Towards Neuro-Symbolic Video Understanding

Supplementary Material

9. Appendix

9.1. Examples of Temporal Logic

Linear Temporal Logic (LTL). In logic, linear temporal logic (LTL), also known as linear-time temporal logic, is one of the most commonly used temporal logic. LTL encodes formulae about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc. LTL is composed of a set of atomic propositions (propositions that are either true or false and cannot be divided into multiple propositions), first-order propositional logic operators ($\wedge$ and, $\vee$ or, $\neg$ not, etc.), temporal operators ($\Box$ always, $\Diamond$ eventually, X next, U until, R release, etc.). The common temporal operators in LTL are

- $\Box\phi$: ϕ is true on the entire subsequent path.
- $\Diamond\phi$: ϕ is true at least once on the subsequent path.
- $\mathsf{X}\phi$: ϕ is true at the next state.
- $\phi_1\mathsf{U}\phi_2$: ϕ_1 must be true until the first time ϕ_2 is true, then ϕ_2 must be true for all the future states.
- $\phi_1\mathsf{R}\phi_2$: ϕ_1 has to be true until and including the point where ϕ_2 first becomes true.

The syntax of LTL is defined as

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathsf{X}\Phi \mid \Phi\mathsf{U}\Phi.$$

Computational Tree Logic (CTL). Computational tree logic (CTL) is a branching-time temporal logic. CTL's model of time is a tree structure in which the future is not determined. Many possible paths are branching out in the future, and one of the paths may turn into reality. CTL encodes formulae about all these branches of the future, i.e., something eventually will happen among all the possible futures, or something happens in at least one of the future branches.

CTL shares the same set of first-order logic operators as LTL and includes all the temporal operators defined in the LTL section. In addition to the existing temporal operators, CTL includes two additional temporal quantifiers: All (A) and Exists (E). They are formally defined as:

- $\mathsf{A}\phi$: ϕ is true on all paths starting from the current state.
- $\mathsf{E}\phi$: ϕ is true on at least one of the paths starting from the current state.

The syntax of CTL is formally defined as

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathsf{E}\Box\Phi \mid \mathsf{EX}\Phi \mid \mathsf{E}[\Phi\mathsf{U}\Phi].$$

Note that compared to LTL, CTL can quantify multiple branches while LTL is capable of one path. However, LTL can encode formulas like "Eventually-Always something happens," while CTL is not capable of.

Probabilistic Computational Tree Logic (PCTL). Probabilistic CTL (PCTL) is an extension of CTL that allows for probabilistic quantification of described properties. The syntax and operations of PCTL are identical to CTL, except for the temporal quantifiers. Instead of the quantifiers (All and Exists) in CTL, PCTL uses probabilistic quantifiers $\mathsf{P}_{\sim\lambda}$. The syntax is formally defined as

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathsf{P}_{\sim\lambda}[\Box\Phi] \mid \mathsf{P}_{\sim\lambda}[\Phi\mathsf{U}\Phi],$$

where $\sim \in \{<, \leq, \geq, >\}$, and $\mathsf{P}_{\sim\lambda}$ indicates the required range of the probability of Φ being satisfied. For example, let $\sim =>$ and $\lambda = 0.6$, $\mathsf{P}_{>0.6}\Phi$ is True if and only if Φ is satisfied over 60% of the time, i.e., Φ has more than 60% probability of being satisfied.

Linear Temporal Logic over Finite Traces. LTL over infinite traces was introduced in the field of computer science as a specification language designed for concurrent programs [27]. It facilitates the characterization of temporal reasoning within the domains of propositions and first-order logic. In our approach, frames of interest are considered as a finite set of frames extracted from a video. We can employ *Linear temporal logic over finite traces* (LTL_f) [3, 5, 6, 33] to handle finite sequences encompassing finite-length trajectories, frames from video, system procedures. Given a set of atomic propositions P, the syntax of LTL_f formulas is defined as follows:

$$\phi ::= p \in P \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{X}\phi \mid \phi_1\mathcal{U}\phi_2. \tag{12}$$

where $p \in P$ is an atomic proposition. It incorporates conditional operators such as AND ($\wedge$), OR ($\vee$), XOR($\oplus$), NOT($\neg$), IMPLY($\rightarrow$), along with temporal operators, ALWAYS($\Box$), EVENTUALLY($\Diamond$), NEXT($\mathcal{X}$), and UNTIL($\mathcal{U}$).

9.2. LLM Benchmark Prompts

We design benchmarks for a balanced comparison that uses LLMs for reasoning over per-frame annotations instead of Temporal Logic (TL). Initially, neural perception models provide per-frame annotations to LLMs such as GPT. These LLMs are then prompted to identify scenes of interest according to specified propositions and TL specifications. We use these benchmarks to evaluate the impact of video length on scene identification performance. The prompts and methodologies applied in these comparisons are elaborated as follows:

Prompt for LLMs

```
You are a Language Assistant to enable searching
a video frame for objects satisfying a condition.
TASK DESCRIPTION: I will be giving you the list
of objects detected in the video and a condition
from the search.  The list of objects will be
specified in this format and include those frames
where an object is detected:
- Frame 1:  [object1, object2, object3]
- Frame 2:  [object1, object2]
A condition is of the form:
- object1 Until object2,
- object1 Until (object2 and object3),
- Eventually object 1,
- Always object 2, etc.
You can use the list of objects detected in the
video to help you.  You don't have access to
the video.  Your job is to specify the frame
numbers where the rule is satisfied.  There can
be many such sequences.  If you cannot find any
sequence, you must return output [None].  To
clarify, you will be looking for frames where the
rule is satisfied.  The output is a list of frame
numbers, such as [1,20, 25, 50], specifying the
frames where the rule holds.  An example of the
list of detections per frame is as follows:
- 01:  [person, car]
- 02:  [person, car]
- 10:  [person, car]
- 30:  [truck]
- 40:  [car]
- 44:  [person]
- 50:  [car, truck]
Case 1:  'person' until 'truck' gives all frames
of 'person' and 'truck' if 'truck' shows up.

Case 2:  Eventually 'person' gives all frames of
'person' eventually showing up.

Case 3:  'car' and 'truck' should give all frames
of 'car', and 'truck' showing up in the same
frame.

Case 4:  Always 'car' should give all frames of
'car' always showing up.

Case 5:  ('car' and 'truck') until 'person' gives
all frames of 'car' and 'truck' showing up in the
same frame if 'person' shows up; all frames of
'person' included.
```