NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models

Anonymous ACL submission

Abstract

Temporal Logic (TL) can be used to rigorously specify complex high-level specification for systems in many engineering applications. The translation between natural language (NL) and TL has been under-explored due to the lack of dataset and generalizable model across different application domains. In this paper, we propose an accurate and generalizable transformation framework of English instructions from NL to TL, exploring the use of Large Language Models (LLMs) at multiple stages. Our contributions are twofold. First, we develop a framework to create a dataset of NL-TL pairs combining LLMs and human annotation. We publish a dataset with 23K NL-TL pairs. Then, we finetune T5 models on the lifted versions (i.e., the specific Atomic Propositions (AP) are hidden) of the NL and TL. The enhanced generalizability originates from two aspects: 1) Usage of lifted NL-TL characterizes common logical structures, without constraints of specific domains. 2) Application of LLMs in dataset creation largely enhances corpus richness. We test the generalization of trained models on five varied domains. To achieve full NL-TL transformation, we either combine the lifted model with AP recognition task or do the further finetuning on each specific domain. During the further finetuning, our model achieves higher accuracy (> 95%) using only <10% training data, compared with the baseline sequence to sequence (Seq2Seq) model.[1]

1 Introduction

Temporal Logic (TL) has been widely used as a mathematically precise language to specify requirements in many engineering domains such as robotics (Tellex et al., 2020), electronics design (Browne et al., 1986), autonomous driving (Maierhofer et al., 2020). TL can capture the complex spatial, temporal, and logical requirements in both human languages and environmental constraints, and can be transformed into executable actions or control inputs for robots (Gundana and Kress-Gazit, 2022; Raman et al., 2013; Boteanu et al., 2016; Patel et al., 2020; Gopalan et al., 2018).

Unlike many robotics works that try to use end-to-end black-box models to infer robotic behaviors directly from natural language (NL) (Ahn et al., 2022), using structured TL as the intermediate has a twofold benefit – the TL can be used for direct planning, and the TL representation can be used to identify specific sources of failure and provide automatic feedback to a non-expert user (Raman et al., 2013). However, TL has a steep learning curve. Communicating one's goals and constraints through NL is much more intuitive to a non-expert. Therefore, a model able to transform NL instructions into TL is a missing but crucial component for interactive robots and engineering designs.

Currently, there is no general tool to perform automated translations between TL and NL that takes the following requirements into consideration:

- **Cross-domain generalization.** Although TL is used in many engineering domains, current NL-to-TL approaches largely constrain their training data to a single domain. These datasets mostly lack plentiful corpus richness of NL-TL and have their own specified formats of Atomic Propositions (AP). Then the models fail to generalize to other domains (Gopalan et al., 2018), even though the structure of TL itself is not dependent on the domain and should be generic.

- **Variability of NL instructions.** Past work often constructs synthetic data algorithmically, causing limited forms of the NL input. Real-world NL utterances cannot be encoded into a small set of rules. Models trained on such homogeneous data fail to generalize to new sentence structures (Brunello et al., 2019).

[1] Datasets and Codes are available

One big bottleneck in the NL-to-TL problem is the lack of data. Although modern statistical methods can outperform rule-based methods (Buzhinsky, 2019), they typically require a huge dataset. This data is expensive and difficult to collect since strong expertise of annotators is needed (Brunello et al., 2019). As outlined above, constraining the domain or form of the NL instructions relieves the pressure of dataset, but also unavoidably undermines the generalizability (Brunello et al., 2019; Patel et al., 2019).

To supplement the data creation process and simultaneously overcome the need for a huge dataset, we propose to use pre-trained LLMs. We utilize GPT-3 (Brown et al., 2020) to assist dataset creation and finetune T5 models (Raffel et al., 2020) to be specialized in NL-to-TL transformation.

Another aspect of our approach is to use 'lifted' versions of NL and TL for finetuning our model, which greatly enhances generalizability. In previous work, models trained on full NL-to-TL transformation often include converting specific individual actions into APs. For example, the AP *"a response is created in Slack"* might be formalized as *"create_Slack"*. As a result, each work has to regularize its own content and style of APs, affecting generalization. Instead of taking this approach, we hide all the APs in our data during finetuning, acquiring a lifted model on lifted NL-to-TL transformation. For the final ground application from full NL into full TL, two methods are proposed, either by combining the lifted model with AP recognition or further transfer learning in one specific domain. For further transfer learning into specific domains, we compare the models with/without pre-training on lifted NL-TL and show its significance.

In this paper, we present two key contributions:

- **Constructing a cross-domian NL-TL dataset.** We generate a dataset of 10K lifted NL-TL pairs using a novel GPT-3-assisted framework. Ablation studies are conducted to show the significance of each part of the framework for dataset construction. In addition, we collect and clean previous datasets (13K) from two varied domains, adapting original full NL-TL pairs into lifted versions. In this way, we publish a dataset of 23K lifted NL-TL pairs. Ablation studies show that the newly created data are indispensable since purely training on the collected data fails to work across domains.
- **Finetuning an lifted NL-to-TL model** on T5 using our data, and demonstrating the improvement in performance compared to former state-of-the-art methods. For application in full NL-to-STL transformation, two methods are proposed. We compare our model to Seq2Seq models and direct few-shot learning by GPT-3, across five domains. The experimental results show that our methods achieve better accuracy (>95% across all domains) and are more data-efficient (<10% domain specific data). We also do the ablation study by training a Seq2Seq model with lifted NL-to-TL dataset, revealing that T5's superior model capacity is essential.

2 Temporal Logic Specifications

2.1 STL Syntax

There are many different versions of TL (Emerson, 1990; Maler and Nickovic, 2004; Koymans, 1990). They are more or less similar in terms of syntax. We will use Signal Temporal Logic (STL) as a representative formal language that supports constraints over the continuous real-time, which is more suitable to capture time-critical missions. In some previous work, Linear Temporal Logic (LTL) is also widely used, which is contained by STL when the time is discrete. We will construct our framework based on STL and show that the trained model also performs well on datasets and tasks using LTL. An STL formula is defined recursively according to the following syntax:

$$\phi ::= \pi^{\mu} \mid \neg\phi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \mathbf{F}_{[a,b]}\phi \mid \mathbf{G}_{[a,b]}\phi \mid \phi \mathbf{U}_{[a,b]}\varphi \quad (1)$$

where ϕ and φ are STL formulas, and π^{μ} is an atomic predicate. $\neg$ (negation), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (imply), and $\Leftrightarrow$ (equal)) are logical operators. $\mathbf{F}_{[a,b]}$ (eventually/finally), $\mathbf{G}_{[a,b]}$ (always/globally), and $\mathbf{U}_{[a,b]}$ (until) are temporal operators with real-time constraints $t \in [a, b]$. Temporal operators with time constraints are illustrated by Table 4, and other operators can be presented using the basic syntax.

2.2 Lifted STL and Lifted NL

We represent our data as 'lifted' NL and STL, in which the specific APs corresponding to individual actions are hidden (following nomenclature from Hsiung et al. (2021)). In our lifted NL and STL,

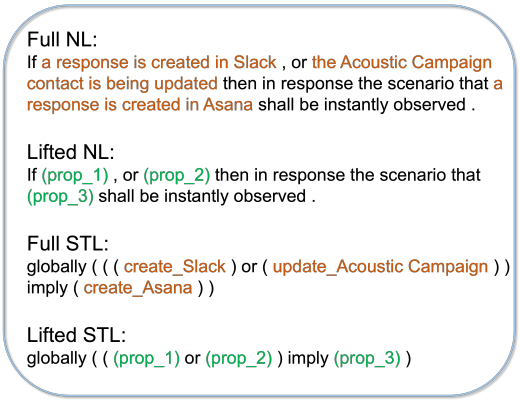

Figure 1: Illustration of lifted NL and lifted STL.

each AP is replaced with a placeholder *prop_i*. In this way, we train our model on the general context of the instruction regardless of the specific APs. The correspondences between full and lifted NL/STL are shown in Figure 1.

2.3 STL Expression Formats

Consider an STL expression as a binary tree, as in Figure 2. When finetuning a text-to-text model, there are different ways of representing the target STL in the form of linear text. Specifically, the targeted tokens can be linearized in an in-order (left subtree, root, right subtree) or pre-order (root, left subtree, right subtree) manner. Meanwhile, the operators can also be represented as the words with their corresponding meanings (rather than as symbols). The training results show that the in-order expression with all the operators replaced by words achieves much better accuracy than other three forms (will discuss in the following Section 4.2).

3 Related Work

Over decades, researchers have methods to translate English sentences into various TL formulae (Brunello et al., 2019; Finucane et al., 2010; Tellex et al., 2020; Raman et al., 2013). However, to simplify the tasks, some previous work typically make strong assumptions to restrict the input text or the output formula, thus limiting the flexibility and generalizability.

The first representative attempt is by Finucane et al. (2010); Tellex et al. (2011); Howard et al. (2014), where the traditional methods typically follow three steps: 1) pre-process given English input by extracting syntactical information, 2) identify patterns or rules for TL through classification, and 3) run an attribute grammar-based parser to derive a target logical format. These methods only work for restricted input NL (Tellex et al., 2020).

Another category of approaches are learning-based. Representative state-of-the-art works are Gopalan et al. (2018); Wang et al. (2021); He et al. (2022). In Gopalan et al. (2018) the authors gather a dataset focusing on Geometric LTL (GLTL), in which the NL and GLTL examples are all for the navigation of a car in the room. Then Seq2Seq models with attention mechanism are trained. Though the accuracy (93.45%) is satisfying, the used GLTLs are relatively simple with each GLTL normally including one to three APs and the dataset also focuses on one confined task. In He et al. (2022) the authors choose to first translate a manually generated set of STL formulae into English sentences and train a semantic parser on the synthetic data. Such synthetic data cannot represent general NL and therefore the trained parser only works well on the original STL formulae.

In Wang et al. (2021) a semantic parser is built to learn the latent structure of NL commands for ground robots. The parser will provide (potentially incorrect) intermediate LTL representations to a motion planner, and the planner will give an executed trajectory as the feedback to see whether the robot's execution satisfies the English input. Such approach has no guarantee on the correctness of the translated TL. In recent months, the work by Fuggitti and Chakraborti (2023) directly applies LLMs like GPT-3 to convert NL to LTL via few-shot learning. The prompts should be well designed and the model will fail once the NL and LTL structures are too complex (we will discuss it in Section 5.1).

These years, starting from the attention mechanism (Vaswani et al., 2017), the rapid progression of pre-trained LLMs in NLP tends to unify many previous seemingly independent tasks into one large pre-trained model, especially the GPT series from OpenAI (Brown et al., 2020), and T5 (Raffel et al., 2020) and PaLM (Chowdhery et al., 2022) from Google. These models are pre-trained with large amounts of natural sentences and codes, intrinsically encoding much logical knowledge (Creswell et al., 2022). This inspires us the new opportunity in NL-to-TL task.

4 Approach

There are 3 steps in our approach. First, generating lifted NL-STL dataset with LLMs. Second, finetuning LLMs to get high accuracy on lifted NL-STL

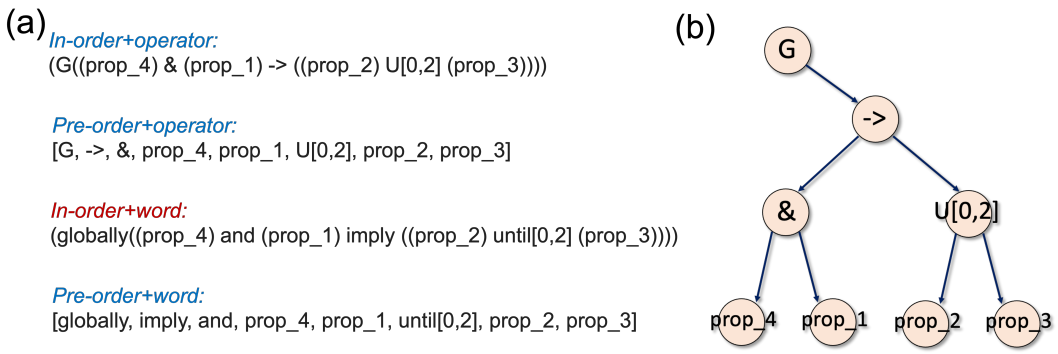

Figure 2: Illustration of different formats of STL expressions. (a) Different expression formats of the same STL. (b) The binary tree representation of STL.

transformation. Third, lifting the data and applying the lifted model. Finally, we also consider the case where lifting is not possible and we must translate end to end by further finetuning the model.

4.1 Data Generation

Algorithm 1 Algorithm for STL synthesis

Input:
Maximum number of APs N
Output:
Synthesized pre-order STL

$two\_subtree$ = [$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\mathbf{U}$, $\mathbf{U}_{[a,b]}$]
$one\_subtree$ = [$\neg$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{F}_{[a,b]}$, $\mathbf{G}_{[a,b]}$]
sub_lists $\leftarrow$ Random prop_list with total length [1, N] ▷ [prop_3, prop_1], [prop_2]
Each sub_list $\leftarrow$ insert operators in $one\_subtree$ + $two\_subtree$ ▷ [$\Leftrightarrow$, $\neg$, prop_3, prop_1], [$\mathbf{G}$, prop_2]
Assembling sub_lists into pre-order STL by appending random $two\_subtree$ operators ▷ [$\mathbf{U}_{[10,30]}$, $\Leftrightarrow$, $\neg$, prop_3, prop_1, $\mathbf{G}$, prop_2]

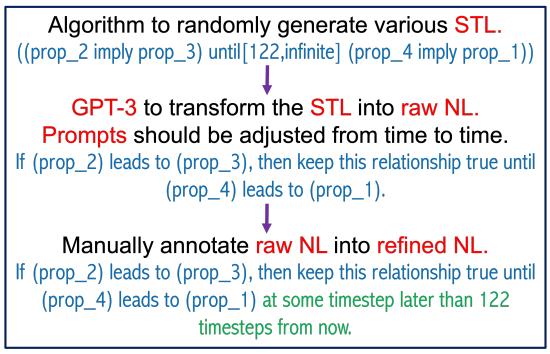

Figure 3: Framework1 to generate NL-STL pairs.

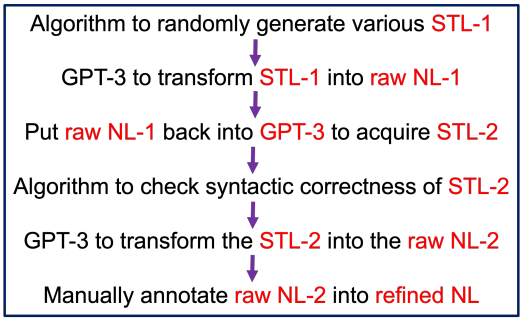

Figure 4: Framework2 to generate NL-STL pairs. One extra loop between NL and STL is added.

We apply the LLM GPT-3 (Davinci-003) to help generate multiple lifted NL and STL pairs. The first intuitive method is to use various NL-STL pairs as prompts and ask GPT-3 to automatically generate more NL-STL pairs. However, it turns out that the model always generates STL and NL with similar syntactic structures as the given prompts, thus limiting the sentence richness. To stimulate GPT-3 to generate sentences with more variations, we ask it to generate corresponding NLs from different STLs. The whole framework (referred as Framework1) is shown in Figure 3. Various pre-order STLs are randomly synthesized by binary tree generation algorithm (See Algorithm 1 and specific discussion in Appendix B). The pre-order STLs are then transformed into in-order expressions via rules. To make the input STL more understandable to GPT-3, the operators are represented with the words of their meanings ($\Rightarrow$ $(imply)$, $\Leftrightarrow$ $(equal)$, $\vee(or)$, etc). Then the GPT-3 will try to generate the raw NL whose semantic meaning is close to the STL. Human annotators then modify the raw NL to make its meaning consistent with the STL. During this process, the NL-STL pairs in prompts will be randomly chosen to make the vocabulary and sen-

tence structure more diversified. We gather 200 NL instructions from 10 volunteers who are familiar with robot tasks and randomly choose 100 NL to serve as the prompt pool, while the other 100 NL serve as the extra testing data. In each iteration of Framework1, 20 pairs are randomly chosen from the prompt pool to form the prompt of GPT-3 (a prompt example is shown in Appendix C.1 and the discussion on how many examples should be included in GPT-3 prompt is shown in Appendix D).

While Framework1 enhances the sentence richness, one issue is that the pure rule-based synthesis of STL sometimes generates unreasonable semantic meanings, or that the STL is too complex to describe it with NL. To solve this problem, an optimized framework (referred as Framework2) is shown in Figure 4. Compared to Framework1, an extra loop between STL and NL is added using GPT-3. In this way, the initial rule-based STL with unreasonable or complex meanings will be automatically filtered by GPT-3 itself. In other words, during the mapping from STL-1 to NL-1, GPT-3 more or less modifies the meanings of the STLs that it cannot fully translate. Then the translated NL-1, though not fully consistent with STL-1, is more reasonable in the view of humans. It turns out that the semantic meanings synthesized by Framework2 are closer to the common human languages, and NL-STL pairs own much less errors to annotate. The average number of annotated pairs is about 80 per person per hour with Framework1, and about 120 per person per hour with Framework2.

We in total create 9867 lifted NL-STL pairs combining both Framework1 and Framework2, with the whole cost of around 100 person-hours. Appendix C.2 shows a prompt example to transform from NL-1 back into STL-2 via GPT-3, and Appendix E shows some example annotations of lifted NL-STL pairs.

Apart from synthesizing and annotating lifted NL-STL pairs with GPT-3, we also collect and annotate the data gathered from Wang et al. (2021) and He et al. (2022). Wang et al. (2021) focuses on robot navigation task with LTL, and He et al. (2022) focuses on circuit controlling with STL. To clean and process the data into lifted NL-STL pairs, the APs in both two datasets are detected and hidden by combining hard-coded algorithms with entity recognition package SpaCy (Honnibal and Montani, 2017). We gather 5K lifted NL-STL pairs from Navigation dataset (Wang et al., 2021) and

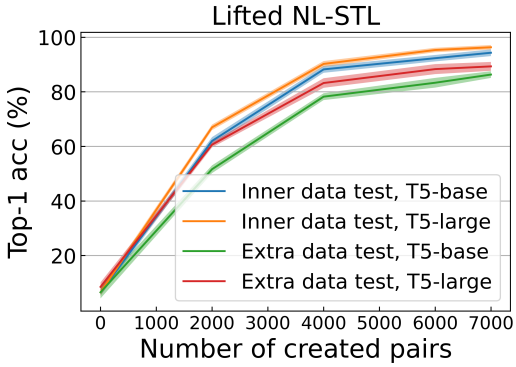

Figure 5: Testing accuracy VS. Number of created NL-STL pairs. The data collected from Navigation and Circuit work are all used during training. The inner data refers to the data generated with the help of GPT-3, and the extra data refers to the instructions collected from volunteers. The figure shows the necessity of the created data.

8K lifted NL-STL pairs from the Circuit dataset (He et al., 2022). Note that the original Navigation dataset uses LTL, while we correct some expression formats to form into STL. The original Circuit dataset contains 120K NL-STL pairs, while we find including 8K examples into our dataset is informative enough to cover the whole corpus richness of Circuit dataset.

Hence, in this work a dataset with in total about 23K lifted NL-STL pairs are created. Appendix I.1 shows the statistics of this lifted NL-STL dataset.

4.2 Model Finetuning

We mainly apply the T5 model (Raffel et al., 2020) to serve as the base LLM for finetuning. To study whether model sizes will influence the performance, T5-base (220M) and T5-large (770M) are both finetuned on the same data. The setting is illustrated in Appendix K.

Though the corrected data from Navigation and Circuit studies already provide multiple examples, these datasets only cover limited conditions and lack generalization. To show the necessity of the newly created data, the T5 models are finetuned with varied number of created NL-STL pairs, as shown in Figure 5. During training, all the data collected from Navigation and Circuit studies are used and the number of created data are varied among different models. The trained models are then tested with either the created data (referred as inner data test) or the NL instructions collected from volunteers (referred as extra data test). Since

	T5-base	T5-large
P.O./word	$70.00 \pm 1.42\%$	$73.10 \pm 1.05\%$
I.O./word	$\mathbf{96.43} \pm \mathbf{0.72}\%$	$\mathbf{97.52} \pm \mathbf{0.65}\%$
P.O./opera.	$72.35 \pm 1.54\%$	$71.95 \pm 1.23\%$
I.O./opera.	$89.94 \pm 0.89\%$	$88.17 \pm 1.02\%$

Table 1: Accuracy of inner data testing for training data with different expression formats. P.O. and I.O. represent Pre-order and In-order, respectively.

minor difference in STLs can cause severe difference in the real meanings, we apply the binary accuracy as the metric, i.e., 100% right or not. We find that the Top-1 testing accuracy will increase greatly with increasing created pairs, with the highest accuracy 97.52% and 90.12% of inner and extra testing, respectively.

Table 1 presents the experimental results with the targeted STL of different formats as discussed in Section 2.3. We find that using the in-order format plus replacing operators with words will largely improve the performance. In-order format is more consistent with natural sentence expressions and lowers the difficulty for finetuning an LLM. This result is different from former conclusions when training Seq2Seq model for NL to STL/LTL tasks, where the pre-order format is better because it naturally avoids the issue of parentheses matching (Wang et al., 2021).

4.3 Ablation Studies

Human Annotation To reveal the significance of human annotation, we train the model with the same amount of raw pairs created by GPT-3 and test them on corrected data. The results are shown in Appendix G.1. We find that annotated dataset can improve the testing accuracy by around 10%.

Framework2 To reveal the significance of the data generated by Framework2, we train the model with either the same amount of data from pure Framework1 or the data combining two frameworks. Utilizing both frameworks improves the accuracy by around 2% (Appendix G.2).

Model Capacity of T5 To reveal the significance of T5's superior model capacity, we train a Seq2Seq model on the same lifted NL-STL dataset for comparison, as shown in Appendix H.

5 Application

Right now we have finetuned T5 model to convert lifted NL into lifted STL. For the real applications, we need one model to convert from full NL to full STL, in which the format of APs should be regularized. To reach this destination, we will display two methods in the following discussion, and compare them with other state-of-the-art models. We test on five datasets across domains Circuit (He et al., 2022), Navigation (Wang et al., 2021), Office email (Fuggitti and Chakraborti, 2023), GLTL (Gopalan et al., 2018; Tellex et al., 2020), and CW (Squire et al., 2015). The examples of full NL-STL pairs in each domain are shown in Appendix F, and the statistics of each dataset are shown in Appendix I.2. Note that some lifted NL-STL pairs in Circuit and Navigation datasets have been used during training the lifted model, while all the full NL-STL pairs have not been seen. All the data in other three domains are independent of the finetuning in lifted models. Our model achieves higher accuracy on full NL-STL transformation with much less training data across all these domains.

5.1 Lifted Model + GPT-3 AP Recognition

In the real applications, we have to formulate how the APs are presented in STL (like 'verb_noun') so that the specified APs can directly connect with controllers. As shown in Appendix L, we directly utilize GPT-3 to recognize APs in the sentence and hide them as *"prop_i"*. Then the lifted model will predict the targeted lifted STL and the hidden APs will be swapped into formatted form to generate the full STL.

Table 2 displays the performance accuracy of this method. We test on three distinct domains and compare with the GPT-3 end-to-end method, i.e., using GPT-3 to directly transform NL into STL. The GPT-3 end-to-end method is proposed by Fuggitti and Chakraborti (2023) recently, aiming to generalize into all various domains. However, in the NL to STL/LTL task, finetuning on a much smaller LLM like T5 is still greatly better than direct few-shot learning on state-of-the-art LLM like GPT-3. The experimental results show that combining finetuned lifted model with AP recognition using GPT-3 can lead to a full task accuracy over 95% across all three tested domains. Table 3 displays the performance of detecting APs with GPT-3. Compared to the direct NL to STL task, AP detection task is much easier to GPT-3. Hence,

	Circuit	Navigation	Office email
GPT-3 end-to-end	$38.25 \pm 6.51\%$	$50.51 \pm 5.08\%$	$58.73 \pm 4.86\%$
T5-large + GPT-3 AP detect	$\mathbf{95.13} \pm \mathbf{1.42}\%$	$\mathbf{95.03} \pm \mathbf{1.20}\%$	$\mathbf{96.73} \pm \mathbf{1.03}\%$
T5-base + GPT-3 AP detect	$94.61 \pm 0.74\%$	$94.73 \pm 1.02\%$	$96.08 \pm 0.97\%$

Table 2: Testing accuracy of full NL-to-STL task for each grounding model. The testing domains are: Circuit (He et al., 2022), Navigation (Wang et al., 2021), Office email (Fuggitti and Chakraborti, 2023).

	Circuit	Navigation	Office email
GPT-3 AP detect accuracy	$98.84 \pm 0.41\%$	$99.03 \pm 0.53\%$	$100.00 \pm 0.00\%$

Table 3: Testing accuracy of recognizing APs with GPT-3 for each domain.

dividing the whole task into AP recognition and semantic parsing are more data-efficient and flexible than pure end-to-end method.

To further test model performance under varied sentence complexity, we plot the testing accuracy VS. the number of APs in Appendix J. As the number of APs in each lifted STL increases, the accuracy of GPT-3 few-shot learning decreases, while the finetuned T5-large model still performs well.

5.2 Transfer Learning

On the condition that we know how the users define the representation of APs, the aforementioned method is suitable to predict the full STL. On the other hand, there are also the condition that we cannot acquire the specific hard-coded rules to formulate AP representation, but only full NL-STL pairs. In these cases, the direct further finetuning may help. In other words, the lifted model has learnt to parse the semantic logical relations, and the further transfer learning is to learn how the APs are regulated in this specific dataset. This direct end-to-end transfer learning serves as the second way for ground applications.

To show that our method is generalizable and data-efficient, we compare our methods to the original Seq2Seq methods implemented in each dataset. Specifically, in the Circuit dataset the authors train the model from the ground using Transformer architecture (Vaswani et al., 2017), and in GLTL and CW datasets the authors implement recurrent neural network (RNN) encoder-decoder framework with Gated Recurrent Unit (GRU) as the core RNN cell. As the work on Navigation dataset uses the final task completion rate as the criterion not the direct LTL accuracy, the LTL prediction accuracy is inherently low. For a fair comparison in Navigation dataset, we implements the same Seq2Seq framework as that in GLTL and CW datasets.

The experimental results are shown in Figure 6. Here the training data number means how many full NL-STL pairs are used during transfer learning or Seq2Seq training. The blue curve represents the accuracy where T5 model first pre-trained on 23K lifted NL-STL pairs, and then finetuned on full NL-STL examples in that domain. The orange curve represents the condtion when T5 model is not pre-trained by lifted NL-STL pairs, but directly fine-tuned based on initial released weights. Compared to the original Seq2Seq model proposed in each dataset, transfer learning with LLM is much more efficient, and the pre-training on lifted NL-STL pairs also displays a great saving on training data requirements. We also find that T5-large model performs better than T5-base model. In all the three domains, the T5-large model with lifted NL-STL pre-training can achieve an accuracy near 95% with only 200 to 500 full NL-STL examples. This amount of example requirement is one magnitude less than the Seq2Seq baselines.

The CW dataset is somewhat unique since it only has 36 different LTLs, meaning there are on average 50 different NLs corresponding to the same LTL. The study in Gopalan et al. (2018) applies this dataset to test the generalizability of the models within the domain. They use some types of LTLs as the training examples for transfer learning, and the left types of LTLs as the testing set. This is to test whether the model can predict the LTLs that it has not seen during the training. We also carry out this experiment and compare with the method in the original paper. As shown in Figure 7, the LLM with finetuning is apparently better than the original baseline.

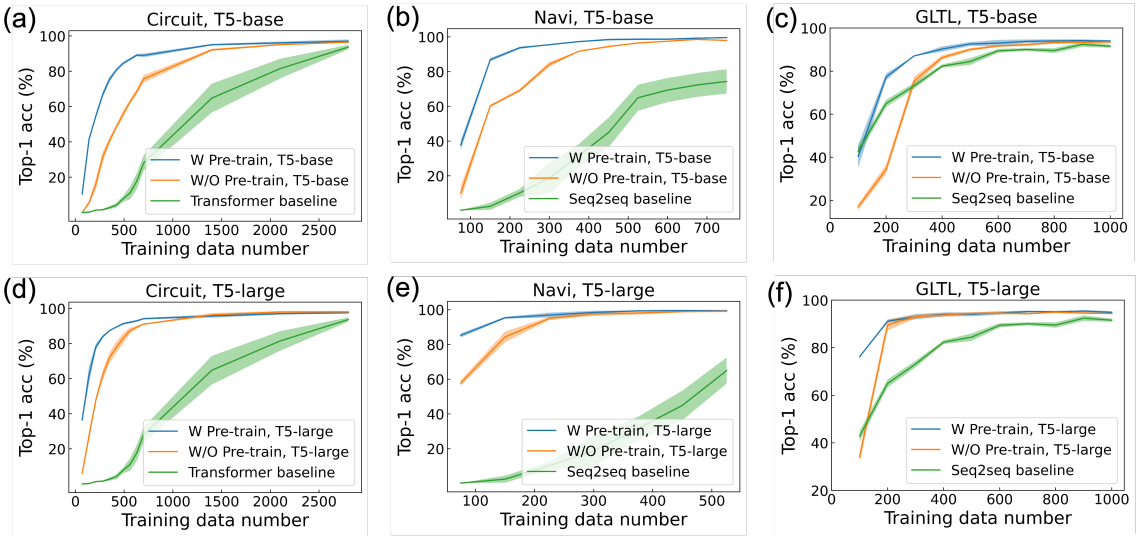

Figure 6: Experimental results for end-to-end transfer learning on Circuit, Navigation, and GLTL datasets.

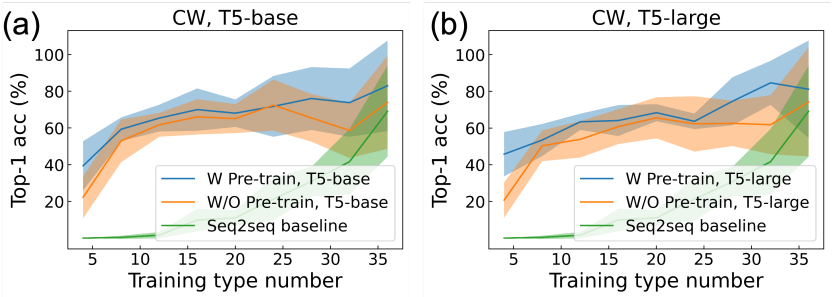

Figure 7: Experimental results for end-to-end transfer learning on CW datasets. This experiment is to compare generalizability of our method with the original state-of-the-art Seq2Seq method.

6 Limitation

In spoken language, coreference is quite common, such as *"pick up the apple and then bring it to me"*. Here *"apple"* and *"it"* refer to the same object. In the five datasets we collected and tested, the coreference problem is not severe since most NL do not have pronouns. For further work, the NER models specialized in dissolving coreferences and converting them into normal APs are needed for more unbounded input sentences.

The evaluation metric here is pure binary accuracy (fully correct or not). Actually, it is quite difficult to judge the similarity or distance of two TLs. Simply calculating token matching or computing truth values both own drawbacks. A more effective metric is needed.

The output of LLMs may sometimes generate incorrect TLs. We build up rule-based methods to check syntactic correctness and correct errors like parentheses matching. Further work can be added to improve output correctness by modifying training procedures and loss functions.

7 Conclusion

We propose a framework to achieve NL-to-TL transformation with the assistance of LLM, from aspects of both data generation and model training. One dataset with about 23K lifted NL-TL pairs is then constructed by which the T5 model is finetuned. Two approaches are implemented to utilize the trained model into full NL-to-TL translation. Experimental results on five varied domains display much better accuracy and generalizablity compared to original methods. The created dataset can be used to train future NL-to-TL models and serve as the benchmark. The proposed framework to finetune LLMs with lifted NL-TL pairs makes it possible for generalizable NL-to-TL translation without the constraints of domains and input instruction structures. As future work, we believe the model can be improved with larger dataset containing more diversified corpus.

Acknowledgements

We thank the help from the volunteers for contributing the natural language instructions.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*.

Adrian Boteanu, Thomas Howard, Jacob Arkin, and Hadas Kress-Gazit. 2016. A model for verifiable grounding and execution of complex natural language instructions. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2649–2654. IEEE.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Michael C. Browne, Edmund M. Clarke, David L. Dill, and Bud Mishra. 1986. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, 35(12):1035–1044.

Andrea Brunello, Angelo Montanari, and Mark Reynolds. 2019. Synthesis of ltl formulas from natural language texts: State of the art and research directions. In *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Igor Buzhinsky. 2019. Formalization of natural language requirements into temporal logics: a survey.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*.

E Allen Emerson. 1990. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier.

Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. 2010. Ltlmop: Experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1988–1993. IEEE.

Francesco Fuggitti and Tathagata Chakraborti. 2023. NL2LTL – a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *AAAI*. System Demonstration.

Nakul Gopalan, Dilip Arumugam, Lawson LS Wong, and Stefanie Tellex. 2018. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, volume 2018.

David Gundana and Hadas Kress-Gazit. 2022. Event-based signal temporal logic tasks: Execution and feedback in complex environments. *IEEE Robotics and Automation Letters*, 7(4):10001–10008.

Jie He, Ezio Bartocci, Dejan Ničković, Haris Isakovic, and Radu Grosu. 2022. Deepstl: from english requirements to signal temporal logic. In *Proceedings of the 44th International Conference on Software Engineering*, pages 610–622.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 7(1):411–420.

Thomas M Howard, Stefanie Tellex, and Nicholas Roy. 2014. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6652–6659. IEEE.

Eric Hsiung, Hiloni Mehta, Junchi Chu, Xinyu Liu, Roma Patel, Stefanie Tellex, and George Konidaris. 2021. Generalizing to new domains by mapping natural language to lifted ltl. *arXiv preprint arXiv:2110.05603*.

Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299.

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.

Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer.

Roma Patel, Ellie Pavlick, and Stefanie Tellex. 2019. Learning to ground language to temporal logical form. In *NAACL*.

Roma Patel, Ellie Pavlick, and Stefanie Tellex. 2020. Grounding language to non-markovian tasks with no supervision of task specifications. In *Robotics: Science and Systems*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton CT Lee, Mitchell P Marcus, and Hadas Kress-Gazit. 2013. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: science and systems*, volume 2, pages 2–1. Citeseer.

Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. 2015. Grounding english commands to reward functions. In *Robotics: Science and Systems*.

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).

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Approaching the symbol grounding problem with probabilistic graphical models. *AI magazine*, 32(4):64–76.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Christopher Wang, Candace Ross, Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2021. Learning a natural-language to ltl executable semantic parser for grounded robotics. In *Conference on Robot Learning*, pages 1706–1718. PMLR.

A STL illustration

$\mathbf{F}_{[a,b]}\phi$	True at time t if there exists a time in the interval $[t+a, t+b]$ where ϕ is true.
$\phi\mathbf{U}_{[a,b]}\varphi$	True at time t if φ is true for some time $t^{'}$ in the interval $[t+a, t+b]$, and for all times between t and $t^{'}$, the formula ϕ holds.
$\mathbf{G}_{[a,b]}\phi$	True at time t if for all times in the interval $[t+a, t+b]$, the formula ϕ holds.

Table 4: STL illustration

B Full algorithm to synthesize multiple STLs

This is the full algorithm to synthesize multiple varied STLs. The blue-colored words are the example output in each step. All the operators are classified into the operator with only one subtree, or the operator with two subtrees. A random ordered $prop$ list is generated with the length less than the upper limit. Then this full list is split into some sub_lists. For each sub_list, operators are randomly appended in the left side until each $prop$ occupy one position in the binary tree. Then these modified sub_lists are assembled back into the full STL by appending operators with two subtrees. The STL generated in this way are syntactically correct, but may own some flaws in semantic meanings. Some rules can be pre-set to avoid the unreasonable conditions, e.g., two negation operation should not appear continually.

Algorithm 2 Full algorithm for STL synthesis

Input:

Maximum number of APs N

Output:

Synthesized pre-order STL

$two\_subtree$ = [$\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\mathbf{U}$, $\mathbf{U}_{[a,b]}$],
$one\_subtree$ = [$\neg$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{F}_{[a,b]}$, $\mathbf{G}_{[a,b]}$]
AP_num = random.randint(1, N) ▷ e.g., 3
prop_list ← Random_ordered Prop list with length AP_num
sub_lists ← randomly divide prop_list ▷ e.g., [prop_3, prop_1], [prop_2]

Creating sub-STLs :
for each sub_list **do**
    num_open_subtree = len(sub_list)
    while num_open_subtree > 1 **do**
        operation ← randomly choose item in $two\_subtree + one\_subtree$
        if operation in $two\_subtree$ **then**
            num_open_subtree -= 1
        end if
        if operation in [$\mathbf{U}_{[a,b]}$, $\mathbf{F}_{[a,b]}$, $\mathbf{G}_{[a,b]}$] **then**
            a, b ← sampling random integers or denoting as $infinity$
        end if
        sub_list.insert(0, operation)
    end while
    save sub_list as sub_STL ▷ e.g., [$\Leftrightarrow$, $\neg$, prop_3, prop_1], [$\mathbf{G}$, prop_2]
end for

Assembling sub-STLs :
Assembling sub_STLs into pre-order STL by appending random $two\_subtree$ operations ▷ e.g., [$\mathbf{U}_{[10,30]}$, $\Leftrightarrow$, $\neg$, prop_3, prop_1, $\mathbf{G}$, prop_2]

C Examples of prompt input to GPT-3

These are the example prompts for GPT-3 to convert between NL and STL, or detect the spans of Atomic Proportions.

Try to transform the signal temporal logic into natural languages, the operators in the signal temporal logic are: negation, imply, and, equal, until, globally, finally, or .
The examples are as following:

STL: (finally [317,767] (prop_3 equal prop_2) imply prop_1)
natural language: It is required that at a certain point within the next 317 to 767 time units the scenario in which (prop_3) is equivalent to the scenario in which (prop_2) happens , and only then (prop_1) .

STL: (((prop_2 until [417,741] prop_3) imply prop_1) or prop_4)
natural language: (prop_2) should happen and hold until at a certain time point during the 417 to 741 time units the scenario that (prop_3) should happen then (prop_1) , or else (prop_4) .

STL: (prop_1 imply finally [300, infinite] prop_2)
natural language: If (prop_1) happens, then some time after the next 300 time steps (prop_2) should happen.

.
.
.

STL: (prop_1 imply (negation prop_2 and (prop_3 until prop_4)))
natural language: If (prop_1), don't (prop_2), instead keep (prop_3) until (prop_4).

STL: (globally [0, 354] prop_1 and (prop_2 imply (globally [0, 521] (prop_3) and finally [521, 996] prop_4)))
natural language: Stay (prop_1) for 354 timesteps, and if (prop_2) happens, then first keep (prop_3) and then let (prop_4) happen at some point during 521 to 996 time steps.

STL: ((prop_1 until prop_3) and (prop_4 imply prop_2))
natural language: Do (prop_1) until (prop_3), but once (prop_4) occurs then immediately (prop_2) .

STL:

Figure 8: Prompts for converting from synthesized STL to NL via GPT-3.

C.1 Prompt example from in-order STL to NL via GPT-3

Figure 8 is a prompt example for GPT-3 to convert from STL to its corresponding NL. The input STL follows in-order expression with all the operators replaced by words with the same meanings. The prompt contains 20 NL-STL pairs, which are randomly picked up from 100 examples and are changed constantly during data creation.

```
Try to transform the following natural languages into signal temporal logics, the operators in
the signal temporal logic are: negation, imply, and, equal, until, globally, finally, or .
The signal temporal logics are pre-order expressions. The examples are as following:

natural language: It is required that for every moment during the interval 489 to 663 either the
event that ( prop_1 ) is detected and in response ( prop_3 ) should happen , or ( prop_2 ) should
be true .
STL: ['or', 'globally [489,663]', 'imply', 'prop_1', 'prop_3', 'prop_2']

natural language: ( prop_1 ) or ( prop_2 ) happens and continues until at some point during the
142 to 365 time units ( prop_4 ) happens and ( prop_3 ) happens at the same time .
STL: ['until [142,365]', 'or', 'prop_1', 'prop_2', 'and', 'prop_4', 'prop_3']

natural language: ( prop_1 ) should not happen and ( prop_2 ) should happen at the same time ,
and the above scenario is equivalent to the case that at some point during the 230 to 280 time
units ( prop_3 ) happens .
STL: ['equal', 'and', 'negation', 'prop_1', 'prop_2', 'finally [230,280]', 'prop_3']
                                                    .
                                                    .
                                                    .
natural language: In the next 0 to 5 time units , do the ( prop_1 ) , but in the next 3 to 4 time
units , ( prop_2 ) should not happen .
STL: ['and', 'globally [0,5]', 'prop_1', 'globally [3,4]', 'negation', 'prop_2']

natural language: While (prop_1) , do (prop_2) , and when (prop_3) , stop (prop_2) .
STL: ['and', 'imply', 'prop_1', 'prop_2', 'imply', 'prop_3', 'negation', 'prop_2']

natural language: If (prop_1) happens, then some time after the next 300 time steps (prop_2)
should happen.
STL: ['imply', 'prop_1', 'finally [300, infinite]', 'prop_2']

STL:
```

Figure 9: Prompts for converting from NL to pre-order STL via GPT-3.

C.2 Prompt example from NL to pre-order STL via GPT-3

Figure 9 is a prompt example for GPT-3 to convert from NL to its corresponding STL. The output STL follows pre-order expression. We have tested that GPT-3 acts with close performance when STL follows either pre-order or in-order formats.

Detect the actions or tasks in the sentence. The examples are as follows:
sentence: walk until whenever travel to flag
actions: walk, travel to flag

sentence: forever touch flag and drop orange
actions: touch flag, drop orange

sentence: drop by or whenever go to flag
actions: drop by, go to flag

sentence: at some time procure pear or stop by flag
actions: procure pear, stop by flag

.
.
.

sentence: never drop pear means that at any time go to flag
actions: drop pear, go to flag

sentence: never drop apple or secure apple
actions: drop apple, secure apple

sentence: do not let go pear or whenever start going to house
actions: let go pear, start going to house

sentence:

Figure 10: Prompts for AP recognition via GPT-3.

C.3 Prompt example for AP recognition via GPT-3

Figure 10 is a prompt example for applying GPT-3 to detect APs in natural sentences. In this example, the specific domain is Navigation.

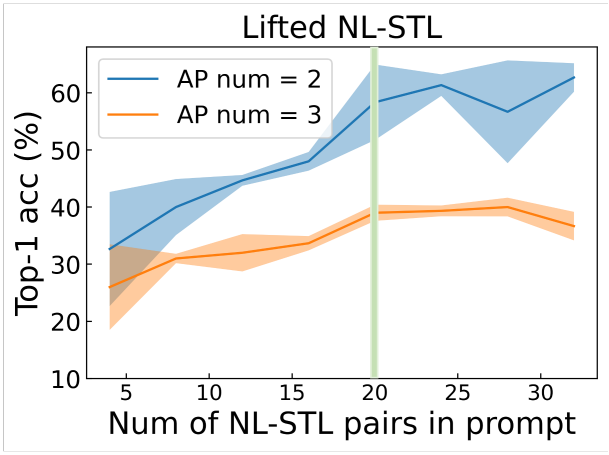

Figure 11: Number of prompt pairs VS. GPT-3 performance.

D Number of NL-STL pairs in GPT-3 prompts

As shown in Figure 11, here we ask GPT-3 to transform from NL to STL and tune the number of NL-STL pairs in the prompt to detect the accuracy evolution. We test on the NL whose targeted STL have the number of APs to be two or three. We find that the prediction accuracy given by GPT-3 will arise with the number of example pairs and turn into a plateau when the number of example pairs increases to larger than 20. Here we choose the number of pairs to be 20 in the prompt.

STL (pre-order+operator)	['<->', '->', 'prop_2', 'prop_3', 'F[55,273]', 'prop_1']
STL (in-order+word)	((prop_2 imply prop_3) equal finally[55,273] prop_1)
Raw natural sentence	If (prop_2) implies (prop_3), then (prop_1) will happen at some point during the next 55 to 273 time units .
Annotated natural sentence	If (prop_2) implies (prop_3), then (prop_1) will happen at some point during the next 55 to 273 time units, and vice versa .
STL (pre-order+operator)	['U[400,infinite]', '->', 'prop_3', 'prop_1', 'negation', 'prop_2']
STL (in-order+word)	((prop_3 imply prop_1) until[400,infinite] negation prop_2)
Raw natural sentence	If (prop_3), then do (prop_1) and keep doing it until (prop_2) happens, but this should never happen .
Annotated natural sentence	If (prop_3), then do (prop_1) and keep confirming to the above state until (prop_2) does not happens at some point after the 400 time units from now .
STL (pre-order+operator)	['<->', 'negation', 'prop_1', 'U[279,438]', 'prop_3', 'prop_2']
STL (in-order+word)	(negation prop_1 equal (prop_3 until[279,438] prop_2))
Raw natural sentence	The scenario in which (prop_1) happens is the same as the scenario in which (prop_3) happens and continues until at a certain time point during the 279 to 438 time units (prop_2) happens .
Annotated natural sentence	The scenario in which (prop_1) does not happen is the same as the scenario in which (prop_3) happens and continues until at a certain time point during the 279 to 438 time units (prop_2) happens .

Table 5: Example annotations from synthesized STLs to raw natural sentences, and further to annotated natural sentences.

E Example annotations of lifted NL-STL pairs

As shown in Table 5.

F Example full NL-STL pairs of each specialized dataset

As shown in Table 6.

Navigation	**STL**	finally (acquire_v pear_n) and globally (finally (go_to_v waste_basket_n))
	NL	when possible acquire pear and repeatedly go to waste basket .
Navigation	**STL**	finally (got_to_v house_n) and finally (go_near_v house_n)
	NL	at any time got to house and when possible go near house .
Navigation	**STL**	advance_to_v tree_n imply finally (get_to_v flag_n)
	NL	advance to tree means that when possible get to flag .
Circuit	**STL**	globally (signal_1_n math equal 89.3 or (signal_2_n more 42.4 and signal_2_n less 91.5) imply globally [0,34] (finally [0,98] (signal_3_n more equal 11.5 and signal_3_n less equal 23.4)))
	NL	In the case the signal_1_n signal is 89.3 , or the signal_2_n signal is greater than 42.4 and below 91.5 , then for every time instant during the coming 34 time units , there needs to exist a certain time instant during the next 98 time units , at which the value of signal signal_3_n should be no less than 11.5 and less than or equal to 23.4 eventually .
Circuit	**STL**	finally (signal_1_n less 92.6 and signal_2_n more equal 57.3)
	NL	At a certain time instant in the future before the end of the simulation signal_1_n is ultimately below 92.6 and signal_2_n will be ultimately at least 57.3 .
Circuit	**STL**	finally ((signal_1_n more equal 4.1 and signal_1_n less equal 59.0) or signal_2_n math equal 41.1)
	NL	There has to be a certain time instant in the future before the end of the simulation , at which the value of signal_1_n needs to be greater than or equal to 4.1 and less than or equal to 59.0 eventually , or signal_2_n finally keeps equal to 41.1 .
GLTL	**STL**	finally ((red_room or blue_room) and finally green_room)
	NL	enter the blue or orange room and proceed until the green room .
GLTL	**STL**	(finally (blue_room) and globally (negation green_room))
	NL	move to the blue room without entering a lime room .
GLTL	**STL**	(finally (yellow_room) and globally (negation blue_room))
	NL	only go through rooms that are not purple to get to the yellow room .
CW	**STL**	finally (blue_room and finally green_room)
	NL	please go to the green room through the blue room .
CW	**STL**	finally red_room
	NL	i want you to go into the red room .
CW	**STL**	finally ((red_room or yellow_room) and finally green_room)
	NL	go thru the yellow or red box to get to the green box .
Office email	**STL**	globally (((a new incident is created in Eventribe) and (a response is created in Trello)) imply (creating an object in Gmail))
	NL	When the transition action that a new incident is created in Eventribe does not get observed , and a response is created in Trello , then the following condition is true : promptly creating an object in Gmail .
Office email	**STL**	((sync Microsoft Teams data) until finally (sending me an SAP and Salesforce))
	NL	sync Microsoft Teams data until when possible sending me an SAP and Salesforce .
Office email	**STL**	globally (((a new lead is added in Marketo) and (creating a new Marketo card)) imply (a new lead is added in Microsoft Teams))
	NL	On condition that a new lead is added in Marketo and creating a new Marketo card , then the event that a new lead is added in Microsoft Teams needs to occur at the same time instant .

Table 6: Examples of full NL-STL pairs in each specialized domain.

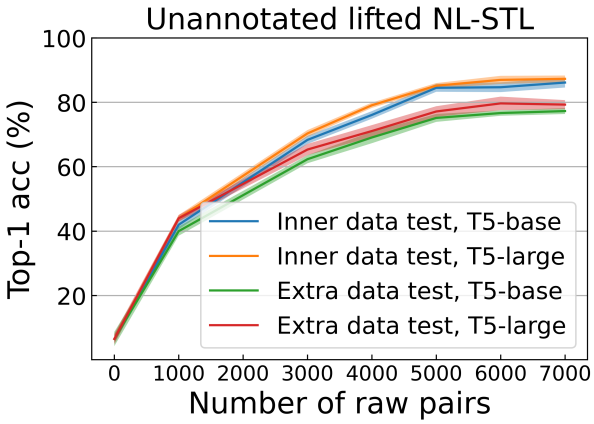

Figure 12: Testing accuracy VS. Number of raw NL-STL pairs. The data collected and re-annotated from Navigation and Circuit work are all used during training. The inner data refers to the data generated with the help of GPT-3, and the extra data refers to the instructions collected from volunteers.

G Ablation Studies

G.1 Significance of Human Annotation

This part is to demonstrate the significance of human annotation for the GPT-3 synthesized data. Figure 12 shows the model accuracy under varied number of training raw pairs. The great thing is that the T5-large model can still achieve a highest testing accuracy of 87.3% and 79.4% on the inner and extra data test, even only using the raw data synthesized from GPT-3. However, compared to the results in Figure 5, models trained on annotated data achieves accuracy about 10% higher than models trained on raw data.

G.2 Significance of Framework2

	3K dataset		**4.5K dataset**	
Domain	1.5K F1 + 1.5K F2	3K F1	3K F1 + 1.5K F2	4.5K F1
Raw data	**78.85** $\pm$ **1.04**%	75.79 $\pm$ 0.98%	**80.48** $\pm$ **0.71**%	79.04 $\pm$ 0.64%
Annotated data	**80.57** $\pm$ **0.86**%	79.76 $\pm$ 0.88%	**88.32** $\pm$ **0.84**%	86.51 $\pm$ 0.77%

Table 7: Testing accuracy of the models with different training datasets. The training data are either raw or annotated, pure from Framework1 (F1) or combining with Framework2 (F2). The experimental results show that the annotated dataset can apparently improve the performance of the model, and models combining the data generated by F1 and F2 outperform the models trained with the same amount of pure F1 data.

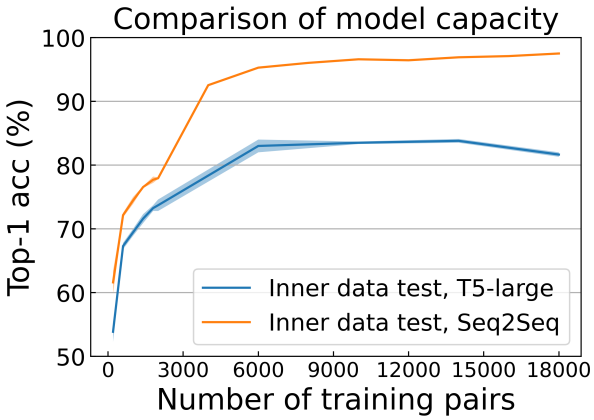

Figure 13: Testing accuracy VS. Number of training lifted NL-STL pairs. Here we use T5-large and Seq2Seq models to train on the lifted data. We detect that the Seq2Seq model reaches a highest accuracy at 83%, while T5-large model reaches a highest accuracy at 97.5%.

H Model Capacity

As shown in Figure 13, T5-large performs much better than Seq2Seq model when training on the same lifted dataset. This reveals the significance to use LLM in this NL-to-TL task.

I Datasets statistics

I.1 Statistics of lifted NL-STL dataset

# APs per STL			# Operators per STL		
avg.	median	max	avg.	median	max
2.648	3	5	3.078	3	7

Table 8: Lifted STL formula statistics: # APs for each formula, # STL operators for each formula.

		# Words per Sent.			
# Sent.	**# Vocab**	avg.	median	max	min
21867	1288	17.512	16	72	3

Table 9: Lifted sentence statistics: # unique sentences, # unique words (vocab), # words per sentence.

I.2 Statistics of full NL-STL datasets across five domains

Domain	# APs per STL			# Operators per STL		
	avg.	median	max	avg.	median	max
Circuit (He et al., 2022)	2.303	2	6	6.898	7	21
Navigation (Wang et al., 2021)	2.003	2	3	2.624	3	5
GLTL (Gopalan et al., 2018)	2.153	2	3	3.486	4	4
CW (Squire et al., 2015)	2.906	2	6	4.778	5	7
Office email (Fuggitti and Chakraborti, 2023)	2	2	2	2.582	3	4

Table 10: Full STL formula statistics in each domain: # APs for each formula, # STL operators for each formula.

Domain	# Sent.	# Vocab	# Words per Sent.			
			avg.	median	max	min
Circuit (He et al., 2022)	120000	265	38.495	37	97	1
Navigation (Wang et al., 2021)	5000	131	9.697	10	16	4
GLTL (Gopalan et al., 2018)	11153	193	11.324	11	27	4
CW (Squire et al., 2015)	3382	188	9.645	10	27	4
Office email (Fuggitti and Chakraborti, 2023)	150	143	13.407	13	25	9

Table 11: Full sentence statistics in each domain: # unique sentences, # unique words (vocab), # words per sentence.

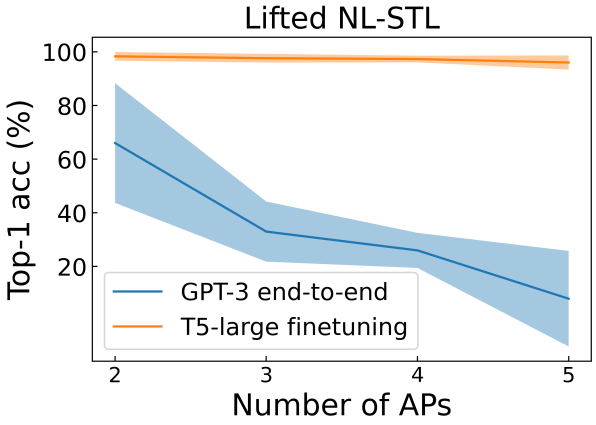

Figure 14: Accuracy VS. number of APs in each lifted STL. We carry out the test with both finetuned lifted T5-model and GPT-3 end-to-end method.

J Accuracy evolution with AP number

This section is to illustrate that directly applying GPT-3 to predict STL from NL via few-shot learning largely decreases the accuracy when the sentence structure is complex. Here we hypothesize that sentence complexity is positively related to the number of APs. As shown in Figure 14, the prediction accuracy decreases rapidly with increasing AP number using GPT-3 end-to-end method. On the other hand, the method to finetune the T5-large using synthesized NL-STL pairs remains high accuracy across different AP numbers.

K Details of implementation

For all the finetuning experiments on both T5-base and T5-large models, we choose the learning rate as 2e-5, a batch size of 16, a weight decaying ratio as 0.01, and run 20 epochs for each setting. Experiments on average finish in 3 hours for T5-base, and 10 hours for T5-large, on a single Nvidia RTX 8000 GPU. Average results and standard deviations are typically acquired from 3 runs with seeds [1203, 309, 316], apart from the transfer learning in CW dataset where 10 runs are carried with seeds [1203, 309, 316, 34, 64, 128, 256, 512, 1234, 234]. For the finetuning on lifted models, the input dataset is split into training set (0.9) and testing set (0.1).

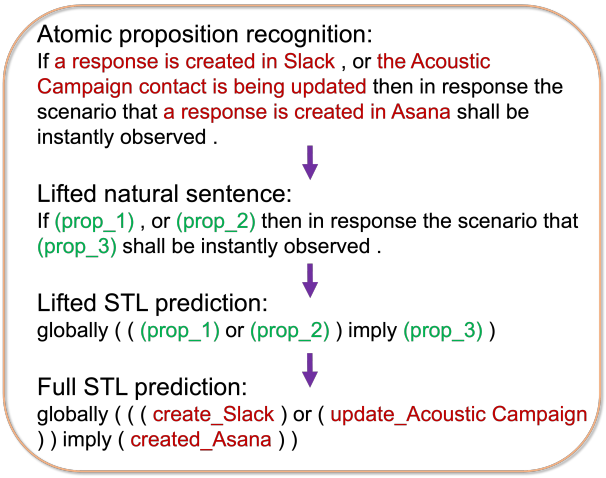

Figure 15: Illustration of full STL conversion by combing with AP recognition task using GPT-3.

L Full STL conversion by combining lifted model with AP recognition

Illustrated in Figure 15