

# Embedding Symbolic Temporal Knowledge into Deep Sequential Models

Yaqi Xie\*, Fan Zhou\*, and Harold Soh

Dept. of Computer Science, National University of Singapore.

{yaqixie, zhoufan, harold}@comp.nus.edu.sg

**Abstract**— Sequences and time-series often arise in robot tasks, e.g., in activity recognition and imitation learning. In recent years, deep neural networks (DNNs) have emerged as an effective data-driven methodology for processing sequences given sufficient training data and compute resources. However, when data is limited, simpler models such as logic/rule-based methods work surprisingly well, especially when relevant prior knowledge is applied in their construction. However, unlike DNNs, these “structured” models can be difficult to extend, and do not work well with raw unstructured data. In this work, we seek to learn flexible DNNs, yet leverage prior temporal knowledge when available. Our approach is to embed symbolic knowledge expressed as linear temporal logic (LTL) and use these embeddings to guide the training of deep models. Specifically, we construct semantic-based embeddings of automata generated from LTL formula via a Graph Neural Network. Experiments show that these learnt embeddings can lead to improvements on downstream robot tasks such as sequential action recognition and imitation learning.

## I. INTRODUCTION

Sequence learning is crucial building-block for AI and robotics; it is applied to various problems including action prediction and policy learning. Significant advances have been made in sequence learning, exemplified by improvements on tasks ranging from visual tracking [1] and language translation [2] to human modeling [3] and game playing [4]. This progress has been largely powered by deep learning [5]. However, deep models often require large amounts of training data, which may limit their application in situations where data is not as readily available. For example, in imitation learning contexts, expert demonstrations are typically expensive and difficult to procure.

In many settings, high-level structured knowledge is often available in addition to data. Consider that humans teach children not only by giving examples, but also through structured information. For example, a parent who is assembling a Lego toy with child may point out that “two blocks can be put together to form a bigger block,” or “according to the picture, the red brick should be on the left”. Similarly, cooking recipes include general tips such as the following when making apple pie: “to cook apples thoroughly, first bring the water to a simmer, and after lowering in the apples, cover the pot with a baking parchment”. However, it remains unclear how we can best leverage these types of structured knowledge in deep neural networks (DNNs).

In this work, we seek to incorporate *temporal knowledge* into deep sequential learning. We focus on temporal

knowledge expressed in finite Linear Temporal Logic ( $LTL_f$ ).  $LTL_f$  is well-defined and unambiguous compared to natural language, yet relatively easy for humans to derive and interpret. These properties make  $LTL_f$  a useful language for specifying temporal relationships and dynamic constraints, and for automated reasoning. As such, various attempts have been proposed to incorporate  $LTL_f$  in sequential models, e.g., in reinforcement learning via planning [6]–[8] and by using logic checkers as auxiliary losses [9]–[11].

Different from prior work, we explore the incorporation of  $LTL_f$  knowledge into neural networks via *real-vector embeddings*. In contrast to symbols and their operators, embeddings are naturally processed by standard deep neural networks. When properly structured/learned, distances (e.g., Euclidean) in the embedding space can be exploited to quickly compute whether a given output from a deep network is consistent with related  $LTL_f$  formulae. Unlike standard model checkers, our approach enables a form of “soft”-satisfiability, which can potentially generalize knowledge. In our cooking example, when making a fruit pie, the knowledge for using apples (e.g., cutting, poaching) may apply to other fruits (e.g., pears). However, these advantages are contingent upon a successful transfer of the information contained within  $LTL_f$  formulae into corresponding real-vectors. A principal challenge is that  $LTL_f$  syntax trees are multifarious; the same formulae can often be written in many different ways, which makes it hard for neural networks to learn relevant semantics.

Our insight is that a *deterministic finite automaton* (DFA) (translated from an  $LTL_f$  specification) incorporates the needed semantics, yet has a structure that is amenable to learning via graph neural networks (GNN) [12], [13]. In particular, we note that the message propagation between nodes in graph convolutional layers can approximate the path traversals used in  $LTL_f$  model checking algorithms. Indeed, our model-checking experiments in Sec. V show that embedding DFAs resulted in far better performance compared to syntax trees.

Based on this idea, we take first steps towards a practical approach for training deep sequential models using a combination of data and prior temporal knowledge. We propose the Temporal-Logic Embedded Automata Framework (T-LEAF) that “steers” deep models during training to be consistent with specified  $LTL_f$  formulae (Fig. 1). Specifically, T-LEAF uses a custom-designed hierarchical embedder to convert DFAs into embeddings that are then used in a *logic loss*. The logic loss encourages the output of models to be consistent

\*Equal Contribution.

# T-LEAF

Temporal-Logic Embedded Automata Framework

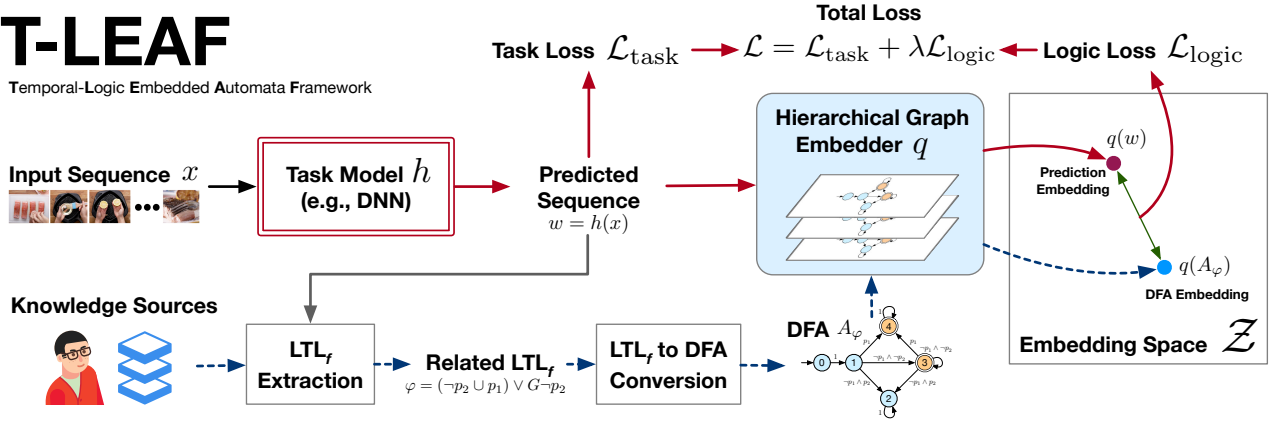


Fig. 1: An overview of the Temporal-Logic Embedded Automata Framework (T-LEAF). T-LEAF trains a deep task model  $h$  (red box) to be consistent with prior knowledge encoded as  $LTL_f$  formulae. The extracted  $LTL_f$  formulae are first converted to a DFA  $A_\varphi$ . Both the DFA  $A_\varphi$  and the predicted sequence  $w$  (from the task model  $h$ ) are projected to their respective embeddings using a hierarchical graph embedder  $q$ . The two embeddings,  $z_\varphi = q(A_\varphi)$  and  $z_w = q(w)$ , are compared to give a logic loss  $\mathcal{L}_{\text{logic}}$ , which is back-propagated to the task model together with the task loss  $\mathcal{L}_{\text{task}}$  (by following the solid red arrows in reverse). In effect,  $\mathcal{L}_{\text{logic}}$  provides an additional training signal that encourages the model  $h$  to produce outputs compatible with existing knowledge.

with the embedding, and hence the  $LTL_f$  specification. Our experiments on two tasks in the robot cooking domain give positive evidence that T-LEAF is able to improve deep models. Specifically, we tested T-LEAF on sequential human activity recognition and imitation learning. In both tasks, training with T-LEAF led to better performance for baseline and state-of-the-art deep networks.

In summary, the main contribution of this paper is a new framework for utilizing symbolic temporal logic in sequential deep models. Experiments show T-LEAF is able to enhance deep models on two challenging tasks. To our knowledge, this is the first work to study graph-based embeddings of  $LTL_f$  formulae, and to demonstrate their usefulness for training better models. We believe further improvements are possible by fine-tuning T-LEAF. More broadly, our results indicate that temporal logic embeddings are promising and warrant further research.

## II. PRELIMINARIES: LINEAR TEMPORAL LOGIC

Linear Temporal Logic (LTL) is a propositional modal logic often used to express temporally extended constraints over state trajectories [14]. In this work, we use LTL interpreted over finite traces, which is also called *finite LTL* ( $LTL_f$ ).  $LTL_f$  has a natural and intuitive syntax. As a formal language, it has well-defined semantics and thus is unambiguously interpretable, which is an advantage over using natural language directly as auxiliary information [15]<sup>1</sup>. In this section, we briefly review relevant background on  $LTL_f$ ; we focus on material relevant to our method and refer readers to excellent review articles [17]–[19] for more information.

**Finite Linear Temporal Logic.** The syntax of  $LTL_f$  for a finite set of propositions  $p \in AP$  includes the standard logic

connectives ( $\wedge, \vee, \neg$ ), true and false symbols, and temporal operators *next* (X) and *until* (U),

$$\varphi := \text{false} | \text{true} | p | \neg p | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | \neg \varphi | X\varphi | \varphi_1 U \varphi_2$$

Finite  $LTL_f$  formulae are interpreted over finite traces  $w = \sigma_0 \sigma_1 \dots \sigma_n$  of propositional states, where each  $\sigma_i$  is a set of propositions from  $P$  that are true in  $\sigma_i$ . We say that  $w$  satisfies a formula  $\varphi$ , denoted  $w \models \varphi$ , when  $w, 0 \models \varphi$ , where:

$$w, i \models p \text{ iff } p \in AP \text{ and } \sigma_i \models p \quad (1)$$

$$w, i \models \neg \alpha \text{ iff } w, i \not\models \alpha \quad (2)$$

$$w, i \models \alpha \wedge \beta \text{ iff } w, i \models \alpha \text{ and } w, i \models \beta \quad (3)$$

$$w, i \models X\alpha \text{ iff } w, i+1 \models \alpha \quad (4)$$

$$w, i \models \alpha U \beta \text{ iff } w, k \models \beta \text{ for some } i \leq k < n \quad (5)$$

$$\text{and } w, j \models \alpha \text{ for all } i \leq j < k \quad (6)$$

$LTL_f$  has been used in AI and robot planning to specify temporally-extended goals [20]–[22] and preferences [23], [24]. Compared to low-level robot programming, specifying (and interpreting) high-level task requirements using  $LTL_f$  is relatively easy for humans. To define a task,  $LTL_f$  needs a high-level domain specific vocabulary comprising a set of propositions that relates to properties of the environment, or the occurrence of events that can be determined to be true or false.

**LTL and Automata.** For every  $LTL_f$  formula  $\varphi$ , we can construct a deterministic finite-state automaton (DFA),  $A_\varphi$ , which accepts the models of  $\varphi$  [25], [26], i.e. the interpretations that satisfy  $\varphi$ . A DFA is a tuple  $A_\varphi = \langle O, \Sigma, o_0, \delta, \alpha \rangle$ , where  $O$  is a finite set of states,  $\Sigma$  contains all subsets of propositions  $\varphi$ ,  $o_0 \in O$  is the initial state,  $\delta \subseteq O \times L(P) \times O$  is a transition relation, where  $L(P)$  is the set of propositional formulae over  $P$ , and  $\alpha \subseteq O$  is a set of accepting states. A run of  $A_\varphi$  on a word  $w = x_1 \dots x_n \in \Sigma^*$  is a sequence of

<sup>1</sup>It is also possible to translate natural language into  $LTL_f$  [16], which can then be analyzed using developed tools (e.g., model checkers) and embedded using T-LEAF.

states  $o_0 o_1 \dots o_n$  such that  $(o_{i-1}, \varphi_i, o_i) \in \delta$  and  $x_i \models \varphi_i$  for each  $i \in 1, \dots, n$ . A run is accepting if  $o_n \in \alpha$ .

A DFA  $A_\varphi$  can be represented as a directed graph  $\mathcal{G}_\varphi = (\mathcal{V}, \mathcal{E})$  with nodes  $v_i \in \mathcal{V}$  representing the automata states, and directed edges  $e_j = (v_s, v_t, F_{s,t}) \in \mathcal{E}$  from a source node  $v_s \in \mathcal{V}$  to a target node  $v_t \in \mathcal{V}$ . Each edge  $e_j$  contains a propositional logic formula  $F_{s,t}$  (also denoted  $F_j$ ) that defines the conditions under which transit is permitted from  $v_s$  to  $v_t$ . There are three types of node states: an initial state, intermediate states, and acceptance states. A given trace is satisfying trace if it begins at the initial state and terminates at one of the acceptance states. For the remainder of this paper, we will assume DFAs are represented as directed graphs.

### III. METHOD: EMBEDDING DFAS FOR DEEP MODEL TRAINING VIA T-LEAF

This section details our primary contribution, i.e., a framework for using symbolic temporal knowledge in the training of deep models. An overview of our proposed Temporal-Logic Embedded Automata Framework (T-LEAF) is shown in Fig. 1. Briefly, the key component in our framework is a hierarchical graph embedder  $q$  that embeds DFAs and predicted sequences from deep models (traces) into a shared real-vector space  $\mathcal{Z} \subseteq \mathbb{R}^d$ . We train  $q$  such that embedded formulae are closer to their satisfying traces. We can then easily evaluate a *logic loss* — that captures how much a given trace satisfies related formulae (and hence the symbolic knowledge) — by computing distances in  $\mathcal{Z}$ . In the following, we first detail the DFA embedder, followed by the logic loss.

**Hierarchical DFA Embedder.** Recall that a DFA can be represented as a directed graph  $\mathcal{G}_\varphi = (\mathcal{V}, \mathcal{E})$  with three node types and propositions along the edges. To embed the information contained in the DFA, we need to embed the overall graph structure, together with the propositions.

We propose a *hierarchical embedder*  $q$  that comprises an edge-embedder  $q_e$  and a meta-embedder  $q_m$ . Embedding a given DFA involves the construction of an intermediate graph  $\hat{\mathcal{G}}_\varphi$  that will be embedded using  $q_m$ . At a high-level, our goal is to convert each edge formula into a corresponding *node feature vector*<sup>2</sup>. The key steps are:

- 1) Embed each proposition  $F_j$  along edge  $e_j$  into a corresponding vector  $s_j = q_e(F_j)$ .
- 2) Construct  $\hat{\mathcal{G}}_\varphi = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ , where  $\hat{\mathcal{V}}$  contains all nodes in the original graph  $\mathcal{G}$ .
- 3) For each edge  $e_j = (v_s, v_d) \in \mathcal{E}$  (in the original DFA graph), create a new node  $\hat{v}_{e_j}$  in  $\hat{\mathcal{V}}$ , and add edges  $(v_s, \hat{v}_{e_j})$  and  $(\hat{v}_{e_j}, v_d)$  to  $\hat{\mathcal{E}}$ .
- 4) Embed the graph  $\hat{\mathcal{G}}$  using the meta-embedder  $q_m$ .
- 5) Obtain the entire graph representation by aggregating node embeddings sampled via random walks initiated from the initial state node [27].

<sup>2</sup>GNNs can process edge features but we found using node features led to better empirical performance.

In this work, both edge-embedder  $q_e$  and meta-embedder  $q$  are multi-layer Graph Convolutional Networks (GCNs) [28] with four layers.

**Embedder Training.** To train our meta embedder, we use a *triplet loss* that encourages formulae embeddings to be close to satisfying traces, and far from unsatisfying traces. Let  $z_\varphi = q(A_\varphi)$  be the DFA embedding. Define  $z_\top = q(w_\top)$  and  $z_\text{F} = q(w_\text{F})$  as the trace embeddings for a satisfying and unsatisfying trace, respectively. Note that the graph structures for traces are linear; edges are a conjunction of propositions at each respective time-step. Our triplet loss is a hinge loss:

$$\ell_{\text{triplet}}(A_\varphi, w_\top, w_\text{F}) = \max\{d(z_\varphi, z_\text{F}) - d(z_\varphi, z_\top) + m, 0\}, \quad (7)$$

where  $d(x, y)$  is the squared Euclidean distance between  $x$  and  $y$ , and  $m$  is the margin. Training the embedder entails optimizing a combined loss:

$$L_{\text{emb}} = \sum_{A_\varphi} \sum_{w_\top, w_\text{F}} \ell_{\text{triplet}}(A_\varphi, w_\top, w_\text{F}), \quad (8)$$

where the summation is over formulas and associated pairs of satisfying and unsatisfying traces in our dataset. In practice, pairs of traces are randomly sampled for each formula during training. In our experiments, the edge-embedder  $q_e$  is trained first (and fixed), followed by the meta-embedder  $q_m$ . We found this approach to improve training stability, and leave alternative training schemes (e.g., joint or interleaved) to future work.

**The Logic Loss.** We train a given target task model  $h$  by augmenting its per-datum task loss with a logic loss  $\ell_{\text{logic}}$ ,

$$\ell = \ell_{\text{task}} + \lambda \ell_{\text{logic}}, \quad (9)$$

where  $\ell_{\text{logic}} = \|q(A_\varphi) - q(h(x))\|_2^2$  is the embedding distance between the DFA (converted from  $\text{LTL}_f$  formula related to the input  $x$ ) and the predicted output sequence  $w = h(x)$ . The task-specific loss  $\ell_{\text{task}}$  depends on the application, e.g., cross-entropy for classification. Lastly,  $\lambda$  is hyper-parameter that trades-off task performance and compliance with prior knowledge.

### IV. RELATED WORK

T-LEAF is related to a body of work on embedding symbolic logic for prediction [29]–[35] and reasoning tasks [36]–[38]. The key difference is that past work has largely focused on general propositional and first-order logic. In contrast, T-LEAF embeds temporal logic, which is useful for many robotics applications.

Prior work using  $\text{LTL}_f$  for robotics has focused primarily on planning [6]–[8], [39]—e.g., synthesizing controllers from the product of  $\text{LTL}_f$  automata and environment automata—and using temporal-logic checkers to shape reward functions [9]–[11], [40]–[42]. Recently, a differentiable LTL loss was proposed in [43]. However, it is limited to continuous control or regression tasks; in binary symbolic domains, the loss is non-differentiable and equivalent to a model checker.

TABLE I: Prediction Accuracy (with Std. Error) on Synthetic Datasets with Varying Complexity. Highest Accuracies in **Bold**.

Embedder Input	Low	Moderate	High
Syntax tree	56.16 (0.92)	54.13 (0.62)	50.00 (0.00)
DFA	<b>83.43</b> (0.73)	<b>78.43</b> (1.54)	<b>66.90</b> (0.70)

Different from the work above, T-LEAF is designed to improve “target” deep sequential models by embedding prior knowledge. Compared to a direct application of a model checker, we posit that learned embeddings may yield a more informative loss and gradients; intuitively, the gradients may provide “directional” information towards regions of complying models. To our knowledge, the T-LEAF logic loss is only differentiable loss for (binary) propositional temporal logic.

## V. EXPERIMENT: MODEL CHECKING

In this section, we focus on testing whether deterministic finite-state automata (DFA) are more amenable to embedding compared to  $LTL_f$  syntax trees. Specifically, we conduct an experiment using a model checking problem: given the embedding of a  $LTL_f$  formula  $\varphi$  and the embedding of a trace  $w$ , predict whether  $w$  models  $\varphi$ .

**Dataset and Experiment Setup.** The complexity of formulae is (coarsely) reflected by its number of propositions  $n_v$  and the syntax tree width  $w_t$ . As such, we synthesized three datasets with  $(n_v, w_t) = (3, 10), (3, 20), (6, 20)$ , corresponding to “Low”, “Moderate” and “High” complexities, respectively. Formulae were translated into DFA using LTLfKit [25], [26].

Our T-LEAF hierarchical embedder uses 4 graph convolution layers, with 100 hidden units per layer, and outputs a 200-dimension embedding for each input automata or trace. The neural network used for classification is a multi-layer perceptron with 512 and 128 hidden units in the first and second layers, respectively.

**Results.** Prediction accuracies with standard errors over three independent runs are reported in Table I. Across the datasets, we observe that using DFA graphs results in far better accuracy compared to  $LTL_f$  syntax trees; the difference in performance is large  $\approx 16\% - 27\%$ . This difference can be explained by examining the embedding spaces; Fig. 2 illustrates a 2D projection of a sample formula and its satisfying/unsatisfying traces. The Syntax Tree embeddings appear to lack structure. In contrast, there is a clear separation between satisfying traces and unsatisfying traces in the DFA embedding space, and the satisfying traces are closer to the formula. These results support our hypothesis that DFA can more easily be embedded using graph neural networks, compared to syntax trees.

## VI. SEQUENTIAL HUMAN ACTION RECOGNITION

In this section, we show how T-LEAF can be applied to Sequential Human Action Recognition. This task is important in many contexts, e.g., a robot needs recognize human

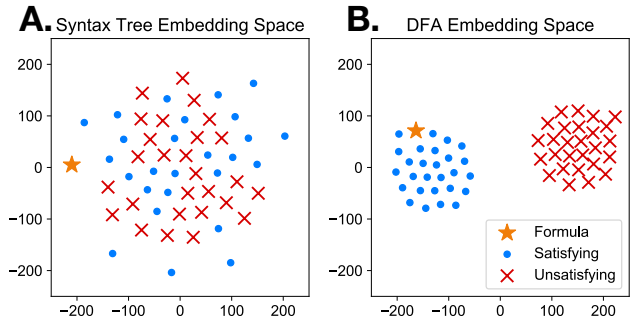


Fig. 2: t-SNE 2D-projection of the learnt embedding spaces for (A.) Syntax Tree inputs, and (B.) DFA inputs. The DFA space shows a clear separation between satisfying traces (blue dots) and unsatisfying traces (red crosses) for the shown formula (star).

actions to assist appropriately. We focus on a cooking domain where a model is trained to predict an  $(action, object)$  pair sequentially given temporal visual information (i.e., image frames). Target models could benefit from temporal common-sense knowledge, such as the affordances of certain objects and the likely ordering between actions. Our goal was to establish if incorporating such prior knowledge via T-LEAF results in better task models.

**Dataset.** We evaluated our method on the *Tasty Video Dataset* [44], which contains 4027 recipe videos involving 1199 unique ingredients. Each recipe is self-contained with an ingredient list, step-wise instructions with temporal alignment, and a video demonstrating the preparation. The Tasty Videos are captured with a fixed overhead camera and focus entirely on preparation of the dish. However, target labels ( $(action, ingredient)$  pairs for each step) were not provided, so we crowd-sourced labels for 500 videos. To balance the labels, we trimmed the number of action and ingredient classes (to 64 actions and 155 ingredients) by removing low frequency classes and combining similar classes, such as *cheddar* and *cheese*.

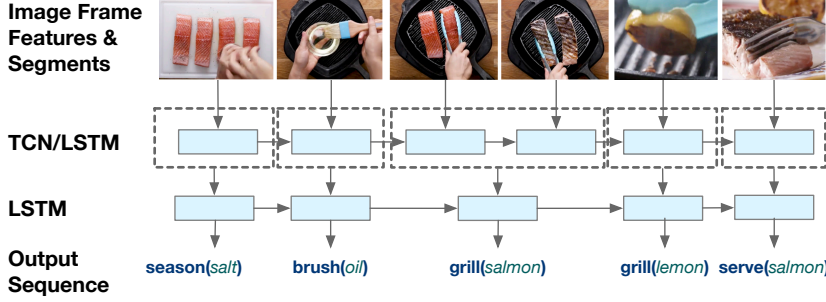
**Temporal Knowledge.** We used  $(action, ingredient)$  pairs as propositions, such as  $(grill, salmon)$ , which is **true** at time step  $t$  if it exists in the  $t^{th}$  step. Two type of rules are defined over the propositions:

- *Affordance Constraints.* These constraints help to eliminate action and ingredient pairs where the action is not afforded by the object, such as  $(cut, milk)$ .
- *Ordering Constraints.* Certain actions must come before other actions if they appear in the same video and are applied on the same object. For example,  $(rinse, cabbage)$  should take place before  $(cook, cabbage)$ . Note that there is no constraint if only one of pairs takes place, or they are not applied on the same object.

We extract affordance constraints and ordering constraints from the complete dataset to form our knowledge base  $K$ .

**Hierarchical Embedder.** The meta embedder  $q$  used was a four-layer GCN with the same structure described in Sec. V. The edge embedder  $q_e$  was two-layer GCN (256 hidden units per layer) that outputs a 100-dimensional embedding.

A.



B.

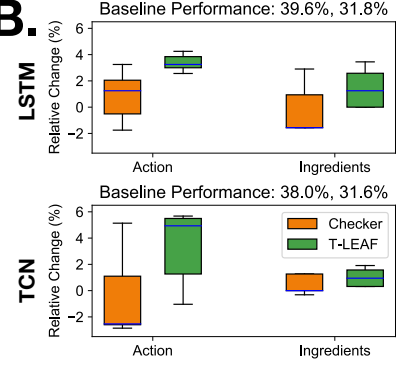


Fig. 3: Sequential Human Action Recognition Experiment. (A.) An overview of the target model. We use ResNet-50 [45] to extract visual features of each video frame, which are aggregated using a LSTM/TCN. The model is topped with two independent bi-directional LSTMs that predict the action and ingredient pair. (B.) Boxplots of relative changes after applying T-LEAF to reference models. T-LEAF improves performance on both the LSTM and TCN target models, as indicated by the positive changes, and generally outperforms target model training using a model checker.

We trained the hierarchical embedder using  $LTL_f$  formulae constructed from  $K$ . For each sample  $x_i$ , we constructed a  $LTL_f$  formula  $\varphi_i$  that only contained propositions and constraints related to current video sample. Specifically, we enumerated all  $(action, ingredient)$  pairs in current video, and extracted relevant constraints from  $K$  if all propositions, i.e., action ingredient pairs, involved in the constraint appeared in the video. The constraints were then combined together to form an  $LTL_f$  formula  $\varphi_i$  specific to the video.  $\varphi_i$  was then translated into a DFA  $A_{\varphi_i}$ . The edges of  $A_{\varphi_i}$  are propositional formulae in Conjunctive Normal Form (CNF), which were fed into edge embedder  $q_e$ . To obtain the vector representation of a given proposition, we concatenated the GLoVe [46] embeddings of the action and ingredient words (100-dimensions each), resulting an 200-dimension vector for each proposition. The edge embedder  $q_e$  was trained first using the triplet loss described in Sec. III, and then fixed to generate edge features of DFA graph  $A_{\varphi_i}$  during the training of meta embedder  $q$ .

**Target Model.** In this experiment, the target model  $h$  is a sequential human action recognition model (Fig. 3.A.). As visual segmentation was not the main focus of our work, we assumed the temporal alignment boundaries as given. We tested two slightly different models that use either a bi-directional LSTM [47] (with 1024 hidden units) or a temporal convolutional network (TCN) [48] to aggregate the video features. The TCN is a multi-layer one-dimension convolutional neural network (conv1d) with different dilation factors across layers; it serves as a representative of a state-of-the-art model for temporal data. The TCN in our experiment has 3 channels of conv1d with 800 hidden units and kernel size 2.

The input to the model  $h$  is the video segment feature sequence. Suppose the  $j^{\text{th}}$  video segment  $c_j$  is composed of  $L$  frames, i.e.  $c_j = \{f_t^j\}_{t=1,2,\dots,L}$ . Each frame  $f_t^j$  is represented as a feature vector (2048-dimensions), which is the output of last fully-connected layer before the softmax

layer in ResNet-50 [45]. We obtain the segment vector  $z_j$  by applying max pooling on output of LSTM or TCN. Then,  $z_j$  is passed to two independent one-layer bidirectional LSTMs for action and ingredient classification. The prediction generated by model at each time step  $j$  is an action-ingredient pair  $(a_j, o_j)$ . The complete prediction sequence for a video sample  $i$  is  $w_i = \{(a_1^i, o_1^i), (a_2^i, o_2^i), \dots, (a_n^i, o_n^i)\}$ , where  $n$  is the number of steps in the video.

**Target Model Training via T-LEAF.** Training the target model  $h$  using T-LEAF is straightforward given the hierarchical embedder  $q$ . For each input  $i$ , we simply compute the logic loss

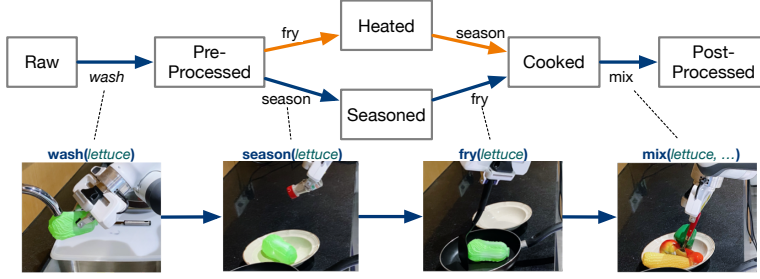
$$\ell_{\text{logic}} = \|q(A_{\varphi_{w_i}}) - q(w_i)\|_2^2.$$

The DFA  $A_{\varphi_{w_i}}$  was obtained by extracting the constraints related to the predicted sequence  $w_i$  from the knowledge base  $K$  (in a similar manner as the training data for the hierarchical embedder). We then optimized the total loss  $\ell = \ell_{\text{task}} + \lambda \ell_{\text{logic}}$ , where the  $\ell_{\text{task}}$  is the cross entropy loss and we set  $\lambda = 5.0$ . Optimization was carried out using Adam [49] with learning rate  $10^{-4}$ .

**Results.** We compared the relative change in accuracy between five pairs of models; each pair was initialized with the same random seed, but trained using only the task loss (the reference model) or using the task loss together with the T-LEAF logic loss or a model-checker loss. Fig. 3.B. summarizes our results in box-plots. We observed T-LEAF improves action prediction by an average of  $\approx 3.3 - 3.4\%$ . The difference in average ingredient prediction accuracy is smaller ( $\approx 1.0 - 1.5\%$ ), possibly due to the logic statements being defined over actions, e.g., the ordering temporal logic was only relevant for actions. Nevertheless, we see improvements over the reference models simply by incorporating temporal logic via our embedder. The relative differences are also consistently higher than the model checker, suggesting that learned embeddings can be more effective at training models to use prior knowledge.



### A. Creative Cooking Dependency Constraints:



### B.

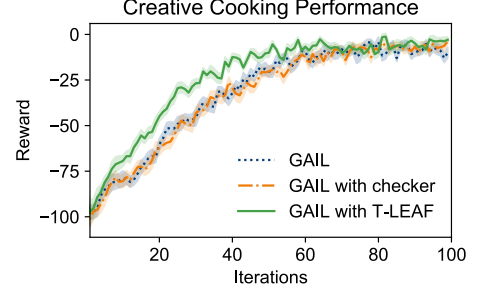


Fig. 4: Creative Cooking Imitation Learning Experiment. (A.) The dependency constraints in the environment, along with a sample sequence of actions executed by the robot. The ingredient (*lettuce*) changes its status from *raw* to *post-processed* as actions are performed. Note an alternative path is permitted by the constraints (the lettuce is fried before seasoned). (B.) T-LEAF improves the convergence rate of the GAIL reference model compared to a model-checker loss.

## VII. IMITATION LEARNING FOR CREATIVE COOKING

Our second experiment involves imitation learning in an object-manipulation and temporal-reasoning environment that we call “Creative Cooking”. The robot starts with a set of randomly sampled raw ingredients (e.g. *apple* and *corn*) and is tasked to prepare a meal. Imitation learning is useful in this scenario where the reward function is difficult to define; the evaluation criteria are complex and diverse.

Unlike the previous experiment, our goal is to obtain a policy  $\pi(a_t|s_t)$ , rather than a sequence model. The robot has access to the environment and a set of expert trajectories. As in many settings, the number of expert demonstrations is small due to high collection cost. We assume that the robot cannot query the expert for more data nor directly observe the reward signals. However, in addition to demonstrations, the experts could provide domain knowledge, high-level guidance, and preferences, which may be hard to learn from limited demonstrations. These type of knowledge could be expressed as  $LTL_f$ , and integrated into our policy via T-LEAF. Due to space constraints, we focus on the key concepts and relegate details to the appendix.

**Environment.** We model Creative Cooking as a discrete Markov Decision Process (MDP) where the each state comprises the sampled ingredients and their corresponding status. There are 50 ingredients and 15 ingredient properties (e.g., *liquid*, *seasoning*). Each ingredient has at least one *property* and one of six possible *statuses*. All ingredients start from the *raw* status. At each step, the robot chooses one out of 31 possible actions (e.g. *wash*, *cut*, *cook*) and the ingredient(s) to apply the action on. Each action is associated with affordance constraints and pre-requisite status requirements (e.g., a carrot needs to be in a *pre-processed* status before cooking). Failing to meet the affordance constraints or the pre-requisite requirements renders the action infeasible and state remains unchanged. Otherwise, the ingredient will change to a specified status. The task goal is to change all ingredients to their required status, which depends on ingredient properties.

**Temporal Knowledge and Hierarchical Embedder.** The

propositions here are action ingredient combinations; the proposition, *action(ingred)* at time step  $t$  is *True* if it’s selected by the robot at time  $t$ , and *False* otherwise. The expert provides affordance constraints and the dependency relationships (Fig. 4.A.) expressed in  $LTL_f$ , which comprises our knowledge base.

Our embedder structure is similar to the previous experiment. As the training dataset, we randomly sample 300 sub-formulae from the knowledge base along with 10 satisfying assignments and 10 unsatisfying assignments. As before, the embedder is trained and fixed before imitation learning.

**Target Model and Training via T-LEAF.** We use GAIL [50] as the imitation learning reference method. During training, we extract the related clauses for the current ingredient to form the DFA  $A_\varphi$ , which is then used in the logic loss. The policy is trained using a linear combination of the logic loss and the GAIL discriminator loss with  $\lambda = 1.0$ .

**Results.** Fig. 4.B. summarizes our results and shows the accumulated rewards across the training iterations (averaged over 20 independent runs, with standard-errors shaded). The final performance of the models are similar, but GAIL with T-LEAF loss converges faster compared to a model checker loss. These results again support the notion that DFA embeddings can be used to train deep models. Note also that the formula related trajectories were previously *unseen* by T-LEAF embedder, which suggests T-LEAF is able to generalize to unseen formulae.

## VIII. CONCLUSION AND FUTURE WORK

This paper proposes T-LEAF, a novel approach for utilizing symbolic temporal logic by embedding  $LTL_f$  DFA via graph neural networks. Empirical results are promising: T-LEAF improved deep models for sequential human action recognition and imitation learning. We believe T-LEAF is a first-step towards an alternative approach for training deep models to be compliant with pre-existing knowledge. We look forward to future improvements and applications in domains where utilizing prior knowledge can be beneficial, such as medical and human-collaborative robotics.

## REFERENCES

- [1] Y. Wu, J. Lim, and M.-H. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.
- [3] H. Soh, Y. Xie, M. Chen, and D. Hsu, "Multi-task trust transfer for human-robot interaction," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 233-249, 2020.
- [4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. R. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, 2017.
- [5] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, 2015.
- [6] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *ICRA*, 2005.
- [7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *ICRA*, 2007.
- [8] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *International Journal of Robust and Nonlinear Control*, 2011.
- [9] I. P. Min Wen and U. Topcu, "Learning from demonstrations with high-level side information," in *IJCAI*, 2017.
- [10] A. A. Mohammadhossein Hasanbeig and D. Kroening, "Logically-correct reinforcement learning," in *arXiv preprint arXiv:1801.08099*, 2018.
- [11] M. L. Littman, U. Topcu, C. I. Jie Fu, M. Wen, and J. Mac-Glashan, "Environment-independent task specifications via gltl," in *arXiv preprint arXiv:1704.04341*, 2017.
- [12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, March 2020.
- [13] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," in *arXiv preprint arXiv:1812.08434*, 2018.
- [14] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977.
- [15] P. Goyal, S. Niekum, and R. J. Mooney, "Using natural language for reward shaping in reinforcement learning," in *IJCAI*, 2019.
- [16] J. Dzifcak, M. Scheutz, C. Baral, and P. W. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *ICRA*, 2009.
- [17] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
- [18] K. Y. Rozier, "Survey: Linear temporal logic symbolic model checking," *Comput. Sci. Rev.*, 2011.
- [19] T. French, J. McCabe-Dansted, M. Reynolds, and D. Larchey-Wendling, "Model checking general linear temporal logic," in *Automated Reasoning with Analytic Tableaux and Related Methods*. Springer Berlin Heidelberg, 2013.
- [20] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artif. Intell.*, 2000.
- [21] J. Baier and S. McIlraith, "Planning with first-order temporally extended goals using heuristic search," in *Proceedings of the National Conference on Artificial Intelligence*, 2006.
- [22] A. Camacho, E. Triantafyllou, C. J. Muise, J. A. Baier, and S. A. McIlraith, "Non-deterministic planning with temporally extended goals: Ltl over finite and infinite traces," in *AAAI*, 2017.
- [23] J. Baier, F. Bacchus, and S. McIlraith, "A heuristic search approach to planning with temporally extended preferences," *Artif. Intell.*, 2009.
- [24] M. Bienvenu, C. Fritz, and S. A. McIlraith, "Specifying and computing preferred plans," *Artif. Intell.*, 2011.
- [25] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi, "Symbolic ltl synthesis," in *IJCAI*, 2017.
- [26] A. Camacho, J. A. Baier, C. J. Muise, and S. A. McIlraith, "Finite LTL synthesis as planning," in *ICAPS*, 2018.
- [27] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, 2018.
- [28] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
- [29] Y. Xie, Z. Xu, K. S. Meel, M. S. Kankanhalli, and H. Soh, "Embedding symbolic knowledge into deep networks," in *NeurIPS*, 2019.
- [30] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Van den Broeck, "A semantic loss function for deep learning with symbolic knowledge," in *ICML*, 2018.
- [31] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *AAAI*, 2016.
- [32] M. Allamanis, P. Chanthirasegaran, P. Kohli, and C. A. Sutton, "Learning continuous semantic representations of symbolic expressions," in *ICML*, 2017.
- [33] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *ACL*, 2015.
- [34] P. Le and W. Zuidema, "Compositional distributional semantics with long short term memory," in *\*SEM@NAACL-HLT*, 2015.
- [35] X.-D. Zhu, P. Sobhani, and H. Guo, "Long short-term memory over recursive structures," in *ICML*, 2015.
- [36] D. Lee, C. Szegedy, M. Rabe, S. Loos, and K. Bansal, "Mathematical reasoning in latent space," in *ICLR*, 2020.
- [37] J. W. Mingzhe Wang, Yihe Tang and J. Deng, "Premise selection for theorem proving by deep graph embedding," in *NIPS*, 2017.
- [38] A. Paliwal, S. Loos, M. Rabe, K. Bansal, and C. Szegedy, "Graph representations for higher-order logic and theorem proving," in *arXiv preprint arXiv:1905.10006*, 2019.
- [39] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, 2009.
- [40] M. Wen and U. Topcu, "Approximately correct learning in stochastic games with temporal logic specifications," in *IJCAI*, 2016.
- [41] C. I. V. Xiao Li and C. Belta, "Reinforcement learning with temporal logic rewards," in *IROS*, 2017.
- [42] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an rl agent using ltl," in *AAMAS*, 2018.
- [43] S. R. Craig Innes, "Elaborating on learned demonstrations with temporal logic specifications," in *RSS*, 2020.
- [44] F. Sener and A. Yao, "Zero-shot anticipation for instructional activities," in *ICCV*, 2019.
- [45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
- [46] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014.
- [47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.
- [48] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," in *arXiv preprint arXiv:1803.01271*, 2018.
- [49] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
- [50] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *NIPS*, 2016.

## APPENDIX

In the following, we provide additional details on the Creative Cooking experiment.

**MDP States.** The environment states consist of both sampled ingredients and their corresponding status. There are 50 ingredients and 15 ingredient properties (e.g., *liquid*, *seasoning*). Each ingredient has at least one property and exactly one status. Properties are used to define and determine action feasibility and environment transitions, and are not reflected in state representation. Each item’s status starts from *raw* and changes depending on the robot actions and environment constraints. There are six possible statuses: *raw*, *pre-processed*, *heated*, *seasoned*, *cooked*, *post-processed*. The transitions between these different status is illustrated in Fig. 4.A.

At initialization of each trajectory (environment reset), 5 ingredients sampled from ingredients set uniformly without replacement, together with a special ingredient *mixture* which handles ingredients merging, composes *trajectory ingredients*. The *trajectory ingredients*’s status are initialized to *raw*, and change via actions as described below. Properties of each ingredient, except *mixture*, remain unchanged by actions. As such, the state representation comprises the 6 *trajectory ingredients* and their respective statuses, resulting a 12-dimensional vector, i.e. (*mixture idx*, *mixture status idx*, *ingred<sub>1</sub> idx*, *ingred<sub>1</sub> status idx*, ..., *ingred<sub>5</sub> idx*, *ingred<sub>5</sub> status idx*).

**MDP Actions.** At each step, the robot chooses one action and the ingredient(s) to apply the action on. There are 31 possible actions classified to 10 categories (e.g. *cut*, *wash*, *cook*). All actions are applied on exactly one ingredient, except action *top-with* and *combine* which are applied on two ingredients. These actions (together with *add*) are specially

designed to handle the merging of ingredients:

- *add(ingred)*: Add *ingred* in *mixture*. The properties of *ingred* will be appended to *mixture*. The original position of *ingred* in the state representation will be replaced by zeros, which represents that the ingredient no longer exists separately from the mixture.
- *combine(ingred<sub>1</sub>, ingred<sub>2</sub>)*: The effect is equivalent to *add(ingred<sub>1</sub>)* and *add(ingred<sub>2</sub>)*, i.e. add *ingred<sub>1</sub>* to *mixture* then add *ingred<sub>2</sub>* to *mixture*.
- *top-with(ingred<sub>1</sub>, ingred<sub>2</sub>)*: Top *ingred<sub>1</sub>* with *ingred<sub>2</sub>*. *ingred<sub>2</sub>*’s position in the state representation will be replaced by zeros, while the status of *ingred<sub>1</sub>* will change appropriately (its properties remain unchanged). For example, topping meat with salt changes the status of the meat to *seasoned*.

**MDP Transitions** Each action is associated with affordance constraints and pre-requisite status requirements. Failing the meet either of these renders the action infeasible and state remains unchanged. Otherwise, the ingredient will change to a specified status.

Affordance constraints are common-sense facts about whether actions can only or cannot be applied to some ingredients (e.g., we cannot cut milk). Pre-requisite status requirements refer to status dependencies (Fig. 4.A.) (e.g., a carrot needs to be in a *pre-processed* status before cooking). Our temporal knowledge consists of both affordance constraints and pre-requisite status requirements, i.e., the dependency relationships.

**MDP Rewards** The task goal is to change all ingredient to their required status, which depends on ingredient properties. The robot gets a large positive reward, +20, if the goal is reached. There is a −1 penalty for each step taken and an extra −1 penalty if the action chosen is infeasible.