaluations indicate that our KGST approach outperforms baseline models in transformation accuracy on STL-DivEn and DeepSTL datasets. Dataset and code will be released upon publication.

1 Introduction

Signal Temporal Logic (STL) (Maler and Ničković, 2004) provides a flexible and precise framework for specifying requirements in safety-critical cyber-physical systems. Extending Temporal Logic (TL) (Pnueli, 1977) by introducing real-time and real-valued constraints, STL can describe not only discrete temporal events but also continuous-time and real-valued dynamic changes. Therefore, STL, as a powerful expression tool for system design, offers valuable guidance in cyber-physical systems, such as autonomous driving (Maierhofer et al., 2020) and robot control (Tellex et al., 2020). But one of the main challenges in leveraging the STL specification is the need to accurately transform the potentially ambiguous and complex constraints expressed in natural language into precise STL logical expressions, as shown in the following example:

- **Natural Language:**
  Whenever the robot detects an obstacle within 1 meter in the first 60 seconds, it should move away from the obstacle and remain at least 1.5 meters away for at least 30 consecutive seconds within the next 50 seconds.
- **Signal Temporal Logic (STL):**
  $\mathbf{G}_{[0,60]}((d_{\text{obs}} < 1) \rightarrow \mathbf{F}_{[0,50]}\, \mathbf{G}_{[0,30]}\, (d_{\text{obs}} \geq 1.5))$

Writing accurate STL formulas directly is a huge burden for domain experts, as it is both time consuming and error-prone. With the development of natural language processing (NLP) technology, researchers have been experimenting with the use of NLP technology to transform natural language into TL and STL expressions, aiming to improve the accuracy of the transformation. For example, Lignos et al. (2015); Ghosh et al. (2016) use predefined pattern formulas to transform natural language sentences into intermediate representation. Subsequently, by applying a set of predefined rules manually, the intermediate representation is mapped to temporal logic formulas. These approaches require extensive domain expertise and involve a steep learning curve (Kulkarni et al., 2013). Specifically, they can only be applied to very restrictive structured natural language expressions that match with the given patterns.

In recent years, due to the great success of deep learning and Large Language Models (LLMs), increasing attention has been paid to use them to solve the transformation problem from natu-

ral language to STL. For example, DeepSTL (He et al., 2022) introduces a grammar-based synthetic data generation technique and trains an attentional translator of English to STL using a transformer-based neural translation technique. NL2TL (Chen et al., 2023) uses LLMs to help create the Natural Language-Temporal Logic dataset, which is then used to fine-tune the T5 models. However, since the dataset is generated based on predefined rules, these efforts suffer from insufficient expressive ability and diversity of expression. In addition, the transformation methods they propose also face challenges in accurately transforming complex natural language into Signal Temporal Logic.

In order to address these challenges, our efforts focus on the following two aspects. Firstly, aiming at developing high-quality and expressively diverse NL-STL datasets to deal with the scarcity of NL-STL datasets, we explore utilizing LLMs to synthesis NL-STL pairs under the guidance of prompts. However, NL-STL pairs generated by LLMs often closely resemble the examples in the prompts. To ensure diversity and comprehensiveness, we introduce a method for constructing the STL-Diversity-Enhanced (STL-DivEn) dataset. We start by handcrafting a seed set of 120 NL-STL pairs, covering both basic and nested logic to serve as the foundation for data augmentation. Next, a clustering algorithm is employed to select representative samples from the seed set. These examplars are used to guide LLMs in generating new NL-STL pairs, which are then refined using rule-based filters and human validation to ensure diversity and precision. Finally, the qualified NL-STL pairs expand the seed set and are stored in the STL-DivEn dataset.

Secondly, transformer-based models perform poorly when handling complex natural language transformation tasks. Transforming NL sentences into STL formulas remains a challenging task due to the complexity of temporal constraints in the requirements of cyber-physical systems, including nested semantics (Boufaied et al., 2021). Even many advanced models, such as GPT-4 (Achiam et al., 2023) and DeepSeek (Liu et al., 2024), while excelling at text generation tasks, still face limitations in transforming NL into STL. To address this limitation, we propose a novel transforming framework called Knowledge-Guided STL Transformation (KGST). This framework operates in two steps: first, we fine-tune an LLM on NL-STL dataset (e.g., STL-DivEn) and use the finetuned LLM to generate a preliminary STL formula from the natural language input; second, the top K similar NL-STL pairs are retrieved from the dataset, and these pairs are referenced as external knowledge; then, GPT-4 is used to evaluate and refine the preliminary STL with the external knowledge to generate the refined STL. Experimental results demonstrate that the KGST framework significantly outperforms existing baseline models in both quantitative and human evaluation metrics, showcasing its advantages in STL transformation tasks.

In general, our contributions are as follows:

- We develop a dataset, named STL-DivEn, containing 16k high-quality NL-STL pairs using LLMs and manual annotation. Compared to the existing DeepSTL dataset, the statistics show that this dataset exhibits significantly greater diversity.
- We propose a Knowledge-Guided STL Transformation (KGST) framework. It substantially improves the accuracy of the NL to STL transformations.
- The proposed KGST framework demonstrates superior performance not only on the newly developed STL-DivEn dataset but also on the existing DeepSTL dataset. This highlights its versatility and robustness across different datasets.

2 Related Work

2.1 From Natural Language to TL and STL

Many researchers have tried to transform natural language sentences into Temporal Logic formulas (Dwyer et al., 1999; Žilka, 2010; Ghosh et al., 2016; Santos et al., 2018; Cosler et al., 2023). For example, Žilka (2010) transform the properties which are specified by controlled English to TL formulas using syntax and grammatical dependency parsing techniques. Santos et al. (2018) also define a controlled natural language to specify how a system model interacts with its environment, and sentences in this controlled language are automatically transformed into TL using predefined rules. Nl2spec (Cosler et al., 2023) derives formal formulas from unstructured natural language using LLMs combined with human corrections. However, these TL-specific approaches cannot be directly applied to STL, as STL involves real-time and real-valued constraints that exceed the expressiveness of TL.

As STL is widely used in academia and industry (Madsen et al., 2018), several efforts have been made to transform natural languages into

STL (He et al., 2022; Chen et al., 2023; Mao et al., 2024; Mohammadinejad et al., 2024). For instance, DeepSTL (He et al., 2022) utilizes grammar-based techniques to synthesize data, which is then used to train Transformer models for transformation. NL2TL (Chen et al., 2023) fine-tunes T5, trained on lifted Natural Language-Temporal Logic (NL-TL) datasets created by LLMs to perform transformation. However, synthetic data generated from specific templates do not capture the full diversity of the real-world language. In addition, DialogueSTL (Mohammadinejad et al., 2024) transforms natural language task descriptions into accurate STL formulas through user interaction and reinforcement learning, but relies on user feedback, increasing the complexity of usage. To address the insufficient dataset and inefficiencies in transformation, we introduce a new comprehensive dataset and propose a framework to improve the transformation from natural languages to STL.

2.2 Instruction Dataset Construction

The generation of instruction datasets involves both manual annotation and synthesis using LLMs. Manual annotation includes designing prompts and labeling them based on human expertise (Srivastava et al., 2023; Conover et al., 2023; Zheng et al., 2023; Zhao et al., 2024; Zhou et al., 2024; Köpf et al., 2024). However, obtaining high-quality data only through manual annotation can be costly. With the growing use of LLMs, research is shifting toward generating data using LLMs, reducing reliance on manual annotation. For example, Taori et al. (2023); Wang et al. (2024); Sun et al. (2024) start with a small set of seed instructions, which are then expanded using in-context learning to generate diverse instruction-response pairs. However, these methods often struggle with ensuring sufficient diversity in the generated data. To address this, strategies such as iterative generate-filter pipelines (Wang et al., 2023) and cluster-based data selection (Köksal et al., 2024) have been proposed. Additionally, WizardLM (Xu et al., 2023) introduces an instruction evolution paradigm to enhance diversity by increasing the complexity of new instructions. In our work, the STL-DivEn dataset is created using manual annotation to generate a small set of high-quality seeds. LLMs are then used with carefully designed instructions to generate various NL-STL pairs, followed by rigorous validation to ensure consistency.

3 Signal Temporal Logic

STL is widely adopted as a specification formalism for cyber-physical systems. For example, *Automatic transmission (AT)*, a widely-used benchmark (Ernst et al., 2020, 2021, 2022), is a transmission controller of automotive systems. It continuously outputs the gear, speed and rpm of the vehicle. One of its safety requirements is as follows: *In the following 27 time units, whenever the speed is higher than 50, the rpm should be below 3000 in three time units.* STL can represent such real-time and real-valued constraints.

Let $\mathbb{R}$ denote the set of real numbers. $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_+$ represent the nonnegative and positive real numbers, respectively. Let $\mathbb{N}_+$ be the set of positive integer numbers.

Let $T \in \mathbb{R}_+$ be a positive real number, and let $d \in \mathbb{N}_+$ be a positive integer. A *d-dimensional signal* is a function $\mathbf{v}\colon [0, T] \to \mathbb{R}^d$, where T is called the *time horizon* of $\mathbf{v}$. Given an arbitrary time instant $t \in [0, T]$, $\mathbf{v}(t)$ is a d-dimensional real vector; each dimension concerns a signal *variable* that has a certain physical meaning, e.g., speed, rpm, acceleration, etc. In this paper, we fix a set X of variables and, without ambiguity, we call a variable a signal (1-dimensional signal).

Definition 1 (STL Syntax)**.** *In STL, atomic formulas α and formulas φ are inductively defined as follows:*

$$\alpha ::\equiv f(x_1, \ldots, x_K) > 0$$
$$\varphi ::\equiv \alpha \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{G}_I\varphi \mid \mathbf{F}_I\varphi \mid \varphi_1 \,\mathbf{U}_I\, \varphi_2$$

where f is a K-ary function $f : \mathbb{R}^K \to \mathbb{R}$, $x_1, \ldots, x_K \in X$, and I is a closed non-singular interval in $\mathbb{R}_{\geq 0}$, i.e., $I = [l, u]$, where $l, u \in \mathbb{R}_{\geq 0}$ and $l < u$. $\mathbf{G}, \mathbf{F}$, and $\mathbf{U}$ are temporal operators, which are known as always, eventually *and* until*, respectively. The always operator $\mathbf{G}$ and eventually the operator $\mathbf{F}$ are two special cases of the until operator $\mathbf{U}$, which can be defined by $\mathbf{F}_I\varphi \equiv \top\, \mathbf{U}_I\, \varphi$ and $\mathbf{G}_I\varphi \equiv \neg\, \mathbf{F}_I\, \neg\varphi$. Other Boolean connectives such as $\vee, \rightarrow$ are introduced as syntactic sugar, i.e., $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$.*

The *Boolean semantics* of an STL formula can be described in a satisfaction relation $(\mathbf{v}, t) \models \varphi$, which represents the signal $\mathbf{v}$ that satisfies an STL formula φ at time t:

$$\begin{aligned}
(\mathbf{v}, t) &\models \alpha && \Leftrightarrow && f(\mathbf{v}(t)) \geq 0 \\
(\mathbf{v}, t) &\models \neg\varphi && \Leftrightarrow && (\mathbf{v}, t) \not\models \varphi \\
(\mathbf{v}, t) &\models \varphi_1 \wedge \varphi_2 && \Leftrightarrow && (\mathbf{v}, t) \models \varphi_1 \wedge (\mathbf{v}, t) \models \varphi_2 \\
(\mathbf{v}, t) &\models \mathbf{G}_{[l,u]}\varphi && \Leftrightarrow && \forall t' \in [t + l, t + u].\, (\mathbf{v}, t') \models \varphi \\
(\mathbf{v}, t) &\models \mathbf{F}_{[l,u]}\varphi && \Leftrightarrow && \exists t' \in [t + l, t + u].\, (\mathbf{v}, t') \models \varphi
\end{aligned}$$

$$(\mathbf{v}, t) \models \varphi_1 \, \mathbf{U}_{[l,u]} \, \varphi_2 \quad \Leftrightarrow \quad \exists t' \in [t+l, t+u].\, (\mathbf{v}, t') \models \varphi_2 \wedge \forall t'' \in [t, t'].\, (\mathbf{v}, t') \models \varphi_1$$

Now, we can formally specify the above *AT safety requirement* by the following STL formula:

$$\mathbf{G}_{[0,27]}(\mathsf{speed} > 50 \rightarrow \mathbf{F}_{[1,3]}(\mathsf{rpm} < 3000)).$$

Note that *nested* STL formula refers to an STL formula where temporal operators are applied within the scope of other temporal operators.

4 Approach

In this section, we first present our approach for constructing the STL-Diversity-Enhanced (STL-DivEn) dataset, which combines manual annotation and LLMs to generate diverse, high-quality data. Second, we introduce the Knowledge-Guided STL Transformation (KGST) framework to further enhance performance in STL transformation.

4.1 Dataset Construction

To build a comprehensive and diverse NL-STL dataset, we follow the steps below: 1) Seed Selection: Manually create an initial set of NL-STL pairs and use clustering algorithm to identify representative seeds, 2) Diversity-Guided Augmentation: Utilize the identified seeds as diverse examples to guide GPT-4 (gpt-4-0125-preview) for augmentation in generating new NL-STL pairs, 3) Quality Assurance: Apply rule-based filters to remove low-quality pairs and human validation to verify semantic consistency, and 4) Dataset Expansion: Add qualified pairs to the seed set and store them in the STL-DivEn database. This pipeline is illustrated in Figure 1.

Seed Selection. Signal Temporal Logic encompasses a variety of complex applications. Without high-quality seeds, the generated data may lack diversity. Therefore, the first step is to build a seed set, which includes natural language descriptions and corresponding STL formulas covering both nested and basic logic, as well as applications in fields such as autonomous driving, robotics, and electronics. To ensure both comprehensiveness and accuracy, these initial NL-STL pairs are manually created. The seed set is created by 6 domain experts, two from each field, resulting in a total of 120 NL-STL pairs, with 40 pairs from each field.

When using GPT-4 to generate new NL-STL pairs, selecting appropriate examples is crucial as the generated NL-STL pairs tend to mimic the provided examples. To ensure diversity, we employ the k-means (Hartigan et al., 1979) to cluster five centers from the seed set, and then use these centers as examples to guide the GPT-4 in data augmentation.

We use the Sentence-Transformers (Reimers and Gurevych, 2019) to map NL-STL pairs into a high-dimensional vector space, determining the cluster centers. This approach prevents any single category of NL-STL pairs from dominating the generated data.

Diversity-Guided Augmentation. After selecting the most representative NL-STL pairs, the next step is to generate new NL-STL pairs based on these seeds to expand the dataset. The five chosen NL-STL instruction seeds are used as input examples for GPT-4, with evolution prompts guiding GPT-4 to generate new NL-STL pairs. The prompts can be found in the Appendix A.2.

Quality Assurance. Since GPT-4 may produce incorrect NL-STL pairs, including those with syntax errors, redundancy with the seed set, or inaccurate semantics, we employ rule-based filtering and human validation to ensure the quality of the dataset.

In detail, rule-based filtering is applied in two stages. The first stage applies the syntax check algorithm to eliminate NL-STL pairs that do not adhere to the syntax rules outlined in Section 3. Each NL-STL pair is then compared to the existing data in the seed set by calculating their Rouge scores (Lin, 2004). If the Rouge score between a new NL-STL pair and all existing seed pairs is below 0.5, the new pair is considered to exhibit sufficient diversity.

Next, the NL-STL pairs that pass the rule-based filtering undergo human validation to ensure consistency between the natural language and STL specifications. Seven annotators who have been trained in STL usage and expressions spend two months conducting the annotation.

Dataset Expansion. To continuously enhance data diversity, NL-STL pairs filtered through rule-based filtering and human validation are added to the seed set as candidates for guiding the next generation. These pairs are also incorporated into the STL-DivEn dataset, which is organized in a structured format that links natural language expressions to their corresponding STL formulas.

4.2 Applying LLMs to Generate Formulas

To enable LLMs to utilize the acquired knowledge more effectively, we structure the NL-STL transformation task as a generate-then-refine process, as shown in Figure 2.

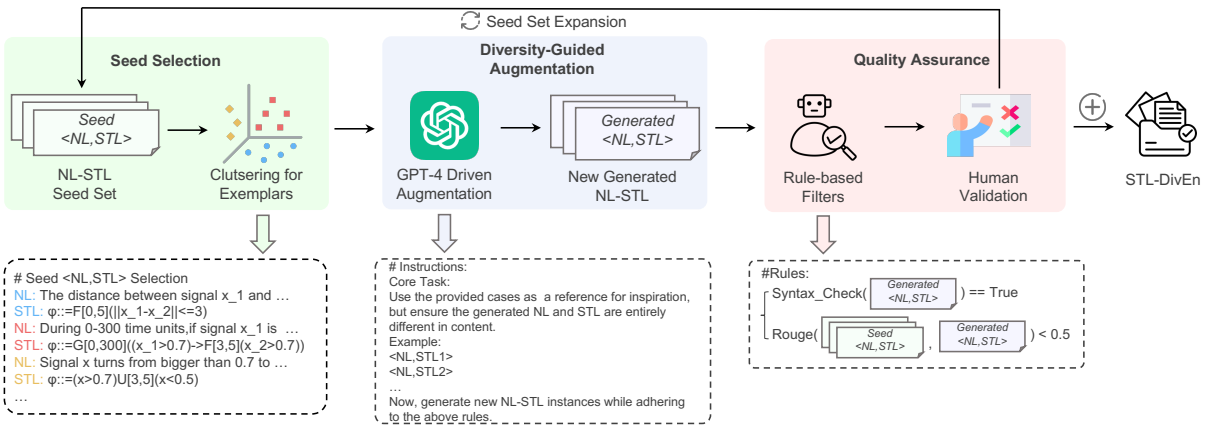

Figure 1: The pipeline of STL-DivEn construction. We first handcraft a set of seed NL-STL pairs. Next, representative NL-STL pairs are selected by clustering to guide GPT-4 in data augmentation. The newly generated NL-STL pairs pass through rule-based filters and human validation. Finally, verified pairs are added to the STL-DivEn dataset and seed set for the next round generation.

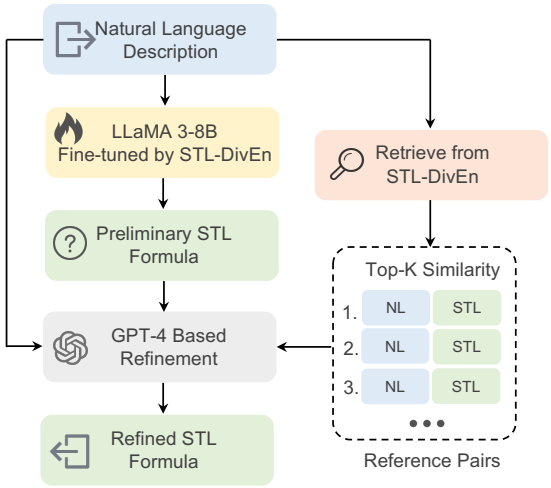

Figure 2: Architecture of Knowledge Guide STL Transformation (KGST).

Specifically, we first fine-tune LLMs such as LLaMA 3-8B on STL-DivEn, enabling them to transform natural language descriptions into preliminary STL formulas.

Next, GPT-4 is employed to refine the preliminary STL formula. Specifically, we select the top K most similar NL-STL pairs from external knowledge (e.g., STL-DivEn) as reference pairs based on the input natural language description using a similarity algorithm, where K is set to 5. These reference pairs, along with the original natural language description and the preliminary STL formula, are then fed into GPT-4.

Finally, GPT-4 evaluates and refines the preliminary STL formula based on the reference pairs, generating the refined STL formula. The prompts used for this process are detailed in Appendix A.3.

5 Experiments

In this section, we conduct experiments on our proposed dataset and the existing benchmark proposed (He et al., 2022) to evaluate our methods.

5.1 Experiment Settings

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

Datasets. We conduct experiments on two NL-STL datasets, including DeepSTL (He et al., 2022) and the proposed STL-DivEn Dataset. Specifically, DeepSTL generates STL formulas through randomly sampling from templates and operator distributions, while STL-DivEn is a dataset created using GPT-4 and human annotation. We randomly selected 14,000 samples from each dataset for the training set and 2,000 samples for the test set.

Evaluation Measures. To evaluate the results of STL generation, we utilize both quantitative metrics and human evaluation in our experiment. In detail, we use three evaluation metrics: STL Formula Accuracy, Template Accuracy (He et al., 2022), and BLEU (Papineni et al., 2002), which are used for the STL generation task. STL Formula Accuracy emphasizes strict alignment of symbols and syntax, Template Accuracy evaluates the completeness of logical structures, and BLEU assesses local semantics and phrase-level similarity. The calculation methods for STL Formula Accuracy and Template Accuracy are provided in Appendix B.

For human evaluation, we randomly selected 100 NL-STL pairs from the test set of STL-DivEn and DeepSTL. Five annotators (all students who have

grasped the usage of STL formulas) are required to compare our model with baseline models. They are unaware of which STL formulas are generated by our model and which are generated by the baseline models. The annotators evaluate whether the STL formula faithfully reflects the natural language description in four aspects: whether the operators in the STL are correct, whether the values are accurate, whether the generated STL conforms to the syntax rules, and whether the semantics are consistent with the natural language description. The evaluation results are labeled as correct only when all aspects are correct; otherwise, they are marked as incorrect if any aspect is wrong.

Baselines and Implementation Details. We conduct the comparison experiments using five baseline methods: DeepSTL, GPT-3.5[1], GPT-4[2], DeepSeek (Liu et al., 2024), and Self-Refine (Madaan et al., 2024). In our experiments, the GPT-4 version is "gpt-4-0125-preview", the GPT-3.5 version is "gpt-3.5-turbo-1106", and the DeepSeek version is "DeepSeek-V3". The Self-Refine method involves GPT-4 generating an initial STL formula, followed by refinement using GPT-4's own knowledge. DeepSTL uses the Adam optimizer (Kingma, 2014) and is trained with the Transformers model architecture. KGST is fine-tuned on LLaMA 3-8B and utilizes GPT-4 for refinement with external knowledge, which is derived from the corresponding training set. Details on hyperparameter determination are provided in Appendix C.

5.2 Experimental Results

In this section, we show our experimental results on the two datasets STL-DivEn and DeepSTL.

5.2.1 Metric-Based Evaluation

The quantitative evaluation results on the STL-DivEn and DeepSTL datasets are shown in Table 1. For the STL-DivEn dataset, our model performs the best (Table 1a). Across the three metrics, our model achieves scores of 0.5587 for STL Formula Accuracy, 0.5627 for Template Accuracy, and 0.2142 for BLEU, surpassing other models. For example, DeepSeek obtains 0.4790, 0.4852, and 0.0791, while GPT-4 obtains 0.4733, 0.4741, and 0.1931 for the respective metrics.

For the DeepSTL dataset, as shown in Table 1b, we also observe that our model achieves the highest scores. It obtains 0.4538 for STL Formula Accuracy, 0.4939 for Template Accuracy, and 0.5686 for BLEU, outperforming all other models. Specifically, DeepSeek obtains 0.2537, 0.3254, and 0.3982, while GPT-4 obtains 0.2262, 0.3048, and 0.2882 for the respective metrics.

Furthermore, we observe a decrease in the performance of the Self-Refine method after refinement. This suggests that refining STL formulas requires external knowledge rather than relying solely on the model's internal capabilities. In conclusion, our KGST model demonstrates superior performance in generating more accurate STL formulas compared to the baseline models.

Model	STL Formula Accuracy	Template Accuracy	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
GPT-4+Self-Refine	0.4422	0.4466	0.0521
KGST	**0.5587**	**0.5627**	**0.2142**

(a) STL-DivEn

Model	STL Formula Accuracy	Template Accuracy	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
GPT-4+Self-Refine	0.2203	0.3019	0.2682
KGST	**0.4538**	**0.4939**	**0.5686**

(b) DeepSTL

Table 1: Metric-based evaluation results.

5.2.2 Human Evaluation

The human evaluation results are shown in Table 2. We use the correctness percentage as a comprehensive evaluation of operator correctness, value accuracy, semantic consistency, and syntax conformity in generated STL formulas. From the results, it can be observed that the evaluators consider the proportion of correct STL formulas generated by our model to be the highest among all methods. For example, on the STL-DivEn dataset, the accuracy of our model is 62.4%, validating the effectiveness of our KGST model.

5.3 Analysis

5.3.1 Corpus Statistics

Table 3 presents the statistics for the DeepSTL and STL-DivEn datasets. Specifically, Table 3a provides statistics on the STL formulas, including sub-

[1]https://platform.openai.com/docs/models/gpt-3-5-turbo

[2]https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4

Model	Accuracy (%)	
	STL-DivEn	DeepSTL
DeepSTL	43.4	42.0
GPT-3.5	48.4	45.6
GPT-4	53.0	48.8
DeepSeek	55.0	49.2
GPT-4+Self-Refine	51.2	47.0
KGST	**62.4**	**54.6**

Table 2: Human evaluation results.

Dataset	#subformula per formula		#STL oper. per formula		#N-gram diversity
	avg.	median	avg.	median	
DeepSTL	6.98	7	6.98	7	1.474
STL-DivEn	14.66	14	20.04	19	2.386

(a) STL formula statistics: # subformula for each STL formula, # operators for each STL formula and # N-gram diversity of STL formulas.

Dataset	#sent.	#word	#words per sent.		# N-gram d“ 1‘iversity
			avg.	median	
DeepSTL	120,000	265	38.49	37	1.132
STL-DivEn	16,000	4,954	35.83	35	2.424

(b) Natural language descriptions statistics: # unique sentences, # unique words, # words per sentences and # N-gram diversity of natural language descriptions.

Dataset	#char per identifier		#digits per constant		#identifiers per formula
	avg.	median	avg.	median	
DeepSTL	5.50	5	2.31	2	2.59
STL-DivEn	2.63	2	1.70	2	7.2

(c) Identifier and constants statistics: # chars used per identifier, # number of digits used per constant and # average number of identifiers per formula.

Table 3: Dataset statistical analysis of DeepSTL and STL-DivEn.

formulas, STL operators, and the N-gram diversity of all STL formulas. A subformula is defined as any well-formed part of a formula that constitutes a complete expression. Table 3b displays statistics for the natural language descriptions, such as the total number of unique sentences, the number of unique words, the average number of words per sentence, and the N-gram diversity of all descriptions. Meanwhile, Table 3c shows the frequency of identifiers and constants.

The numbers of subformulas and operators in each STL formula indicates that the formulas in the STL-DivEn dataset have more complex structures. The total word count of 4,954 unique words in STL-DivEn, compared to only 265 words in DeepSTL, highlights the richer vocabulary in the STL-DivEn dataset. Additionally, both the N-gram diversity of the STL formulas and the natural language descriptions demonstrate a greater level of diversity in STL-DivEn. In conclusion, STL-DivEn is a comprehensive and diverse dataset, making it a valuable resource for further research.

Model	STL Formula Accuracy	Template Accuracy	BLEU
KGST	0.5587	0.5627	0.2142
- w/o Fine-tuning	0.5360	0.5390	0.1978
- w/o Refinement	0.4956	0.5007	0.1784

Table 4: Ablation experimental results on STL-DivEn.

5.3.2 Ablation Study

To validate the effectiveness of the fine-tuning and refinement modules, we conduct ablation experiments on STL-DivEn, with results shown in Table 4. KGST w/o Refinement indicates the KGST model with the Refine module removed, where STL is generated solely by fine-tuning the LLMs. The results show that when STL is generated using only the fine-tuned LLMs, the metrics are higher than those of the baseline models but lower than those of the complete KGST model. KGST w/o Fine-tuning indicates the KGST model with the fine-tuning module removed, where STL is generated using only the top five high-similarity NL-STL pairs retrieved from external knowledge as references. Compared to the complete KGST model, all metrics show a decrease, but still higher than those of the baseline models. Therefore, we conclude that both fine-tuning and refinement play active roles in STL generation.

5.3.3 Case Study

To intuitively demonstrate how KGST improves the quality of STL generation, we present a case study in Table 5. In this study, we compare the STL formulas generated by KGST with those generated by GPT-4 and the fine-tuned LLaMA 3-8B model. In Case 1, according to the natural language description, $x_3 > 2$ must occur within 2 to 4 time units in the future. However, GPT-4 incorrectly uses $\mathbf{F}_{[2,4]}(x_3 > 2)$ to express a logical "until", which is not accurate. On the other hand, while the syntax of LLaMA 3-8B is not fully compliant (e.g., it does not explicitly use $\mathbf{G}_{[20,50]}$ to indicate the global time interval constraint), its basic logic is correct. In Case 2, both GPT-4 and LLaMA 3-8B use incorrect syntax for the triggering condition. The correct expression should use the global operator $\mathbf{G}_{[0,500]}$ to specify that the triggering condition must be monitored across the entire time interval, rather than at a specific point in time. Furthermore, in

Case 1:
NL (STL-DivEn): Between time 20 and 50, the sum of signals x_1 and x_2 must not exceed 1.5, unless within 2 to 4 time units later, x_3 exceeds 2.
GPT4: $\mathbf{G}_{[20,50]}(x_1 + x_2 \leq 1.5 \rightarrow \mathbf{F}_{[2,4]}(x_3 > 2))$ **LLaMA 3-8B (Finetuned):** $20 \leq t \leq 50 \rightarrow ((x_1 + x_2 \leq 1.5)\ \mathbf{U}_{[2,4]}\ (x_3 > 2)$ **KGST:** $\mathbf{G}_{[20,50]}((x_1[t] + x_2[t] \leq 1.5)\ \mathbf{U}_{[2,4]}\ (x_3[t] > 2))$
Ground Truth: $\mathbf{G}_{[20,50]}((x_1[t] + x_2[t] \leq 1.5)\ \mathbf{U}_{[2,4]}\ (x_3[t] > 2))$
Case 2:
NL (STL-DivEn): Whenever signal z_2 falls below -0.5 or exceeds 0.5 within 0 to 500 time units, signal z_1 must exceed 1 within the next 200 time units and maintain that level for at least 50 time units.
GPT4: $\mathbf{F}_{[0,500]}(z_2[t] < -0.5 \vee z_2[t] > 0.5) \rightarrow (\mathbf{F}_{[0,200]}(z_1 > 1) \wedge \mathbf{G}_{[0,50]}(z_1 > 1))$ **LLaMA 3-8B (Finetuned):** $\mathbf{F}_{[0,500]}((z_2[t] < -0.5 \vee z_2[t] > 0.5) \rightarrow \mathbf{F}_{[0,200]}\ \mathbf{G}_{[0,50]}\ (z_1 > 1))$ **KGST:** $\mathbf{G}_{[0,500]}((z_2[t] < -0.5 \vee z_2[t] > 0.5) \rightarrow \mathbf{F}_{[0,200]}\ \mathbf{G}_{[0,50]}\ (z_1[t] > 1))$
Ground Truth: $\mathbf{G}_{[0,500]}((\lvert z_2 \rvert > 0.5) \rightarrow (\mathbf{F}_{[0,200]}\ \mathbf{G}_{[0,50]}\ (z_1 > 1)))$

Table 5: Generated STL formulas from different models on STL-DivEn.

the formula generated by GPT-4, $\mathbf{F}_{[0,200]}(z_1 > 1)$ and $\mathbf{G}_{[0,50]}(z_1 > 1)$ are used in parallel, but there is no indication of the sequential relationship. The correct logic should specify that $z_1 > 1$ must first occur, followed by its persistence for 50 time units. These results confirm that KGST effectively corrects errors in the generated STL, such as misused operators or invalid syntax.

5.3.4 Impact of Refinement

To validate the impact of refinement on specific error types, we track four types of errors in 100 generated STL formulas: incorrect operator usage, value errors, syntax violations, and semantic inconsistencies with the corresponding NL. The differences before and after the refinement process are shown in Figure 3, and it is observed that the frequencies of all error types have decreased.

We also conduct an experimental analysis of the iteration rounds by calculating the STL Formula Accuracy, Template Accuracy, and Bleu score for different numbers of refinement iterations on STL-DivEn. Figure 4 shows that as the number of iterations increases, there is no significant impact on the effect of refinement, because each iteration uses the same NL-STL as the reference.

5.3.5 Scaling Effect

Figure 5 presents the results of the scaling effect experiments on the STL-DivEn dataset. It illustrates how STL Formula Accuracy changes as the dataset

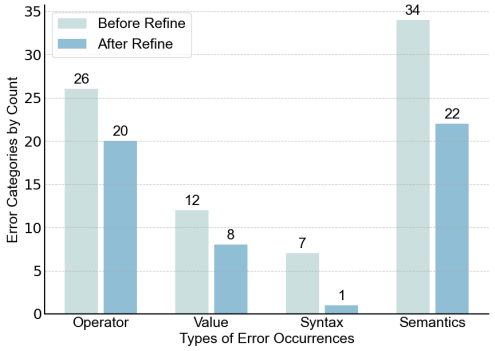

Figure 3: Tracking errors before and after refinement.

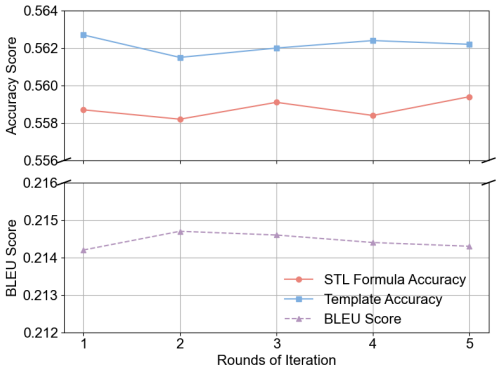

Figure 4: Impact of iteration rounds on refinement.

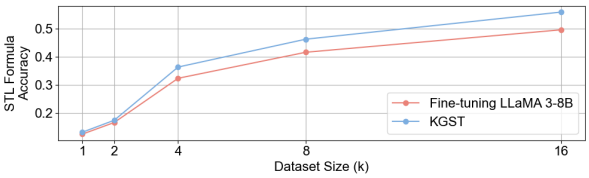

Figure 5: Scaling effect of STL-DivEn dataset on STL formula accuracy.

size increases. Both the fine-tuning and KGST show gradual improvement with the growth of the dataset, with KGST consistently outperforming the fine-tuning across all dataset sizes, particularly on larger datasets. The performance on other evaluation metrics can be found in Appendix D.

6 Conclusion

In this work, we present a new dataset, STL-DivEn, which features NL-STL pairs with enhanced diversity. Additionally, we introduce the KGST framework, a novel approach for transforming natural languages into STL. Results from both metric-based evaluations and human evaluations demonstrate that our approach significantly improves transformation capabilities across two datasets. Our approach facilitates the automatic extraction of temporal and continuous constraints in cyber-physical systems, supporting efficient and reliable modeling to ensure the safety and robustness.

Limitations

Our dataset is currently built using GPT-4 rather than directly derive from requirement documents of real-world cyber-physical systems. Although we have already guided GPT-4 to generate diverse NL-STL pairs, it may still not fully cover the temporal property patterns of real-world cyber-physical systems, or the dataset may be biased. This may limit the effectiveness and accuracy of our model when applied to real-world cyber-physical systems.

To address this issue, at least the following approaches can be considered in the future. First, we can extract temporal property patterns from existing real-world cyber-physical systems. Second, for specific domains like autonomous driving, we can extract necessary data from domain-related requirements documentation, e.g., international standards related to AUTOSAR for electronic vehicles. Furthermore, we can infer possible timing properties and other temporal characteristics of cyber-physical systems by simulating their real interactive environments. In this way, our dataset can be continuously enriched by incorporating human validation to train better models.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Chaima Boufaied, Maris Jukss, Domenico Bianculli, Lionel Claude Briand, and Yago Isasi Parache. 2021. Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization. *J. Syst. Softw.*, 174:110881.

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. 2023. Nl2tl: Transforming natural languages to temporal logics using large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15880–15903.

Mike Conover, Matt Hayes, Ankit Mathur, Xiangrui Meng, Jianwei Xie, Jun Wan, Ali Ghodsi, Patrick Wendell, and Matei Zaharia. 2023. Hello dolly: Democratizing the magic of chatgpt with open models. *Databricks blog. March*, 24.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *CAV 2023*, volume 13965 of *LNCS*, pages 383–396. Springer.

Matthew B Dwyer, George S Avrunin, and James C Corbett. 1999. Patterns in property specifications for finite-state verification. In *ICSE 1999*, pages 411–420.

Gidon Ernst, Paolo Arcaini, Ismail Bennani, Aniruddh Chandratre, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Khouloud Gaaloul, Jun Inoue, Tanmay Khandait, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Masaki Waga, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. 2021. ARCH-COMP 2021 category report: Falsification with validation of results. In *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 133–152. EasyChair.

Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. 2020. ARCH-COMP 2020 category report: Falsification. In *7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 140–152.

Gidon Ernst, Paolo Arcaini, Georgios Fainekos, Federico Formica, Jun Inoue, Tanmay Khandait, Mohammad Mahdi Mahboob, Claudio Menghi, Giulia Pedrielli, Masaki Waga, Yoriyuki Yamagata, and Zhenya Zhang. 2022. ARCH-COMP 2022 category report: Falsification with ubounded resources. In *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90 of *EPiC Series in Computing*, pages 204–221. EasyChair.

Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. 2016. Arsenal: automatic requirements specification extraction from natural language. In *NFM 2016*, pages 41–46. Springer.

John A Hartigan, Manchek A Wong, et al. 1979. A k-means clustering algorithm. *Applied statistics*, 28(1):100–108.

Jie He, Ezio Bartocci, Dejan Nickovic, Haris Isakovic, and Radu Grosu. 2022. Deepstl - from english requirements to signal temporal logic. In *ICSE 2022*, pages 610–622. ACM.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Abdullatif Köksal, Timo Schick, Anna Korhonen, and Hinrich Schuetze. 2024. Longform: Effective instruction tuning with reverse instructions. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. 2024. Openassistant

conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36.

Dhanashree Kulkarni, Andrew N Fisher, and Chris J Myers. 2013. A new assertion property language for analog/mixed-signal circuits. In *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, pages 1–8. IEEE.

Constantine Lignos, Vasumathi Raman, Cameron Finucane, Mitchell P. Marcus, and Hadas Kress-Gazit. 2015. Provably correct reactive control from natural language. *Auton. Robots*, 38(1):89–105.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Curtis Madsen, Prashant Vaidyanathan, Sadra Sadraddini, Cristian Ioan Vasile, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta. 2018. Metrics for signal temporal logic formulae. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 1542–1547. IEEE.

Sebastian Maierhofer, Anna-Katharina Rettinger, Eva Charlotte Mayer, and Matthias Althoff. 2020. Formalization of interstate traffic rules in temporal logic. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 752–759. IEEE.

Oded Maler and Dejan Ničković. 2004. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT 2004*, volume 3253 of *LNCS*, pages 152–166. Springer.

Yuchen Mao, Tianci Zhang, Xu Cao, Zhongyao Chen, Xinkai Liang, Bochen Xu, and Hao Fang. 2024. Nl2stl: Transformation from logic natural language to signal temporal logics using llama2. In *2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, pages 469–474. IEEE.

Sara Mohammadinejad, Sheryl Paul, Yuan Xia, Vidisha Kudalkar, Jesse Thomason, and Jyotirmoy V Deshmukh. 2024. Systematic translation from natural language robot task descriptions to stl. In *International Conference on Bridging the Gap between AI and Reality*, pages 259–276. Springer.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL 2002*, pages 311–318.

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

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Tainã Santos, Gustavo Carvalho, and Augusto Sampaio. 2018. Formal modelling of environment restrictions from natural-language requirements. In *SBMF 2018*, pages 252–270. Springer.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. 2024. Principle-driven self-alignment of language models from scratch with minimal human supervision. *Advances in Neural Information Processing Systems*, 36.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. 2020. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):25–55.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *ACL 2023*.

Zifeng Wang, Chun-Liang Li, Vincent Perot, Long Le, Jin Miao, Zizhao Zhang, Chen-Yu Lee, and Tomas Pfister. 2024. Codeclm: Aligning language models with tailored synthetic data. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3712–3729.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. In *ICLR 2024*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, et al. 2023. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *ICLR 2023*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

Lukáš Žilka. 2010. *Temporal logic for man*. Ph.D. thesis, Master's thesis, Brno University of Technology.

A Prompts input to Large Language Models

In this section, we present the prompts designed to guide large language models.

A.1 Prompts for GPT-4 to generate NL-STL pairs

Dataset Construction Prompt:

I am constructing a dataset that pairs Natural Language descriptions with their corresponding Signal Temporal Logic (STL) expressions.
Please generate three unique instances in each request.
Use the provided cases as a reference for inspiration, but ensure the generated NL and STL are completely different in content.
{Example_Pairs}
Now, generate new NL-STL instances while adhering to the above rules.
The format for each generated pair must adhere strictly to the following format:
NL:[Natural Language Description]
STL:[Signal Temporal Logic Expression].

Figure 6: Evolution Prompts for GPT-4 in NL-STL Pairs Generation.

Figure 6 shows the prompt used for GPT-4 to generate NL-STL pairs. The example pairs are selected from the seed set using a clustering algorithm.

A.2 Prompts for LLMs to Generate STL

STL Generation Prompt:

Please translate the Natural Language into STL specification.
Let a and b be two variables, and let φ be the specification. The rules are as follows:
1. $\varphi_1 U[a,b] \varphi_2$ indicates that there exists a moment t' such that φ_1 is satisfied before t', and φ_2 is satisfied at t', where t' is within a time distance of a to b from the current moment.
2. F[a,b]φ indicates that there exists a point within the interval [a, b] where φ is satisfied.
3. G[a,b]φ indicates that φ is satisfied at every point within the interval [a, b].
Additionally, assume signals x1[t], x2[t], . . . , xn[t], then atomic predicates are of the form:f(x1[t], . . . , xn[t]) > 0.
The STL formula should only contain atomic propositions, boolean operators &, ~, ->, <-> and temporal operators U[a,b], G[a,b], F[a,b].

Figure 7: The prompt for Baseline Models to generate STL formulas.

Figure 7 shows the prompts used for baseline models, including GPT-3.5, GPT-4, and DeepSeek, to generate STL formulas from input natural language descriptions.

A.3 Prompts for Refinement Part in KGST

KGST Prompt:

Given the input natural language description {Input_Natural_Language}
and the preliminary STL formula {Preliminary_STL}.
Validate and refine the STL formula using the following five most similar NL-STL pairs from external knowledge: {Reference_Pairs}.
Ensure that the refined STL accurately captures the intended meaning.
Correct any inconsistencies to improve clarity and precision.
Refined STL:

Figure 8: The prompts in the refinement part of KGST.

Figure 8 shows the prompts used for KGST to refine the preliminary STL. Reference pairs refer to the top K NL-STL pairs selected from external knowledge based on their similarity to the transformed natural language.

Feedback Prompt:

The following STL specification was generated from a natural language description.
Please review the STL formula for correctness, clarity, and adherence to the following rules:\n
1. Temporal operators should include U[a,b], F[a,b], and G[a,b].
2. Use atomic predicates in the form of f(x1[t], ..., xn[t]) > 0.
3. Boolean operators should be limited to &, ~, ->, and <->.
4. Ensure the STL formula accurately represents the intent of the natural language description.
Identify any errors, ambiguities, or improvements needed.
Natural Language: {Input_Natural_Language}
Preliminary STL: {Preliminary_STL}
Feedback:

Figure 9: The prompts in the feedback part of Self-Refine.

A.4 Prompts for Self-Refine

Refiner Prompt:
Based on the provided feedback, refine the STL specification to address the identified issues.
Ensure that the updated STL formula:
1. Correctly reflects the original natural language intent.
2. Follows the syntax rules for STL with appropriate temporal and boolean operators.
3. Improves clarity, correctness, and logical consistency.\n\n"
Natural Language: {Input_Natural_Language}
Preliminary STL: {Preliminary_STL}
Feedback: {Feedback}
Refined STL:

Figure 10: The prompts in the refinement part of Self-Refine.

Figure 9 shows the prompts used for GPT-4 to generate feedback on whether the STL is correct based on the STL generation criteria for the given natural language input and its corresponding STL. Figure 10 shows the prompts used for GPT-4 to refine the preliminary STL based on the feedback.

A.5 Prompts for KGST w/o Finetune

KGST w/o Finetune Prompt:
Given the input natural language description {Input_Natural_Language}
Generate the STL formula refering the following five most similar NL-STL pairs : {Reference_Pairs}.
Ensure that the generatedSTL accurately captures the intended meaning.
Generated STL:

Figure 11: The prompts for KGST w/o Finetune to generate STL.

Figure 11 shows the prompts used for GPT-4 to generate STL based on the input natural language description and the top K NL-STL pairs retrieved from external knowledge with the highest similarity to the input, which serve as reference pairs in the context.

B Evaluation Metrics

STL formula accuracy (A_F) and template accuracy (A_T). The first metric measures the alignment accuracy between the reference and predicted sequences at the string level, while the second metric involves transforming both the reference and predicted instances into STL templates and then calculating their alignment accuracy. For example:

Formula: eventually $(a < 5) \Rightarrow$ Template : G (ϕ)

Formula: eventually $(b < 5) \Rightarrow$ Template : G (ϕ)

The first line represents the reference sequence, and the second line corresponds to the model's prediction. To illustrate more clearly, spaces are inserted between each token, resulting in six tokens in the formula and four tokens in the template. In the formula, five tokens appear in the same positions—'G', '(', '$<$', '5', ')'—while the remaining token 'a' in the reference is mistranslated as 'b'. Therefore, the formula accuracy (A_F) is calculated as:

$$A_F = \frac{5}{6}$$

For the template, since all tokens align perfectly, the template accuracy (A_T) equals:

$$A_T = 1$$

C Details of Implementation

The experiments are conducted on eight NVIDIA 4090 GPUs, with all implementations utilizing PyTorch[3], LLaMA-Factory[4], and Huggingface's Transformers[5]. To ensure efficient training, the learning rate is set to 5e-5 and the batch size is 16. To ensure the adequacy of the training results, the model is run for 10 epochs under each setting.

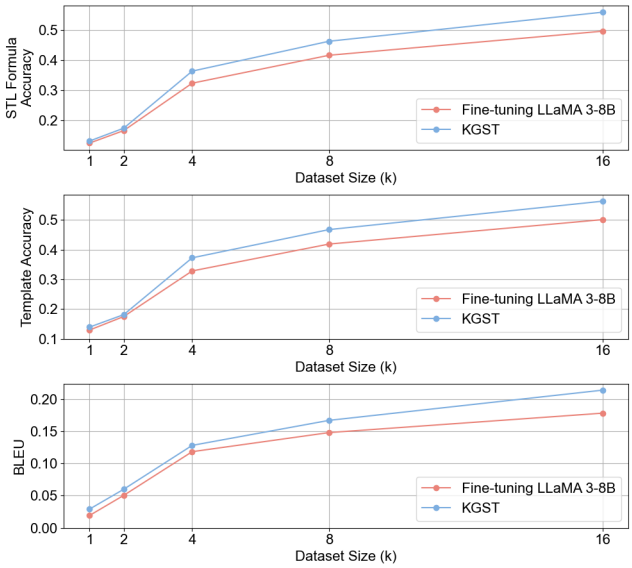

Figure 12: Scaling effect of STL-DivEn on three evaluation metrics.

D Scaling Effect of Multi-Metrics

Figure 12 shows the scaling effect of the STL-DivEn dataset, illustrating the performance metrics of STL generation after fine-tuning with Llama-3-8B on the STL-DivEn dataset, as well as the performance of KGST in generating STL formulas. The metrics include STL formula accuracy, template accuracy, and BLEU score, as the dataset size increases from 1k to 16k.

[3]https://pytorch.org/

[4]https://github.com/hiyouga/LLaMA-Factory

[5]https://github.com/huggingface/transformers