Instructing Robots with Natural Language via Bi-RNNs for Temporal Logic Translation

Suryansh Sharma, Ki Myung Brian Lee, Mason Brown, Graeme Best
University of Technology Sydney, Australia
{suryansh.sharma, kmbrian.lee, mason.brown, graeme.best}@uts.edu.au

Abstract

We consider the problem of planning trajectories that satisfy natural language instruction. We explore translating natural language commands to temporal logic formulae to resolve ambiguities for planning. Our main contribution is a new bi-directional recurrent neural network (Bi-RNN) architecture for this translation task. We experimentally show that the proposed Bi-RNN architecture achieves 1.6% better accuracy, 20% faster inference time, and 98% faster training time compared to leading models owing to bidirectional processing. The overall system, including a planning algorithm, exhibits useful diverse behaviours that satisfy given instructions.

1 Introduction

Natural language is central to how humans interact with other intelligent beings. Humans naturally communicate their intent and instructions to others in natural language, whether spoken or written. A similar ability to instruct *robots* with natural language will significantly enhance accessibility for non-expert users, accelerating the adoption of robots. Natural language communication with robots will render interaction as seamless and intuitive as with humans.

Recent advances in natural language processing (NLP) and formal methods provide some means toward this end. NLP techniques allow machines to understand natural language. Meanwhile, formal methods provide high-level task representations such as temporal logic that facilitate automatic verification and synthesis of behaviours [Kupferman, 2018].

However, instructing robots with natural language remains challenging. Black-box NLP techniques typically suffer from contextual and spatial ambiguities. A common remedy is augmentation with more data or other modalities, such as vision, which necessitates a trade-off between accuracy and inference time. Formal task representations are often unreadable for a non-expert user.

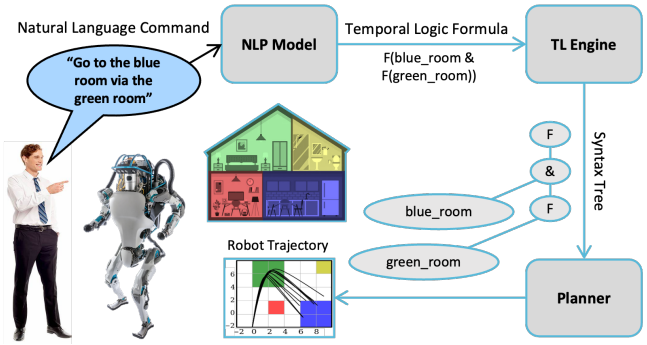


Figure 1: Use case diagram of the proposed pipeline that takes natural language commands to instruct the path of a robot. A natural language sentence is pre-processed and fed into an NLP model for inference. A NLP model outputs a temporal logic (TL) formula. The formula is passed through a TL engine, creating a syntax tree of the generated TL formula. The syntax tree is passed through a planner to generate robot trajectories.

To bridge this gap, we explore using temporal logic as an intermediate high-level task representation, as illustrated in Fig. 1. Temporal logic side-steps ambiguity challenges because it allows concrete specification of execution order and well-defined mapping of spatial concepts [Li *et al.*, 2020] to geometric locations. Automatic translation from natural language enhances accessibility for non-expert users. Previous work [Gopalan *et al.*, 2018; Chen *et al.*, 2023] shows that such a translation is possible for linear temporal logic.

Our main contribution is a new bidirectional recurrent neural network (Bi-RNN) architecture for translating natural language into signal temporal logic (STL) specifications. Bi-RNNs capture enhanced contextual information by taking previous and subsequent inputs [Schuster and Paliwal, 1997], whereas conventional RNNs rely only on the previous input. Bidirectional processing offers a better trade-off between accuracy and complexity.

Our experimental results support the benefits of the proposed Bi-RNN architecture. The proposed model exhibits significantly improved performance metrics, with 27.4% higher accuracy than [Gopalan *et al.*, 2018], and 1.61% higher than [Chen *et al.*, 2023], while the training time is significantly reduced by around 98% compared to both models. Combined with planning algorithms for temporal logic, we demonstrate that the proposed methods serve as a useful framework for an effective human-robot interface.

2 Related Work

To generate robot behaviours that comply with a given natural language command, a possible approach is to train a neural network as a 'black box' interface, as done in modern NLP. Neural network models may be trained to either directly generate robot behaviours by imitating annotated behaviours [Stepputtis *et al.*, 2020], or predict the reward function corresponding to a given natural language command [Williams *et al.*, 2018; Xie *et al.*, 2023a]. Such black-box methods are convenient and offer generalisation to unseen vocabulary. However, generalisation also implies potential hallucination [Bang *et al.*, 2023], and the opaqueness precludes opportunities for certifying or debugging the system.

Classical NLP methods, on the other hand, are fully transparent owing to using a deterministic parser for 'structured' natural language with a known, hand-coded grammar [Naseem *et al.*, 2021]. In robotics, structured-language approaches have been explored with formal task representations such as temporal logic [Finucane *et al.*, 2010] and planning domain description language (PDDL) [Xie *et al.*, 2023b]. Task representations can serve as the objective function for planning, as task completion can be algorithmically verified given robot behaviour [Kupferman, 2018]. In particular, temporal logic offers a favourable trade-off between expressiveness and planning efficiency, with many algorithmic tools for plan synthesis. However, hand-coded grammar is cumbersome for the system designer or restrictive for the user because the same natural language command may be paraphrased in many ways.

A recently emerging paradigm is to use neural network models in conjunction with formal task representations serving as an intermediate layer. [Gopalan *et al.*, 2018] presents a neural sequence-to-sequence model trained to translate commands into linear temporal logic using RNNs, combined with a Markov decision process solver to generate plans. [Chen *et al.*, 2023] builds on this model by adding dropout layers, which showed improved performance. In a similar setting, we consider using Bi-RNNs [Schuster and Paliwal, 1997], which previously showed superior performance in other NLP tasks [Wang *et al.*, 2020].

3 Background on Random Signal Temporal Logic

We consider a time-unbounded fragment of random signal temporal logic (RSTL) [Lee *et al.*, 2021b][1] as an intermediate high-level task specification between natural language commands and the planning algorithm. RSTL is defined over a set of event predicates $\mathcal{E}$, which can be, for example, "target encountered" or "reached destination". Given a set of event predicates, RSTL can specify task specifications such as "eventually go to destination" and "always avoid target". RSTL naturally supports reasoning over uncertainty, unlike deterministic STL [Donzé, 2013], and can hence be used to encode ambiguity inherent in natural language commands.

Formally, RSTL is built from a set of event predicate $E \in \mathcal{E}$ is characterised by a Bernoulli random field over the state space of a robot:

$$E \sim P(E \mid \mathbf{x}, t), \tag{1}$$

where $\mathbf{x}$ is the state of the robot, and $P(E \mid \mathbf{x}, t)$ is the probability of event E occurring at state $\mathbf{x}$ and time t. Given a set of random events $\mathcal{E}$, the syntax of an RSTL formula Φ is given in Backus-Naur form as:

$$\Phi := \top \mid E \mid \neg\Phi \mid \Phi \vee \Psi \mid \Phi \mathcal{U} \Psi, \tag{2}$$

where $\top$ is logical true, $E \in \mathcal{E}$, and Ψ, Φ are RSTL formulae. $\neg$ is logical negation, $\wedge$ is logical conjunction. $\mathcal{U}$ is the temporal operator 'Until', and $\Phi \mathcal{U} \Psi$ means Φ must hold true until Ψ. Other operators such as $\wedge$ (conjunction), $\mathcal{F}$ ('in **F**uture', i.e., eventually) and $\mathcal{G}$ ('**G**lobally', i.e., always) can be derived from the syntax as:

$$\begin{aligned} \Phi \wedge \Psi &= \neg(\neg\Phi \vee \neg\Psi), \\ \mathcal{F}\Phi &= \top \mathcal{U} \Phi, \\ \mathcal{G}\Phi &= \neg\mathcal{F}\neg\Phi. \end{aligned} \tag{3}$$

Whether a given trajectory $\mathbf{X} = \mathbf{x}_0 ... \mathbf{x}_T$ satisfies an RSTL formula Φ at time t is measured in terms of *probability of success*, $P(\mathbf{X}, t \models \Phi)$. We use the mutually exclusive (ME) approximation rule from [Lee *et al.*, 2021b]. The benefits of using the ME approximation (4) are numerical stability [Lee *et al.*, 2021b] and its alternative interpretation as a smooth approximation of deterministic STL [Gilpin *et al.*, 2021].

The ME rule recursively computes the probability of success in terms of *log-odds* $L(\mathbf{X}, t \models \Phi) =$

[1] RSTL introduced in [Lee *et al.*, 2021b] supports time domains on temporal operators $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{U}$, so that the operators are only active within the domain. We consider a subset of this language where the time domains are always unbounded.

$\log \frac{P(\mathbf{X},t\models\Phi)}{1-P(\mathbf{X},t\models\Phi)}$ as follows:

$$\begin{aligned} L(\mathbf{X}, t \models \top) &= \infty \\ L(\mathbf{X}, t \models E) &= \log \frac{P(E \mid \mathbf{x}_t, t)}{1 - P(E \mid \mathbf{x}_t, t)}, \\ L(\mathbf{X}, t \models \neg\Phi) &= -L(\mathbf{X}, t \models \Phi), \\ L(\mathbf{X}, t \models \bigvee_i \Phi_i) &= \log \sum_i \exp L(\mathbf{X}, t \models \Phi_i), \\ L(\mathbf{X}, t \models \mathcal{F}\Phi) &= \log \sum_{\tau=t}^{T} \exp L(\mathbf{X}, \tau \models \Phi). \end{aligned} \tag{4}$$

The ME approximation (4) is differentiable with respect to trajectory $\mathbf{X}$ as long as the event predicates are differentiable [Lee *et al.*, 2021b]. This enables trajectory and control synthesis via gradient ascent similar to model predictive control:

$$\mathbf{U}_{i+1} = \mathbf{U}_i + \epsilon \frac{\partial}{\partial \mathbf{X}} L(\mathbf{X}, 0 \models \Phi) \frac{\partial \mathbf{X}}{\partial \mathbf{U}}, \tag{5}$$

where $\mathbf{U}_i$ is the i-th solution for control actions. The partial derivative $\frac{\partial \mathbf{X}}{\partial \mathbf{U}}$ is obtained by differentiating the dynamic model.

4 Problem Formulation

We are interested in synthesising robot behaviours that comply with natural language commands. We consider a robot described by a discrete-time dynamic model:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t), \tag{6}$$

where $\mathbf{x}_t$ and $\mathbf{u}_t$ denote the state and control vectors for the robot, respectively, and $\mathbf{f}$ is the dynamic model.

Our objective is to synthesise a sequence of control actions $\mathbf{U} = \mathbf{u}_0 \dots \mathbf{u}_T$ of horizon T that satisfies a natural language command C^{NL} from the user. We write $\mathbf{X} \models C^{NL}$ to mean that a trajectory $\mathbf{X}$ satisfies command C^{NL}. Then, the problem of instructing robots with natural language can be formulated as finding a sequence of control actions, which can be stated as:

Problem 1 (Natural language planning)**.** *Given the dynamic model* (6) *and a cost function c, find a sequence of control actions* $\mathbf{U}$ *with minimal cost that satisfies a given natural language command* C^{NL}*:*

$$\begin{aligned} \min_{\mathbf{U}} \quad & \sum_{\tau} c(\mathbf{u}_\tau) \\ s.t. \quad & \textit{dynamics } (6), \\ & \mathbf{X} \models C^{NL}. \end{aligned} \tag{7}$$

Here, the cost function $c(\mathbf{u})$ penalises control actions so that they are not needlessly large. More importantly, a glaring challenge in Problem 1 is to algorithmically determine whether a trajectory $\mathbf{X}$ satisfies the natural language command $\mathbf{X} \models C^{NL}$.

We mitigate this challenge by translating the natural language command C^{NL} to an RSTL formula Φ as an intermediate interface. With such translation, solving Problem 1 is reduced to planning a satisfactory trajectory for the translated RSTL formula Φ, which can be readily achieved by gradient ascent (5).

To this end, we assume that a labelled dataset is available between natural language commands and temporal logic formula and learn a translation function that produces equivalent output. We consider two formulae Φ_1 and Φ_2 *logically equivalent* (i.e. $\Phi_1 \equiv \Phi_2$) if the corresponding Büchi automata are isomorphic when interpreted as linear temporal logic[2]. Unlike string equivalence in typical NLP, temporal logic formaulae may be written in different order but remain logically equivalent. For example, $\mathcal{F}(A) \wedge \mathcal{F}(B) \equiv \mathcal{F}(B) \wedge \mathcal{F}(A)$. The translation problem is then posed as maximising the logical equivalence between predicted and ground truth formualae:

Problem 2 (Temporal logic translation)**.** *Given a labelled dataset* $\mathcal{D} = \{(C_n^{NL}, \Phi_n)\}_{n=1}^N$*, find a translation function F that maximises overall logical equivalence between ground truth and prediction:*

$$\begin{aligned} \max_{F} \quad & \sum_n 1(\hat{\Phi}_n \equiv \Phi_n), \\ s.t. \quad & \hat{\Phi}_n = F(C_n^{NL}), \end{aligned} \tag{8}$$

where $1(\hat{\Phi}_n \equiv \Phi_n) = 1$ *if* $\hat{\Phi}_n \equiv \Phi_n$ *and* 0 *otherwise.*

5 Bi-RNN for Temporal Logic Translation

Bi-RNN [Schuster and Paliwal, 1997] is proposed as a parameterisation of the translation function to solve Problem 2. Solving Problem 2 allows solving Problem 1 with gradient ascent toward RSTL satisfaction as per (5).

An overview of the proposed Bi-RNN model is shown in Fig. 2. A natural language command input is presented as a string and is converted to a numeric format by the tokeniser. The numeric output from the tokeniser is processed through the Bi-RNN network and converted back to an appropriate TL formula in string format. In what follows, we detail the functions of each component.

5.1 Tokeniser

The tokeniser converts string data to a numeric format. Given a sentence in string format, the tokeniser outputs

[2]See [Kupferman, 2018] for Büchi automaton construction, and algorithmic determination of isomorphism. We use the implementation in SPOT toolbox [Duret-Lutz and Poitrenaud, 2004]

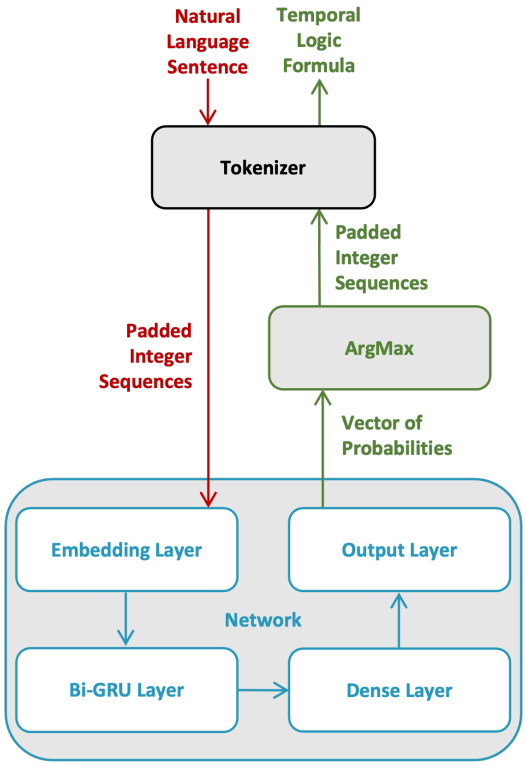


Figure 2: Proposed network architecture. The tokeniser converts natural language and temporal logic commands in string format to sequences of integers. The embedding layer converts a sequence of integers into a sequence of dense vectors. The bidirectional GRU (Bi-GRU) layer learns sequential features. The dense and output layers learn translation from sequential features to temporal logic formulae.

a sequence of integers by assigning an integer token to each unique word. The output is padded with zeros to comply with a set length known as the max length. This process requires a dictionary-like structure where the key is the index of the integer token, and the value is each word. We construct tokens over the input natural language commands and the output TL formulae to share the tokens for predicates.

All punctuation is stripped for the input natural language commands, and tokens are assigned to each unique word in the dataset separated by the whitespace character. For the output temporal logic formulae, tokens are assigned to all predicates found in the dataset, and the TL operators, $\mathcal{F}, \mathcal{G}, \mathcal{U}, \wedge, \vee$. The output sequence length is set as the maximum length of the strings in the dataset.

5.2 Network Architecture

The proposed network F takes as input the padded sequence of integers x from the tokeniser and outputs a dense probability vector $\hat{y} = F(x;\theta)$ where θ are parameters. The network comprises three main types of layers, which are 1) embedding, 2) bidirectional gated recurrent unit (GRU), and 3) dense layers, as visualised in the "network" block of Fig. 2.

	Output Dim.	Activation
Tokeniser	$(\cdot, 34)$	NA
Embedding	$(\cdot, 34, 100)$	None
BiGRU	$(\cdot, 34, 128)$	sigmoid & tanh
Dense	$(\cdot, 34, 64)$	ReLU
Output	$(\cdot, 34, 152)$	Softmax

Table 1: Network Architecture Table: The table represents the output dimensions of each layer in the network along with the activation function used.

The embedding layer converts the padded sequence of integers from the tokeniser into a sequence of dense vectors. Letting the input sequence of integers be x, the i-th output of the embedding layer is simply the x_i-th column of the weight matrix:

$$w_i = W_{x_i}^{\text{embedding}} \tag{9}$$

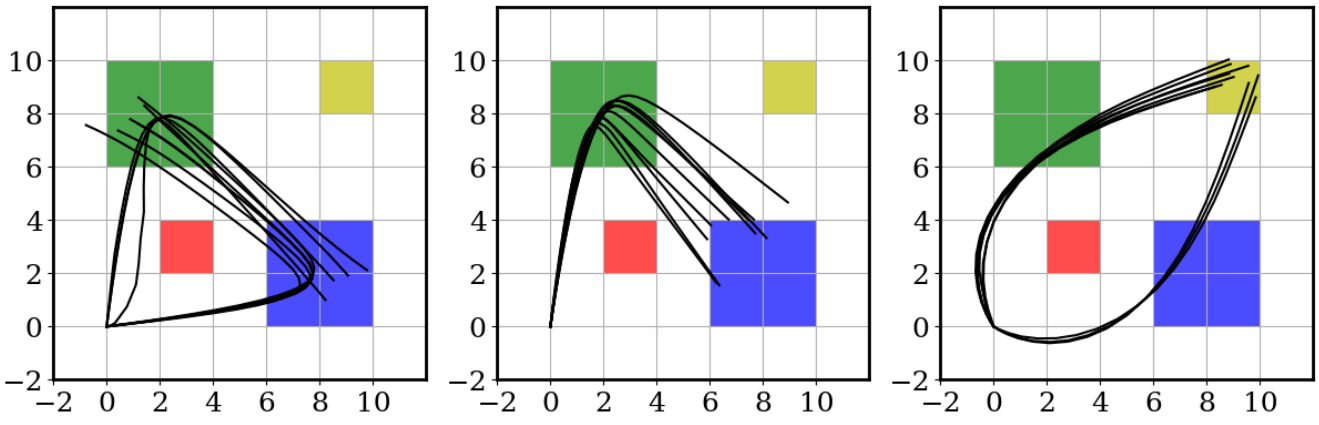
The embedded word vectors are fed into a bidirectional GRU to learn sequential patterns in the natural language command. Bi-directional processing improves understanding of contextual information and is beneficial in translation tasks [Cho *et al.*, 2014]. The bidirectional GRU layer trains two independent GRU cells in backward and forward directions:

$$\begin{aligned} h_i^{\text{BiGRU}} &= [h_i^{\text{forward}}, h_i^{\text{backward}}], \\ h_i^{\text{forward}} &= \text{GRU}(w_i, h_{i-1}^{\text{forward}}), \\ h_i^{\text{backward}} &= \text{GRU}(w_i, h_{i+1}^{\text{backward}}). \end{aligned} \tag{10}$$

Each GRU cell computes an output for index i based on the previous (for forward GRU) or the next (for backward GRU) output. In doing so, the previous (or next) output is *gated* by update and reset vectors z_i and r_i, which are given by:

$$\begin{aligned} z_i &= \text{sigmoid}(W^{zx} w_i + W^{zy} h_\star + b^z), \\ r_i &= \text{sigmoid}(W^{rx} w_i + W^{ry} h_\star + b^r), \\ \hat{h}_i &= \tanh\left(W^{yx} w_i + W^{yy}(r_i \otimes h_\star) + b^y\right), \\ h_i &= z_i \otimes \hat{h}_i + (1 - z_i) \otimes h_\star, \end{aligned} \tag{11}$$

where $\otimes$ denotes the Hadamard element-wise product, $h_\star$ is the previous output for the forward GRU, and the next for backward. Here, the reset vector r_i controls the influence of $h_\star$ on the proposed output $\hat{h}_i$. The update vector z_i linearly mixes between the previous/next output $h_\star$ and $\hat{h}_i$ The update and reset vectors z_i and r_i are



(a) (b) (c)

Figure 3: Example behaviours from the system. (a) "*Go to the green room and the blue room*"; (b) "*Go to the blue room via the green room*"; (c) "*Avoid landmark 1 (red) and go to the yellow room*".

computed as independent neural networks with sigmoid activation, which has output range $[0., 1.]$.

The sequential features learnt using BiGRU are fed into two fully connected layers, one with rectified linear unit (ReLU) [Fukushima, 1969] and the other with softmax activation:

$$\begin{aligned} h_i^{\text{dense}} &= \text{ReLU}(W^{\text{dense}} h_i^{\text{BiGRU}} + b^{\text{dense}}), \\ \hat{y}_i &= \text{softmax}(W^{\text{output}} h_i^{\text{dense}} + b^{\text{output}}). \end{aligned} \tag{12}$$

These two fully connected layers translate the sequential features from BiGRU to the temporal logic formula. ReLU allows learning a complex FC layer while avoiding the issue of vanishing gradients [Fukushima, 1969] common in neural network training. The softmax layer generates a probabilistic output representing the probability of each token appearing in the translated formula. Thus, an arg max operation over the tokens yields a sequence of integers that can be de-tokenised to a string format.

5.3 Training and Evaluation

To train the proposed model, we first construct a tokeniser from the dataset as per Sec. 5.1 by identifying unique word tokens in the dataset. Using the tokeniser, natural language commands and temporal logic formulae are preprocessed into padded sequences of integer indices. This promotes the use of the sparse categorical cross-entropy loss, which can be computed as:

$$L(\theta) = \sum_i \log \hat{y}_i[x_i], \tag{13}$$

where $\hat{y}_i[x_i]$ denotes the x_i-th element in the probability vector $\hat{y}_i$. The cross-entropy loss is a useful proxy to the logical equivalence metric in Problem 2. It is useful because the cross-entropy loss is differentiable, whereas the logical equivalence metric is not due to the intermediate construction of a Büchi automaton. The two objectives were found to be strongly correlated in our experiments.

6 Experimental Results

We experimentally evaluate the proposed Bi-RNN architecture. We first detail the implementation of the proposed model and demonstrate the overall system from commands to plan generation in a mock indoor environment. We then compare the performance of the proposed model to previous work [Gopalan *et al.*, 2018; Chen *et al.*, 2023], in terms of accuracy and computation time.

6.1 Implementation

The proposed Bi-RNN was implemented using the Tensorflow Keras API [Chollet and others, 2015]. The gradient-ascent planner was implemented using Tensorflow based on [Lee *et al.*, 2021b]. The proposed model was trained on the `Cleanup World` dataset, which contains natural language commands and the corresponding TL formulae [Gopalan *et al.*, 2018]. The dataset was shuffled and split into training (for backpropagation), validation (for assessing performance during training) and testing sets (for assessing performance post-training).

There were 34 unique tokens, and the maximum sequence length was 152. Table 1 summarises the corresponding sizes of the layers. Adding further hidden layers to the proposed architecture was found to have a detrimental impact on performance. We train the model using the Adam optimiser [Kingma and Ba, 2015] with

	Ground Truth	[Gopalan *et al.*, 2018]	[Chen *et al.*, 2023]	Proposed
Go to landmark one without leaving the purple room.	$F(\textbf{landmark}_1)$ $\&G(\textbf{purple\_room})$	$F(\textbf{landmark}_1)$ $\&G(\textbf{purple\_room})$	$F(\textbf{landmark}_1)$ $\&G(\textbf{purple\_room})$	$F(\textbf{landmark}_1)$ $\&G(\textbf{purple\_room})$
Go to the second floor, then go to the purple room.	$F(\textbf{second\_floor}$ $\&F(\textbf{purple\_room}))$	$F($blue_room $\&F($purple_room$))$	$F($third_floor $\&F($purple_room$))$	$F(\textbf{second\_floor}$ $\&F(\textbf{purple\_room}))$
Go to landmark two but always avoid landmark one.	$F(\textbf{landmark}_2$ $\&\neg\textbf{landmark}_1)$	$F($landmark$_2$ $\&\neg$landmark$_2)$	$F(\textbf{landmark}_2$ $\&\neg\textbf{landmark}_1)$	$F(\textbf{landmark}_2$ $\&\neg\textbf{landmark}_1)$
Go directly to the purple room without changing floors.	$F(\textbf{purple\_room})$	$F($purple_room$)$ $\&G($second_floor$)$	$F($purple_room$)$ $\&G($first_floor$)$	$F($third_floor$)$
Move to the third floor without entering the yellow room.	$F(\textbf{third\_floor}$ $\&\neg\textbf{yellow\_room})$	$F($yellow_room $\&\neg$yellow_room$)$	$F($first_floor $\&\neg$yellow_room$)$	$F(\textbf{third\_floor}$ $\&\neg\textbf{yellow\_room})$

Table 2: Examples of predictions made by [Gopalan *et al.*, 2018], [Chen *et al.*, 2023] and the proposed model for a given natural language sentence with their respective ground truth value. Results were obtained from the best-saved checkpoints from a checkpoint series stored while training over 400 epochs. The correct values are shown in green.

a 0.001 learning rate. All models were trained with the same hardware: AMD Ryzen 7 5800 8-Core Processor CPU, NVIDIA GeForce RTX 3070 GPU and 32 GB RAM.

6.2 Demonstration of the System

Example behaviours from the overall system are illustrated in Fig. 3. Three natural language commands were presented, which are a) "*Go to the green room and the blue room*", b) "*Go to the blue room via the green room*", and c) "*Avoid landmark 1 (red) and go to the yellow room*". These natural language commands were translated to TL using the proposed Bi-RNN model, from which the RSTL gradient ascent planner generated plans with ten random initial conditions.

The Bi-RNN model produced correct outputs as a) $\mathcal{F}(\texttt{green\_room}) \wedge \mathcal{F}(\texttt{blue\_room})$; b) $\mathcal{F}(\texttt{green\_room} \wedge \mathcal{F}(\texttt{blue\_room}))$; and c) $\mathcal{G}(\neg\texttt{landmark\_1}) \wedge \mathcal{F}(\texttt{yellow\_room})$ respectively. Consequently, it can be seen that the planner generates appropriate behaviours in Fig. 3. Example outputs from the Bi-RNN model are shown in Table 2.

In particular, the behaviours in Figs. 3(a) and (c) is interesting because two distinct modes of behaviours are apparent. This is because the natural language command does not specify the order in (a), and a particular side for (c). Meanwhile, the behaviour in Fig. 3(b) has a single mode of behaviour because the order was specified in the natural language command. This demonstrates the suitability of TL as a task representation, as it offers avenues for greater flexibility.

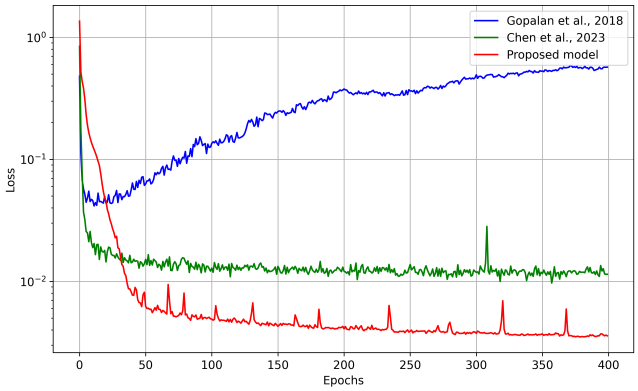


Figure 4: Loss curves for the three models. The proposed model achieves the best optimum. The baseline models either diverge or achieve a worse optimum.

6.3 Performance Comparison

We compare the performance of the proposed model against [Gopalan *et al.*, 2018] and [Chen *et al.*, 2023]. [Gopalan *et al.*, 2018] uses an EncoderRNN and an attention DecoderRNN. [Chen *et al.*, 2023] additionally uses attention with dropout layers. The proposed model does not use an attention mechanism.

[Gopalan *et al.*, 2018] and [Chen *et al.*, 2023] are trained using negative log likelihood as originally proposed. The learning rate for the [Gopalan *et al.*, 2018] model was 0.01, and 0.001 for the [Chen *et al.*, 2023] model. All models were trained for 400 epochs while saving checkpoints at every epoch. The training loss and logical equivalence accuracy over the epochs are shown in Figs. 4 and 5. It can be seen that the training loss

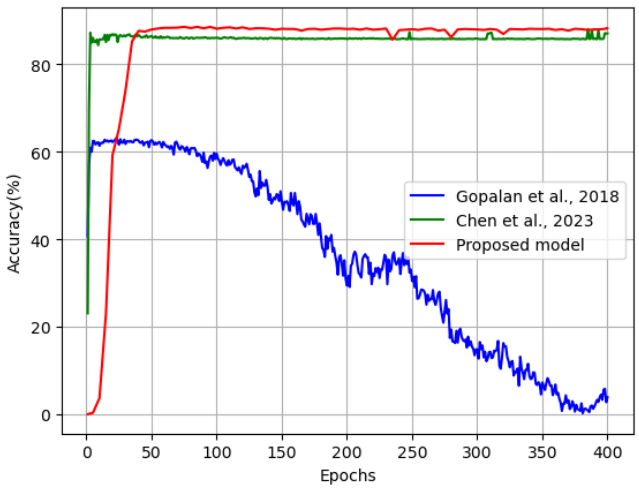


Figure 5: Logical equivalence of the three models over training epochs. Whereas the proposed model achieves the optimum later than the baseline models, the final accuracy is higher and more stable.

is stable for the proposed model and [Chen *et al.*, 2023], whereas [Gopalan *et al.*, 2018] shows an increase after 14 epochs. A similar pattern is observed for logical equivalence accuracy in Fig. 5. This shows that the proposed Bi-RNN architecture is more suitable for representing the commands.

Table 3 shows the best-performing model accuracy. The best-performing models were from the last epoch for the proposed model and [Chen *et al.*, 2023], and the 14th epoch for [Gopalan *et al.*, 2018]. The proposed model performs the best at 89.12% on the testing set, on par with [Chen *et al.*, 2023] at 88.68% on the validation set and the best at 93.34 % on the training set. Notably, the proposed model shows significantly better accuracy than [Gopalan *et al.*, 2018], which shows 64.37%, 63.15%, and 61.31% accuracy on training, validation, and testing sets, respectively. These results show that bidirectional processing is more effective than an attention mechanism in temporal logic translation tasks.

The computation time characteristics are shown in Table 4. The proposed model is the fastest in both training and single-query inference. The proposed model shows a faster training time of around 2 seconds, whereas the previous models take around 120 seconds per epoch. The faster training time allows potential extensions, such as updating the model with user feedback. For all models, the inference time is around 0.2 milliseconds, although the proposed model is still the fastest. GPU acceleration has minimal effect on computation time for a single query. This signifies the possibility of deployment to embedded CPU-only hardware with compute constraints.

	Training	Validation	Testing
[2018]	65.11 (64.37)	64.22 (63.15)	62.5 (61.31)
[2023]	90.12 (90.13)	**88.68 (88.68)**	87.5 (87.5)
Proposed	**93.34 (93.34)**	**88.68 (88.68)**	**89.12 (89.12)**

Table 3: Comparison of accuracy (%) between the proposed and prior models. The best is highlighted green. String equivalence accuracy is parenthesised.

	Training	GPU	CPU
[Gopalan *et al.*, 2018]	1.35×10^2	2.45×10^{-4}	2.75×10^{-4}
[Chen *et al.*, 2023]	1.27×10^2	2.49×10^{-4}	2.89×10^{-4}
Proposed Model	**2**	$\mathbf{2.0 \times 10^{-4}}$	$\mathbf{2.2 \times 10^{-4}}$

Table 4: Computation time (in sec) for training and CPU/GPU inference with best highlighted green. Training time is per epoch. Inference time is for a single query.

7 Conclusion

We proposed a Bi-RNN architecture for temporal logic translation to allow planning for natural language instruction. The proposed model provides higher accuracy with reduced training and inference time. Reduced training time enables online updates to the model in the future, which is compelling for robotics applications. We also plan to implement logical equivalence during training. More broadly, we are exploring how natural language instructions can benefit human-robot teaming in a decentralised ISR context [Lee *et al.*, 2021a; Best *et al.*, 2019].

Acknowledgements

This work was supported in part by the Trusted Autonomous Systems D-CRC.

References

[Bang *et al.*, 2023] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[Best *et al.*, 2019] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Dec-MCTS: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[Chen *et al.*, 2023] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. NL2TL: Transforming natural languages to temporal logics using large language

models. In *Proc. of Association for Computational Linguistics Rolling Review*, 2023.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of Conference on Empirical Methods in Natural Language Processing*, 2014.

[Chollet and others, 2015] François Chollet et al. Keras. GitHub, 2015. https://keras.io.

[Donzé, 2013] Alexandre Donzé. On signal temporal logic. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, pages 382–383, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Duret-Lutz and Poitrenaud, 2004] Alexandre Duret-Lutz and Denis Poitrenaud. SPOT: An extensible model checking library using transition-based generalized Büchi automata. In *Proc. of IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 76–83, 2004.

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

[Fukushima, 1969] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[Gilpin *et al.*, 2021] Yann Gilpin, Vince Kurtz, and Hai Lin. A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control System Letters*, 5(1):241–246, 01 2021.

[Gopalan *et al.*, 2018] Nakul Gopalan, Dilip Arumugam, Lawson L. S. Wong, and Stefanie Tellex. Sequence-to-sequence language grounding of non-Markovian task specifications. In *Proc. of Robotics: Science and Systems*, 2018.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of International Conference on Learning Representations*, 2015.

[Kupferman, 2018] Orna Kupferman. Automata theory and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*. Springer International Publishing, 2018.

[Lee *et al.*, 2021a] Ki Myung Brian Lee, Felix H Kong, Ricardo Cannizzaro, Jennifer L Palmer, David Johnson, Chanyeol Yoo, and Robert Fitch. Decentralised intelligence, surveillance, and reconnaissance in unknown environments with heterogeneous multi-robot systems. In *Proc. of ICRA2021 Workshop on Robot Swarms in the Real World: From Design to Deployment*, 2021.

[Lee *et al.*, 2021b] Ki Myung Brian Lee, Chanyeol Yoo, and Robert Fitch. Signal temporal logic synthesis as probabilistic inference. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 5483–5489, 2021.

[Li *et al.*, 2020] Tengfei Li, Jing Liu, Haiying Sun, Xiang Chen, Lipeng Zhang, and Junfeng Sun. A spatio-temporal specification language and its completeness & decidability. *Journal of Cloud Computing*, 9(1):65, Nov 2020.

[Naseem *et al.*, 2021] Usman Naseem, Imran Razzak, Shah Khalid Khan, and Mukesh Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *ACM Transactions on Asian Low-Resource Language Information Processing*, 20(5), Jun 2021.

[Schuster and Paliwal, 1997] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45, 1997.

[Stepputtis *et al.*, 2020] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

[Wang *et al.*, 2020] Bingyuan Wang, Fang Miao, Xueting Wang, and Libiao Jin. Text classification using a bidirectional recurrent neural network with an attention mechanism. In *Proc. of International Carnahan Conference on Security Technology*, 2020.

[Williams *et al.*, 2018] Edward Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *Proc. of IEEE International Conference on Robotics and Automation*, 2018.

[Xie *et al.*, 2023a] Amber Xie, Youngwoon Lee, Pieter Abbeel, and Stephen James. Language-conditioned path planning. In *Proc. of Conference on Robot Learning*, 2023.

[Xie *et al.*, 2023b] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.