Temporal Behavior Trees for Reinforcement Learning: Specifications as Rewards, Curricula, and Diagnostics

Vladimir Krsmanović[1*] **Bernd Finkbeiner**[1,2]
[1]*CISPA Helmholtz Center for Information Security* [2]*Technical University of Munich*
{vladimir.krsmanovic,finkbeiner}@cispa.de

Abstract

Reinforcement learning from formal specifications often separates task definition, reward shaping, curriculum design, and policy diagnostics into different mathematical objects. This fragmentation requires additional conversion steps and discards structure already present in the specification, such as subtask boundaries, ordering constraints, and quantitative measures. We introduce TBTCL (Temporal Behavior Tree Centric Learning), a framework that uses Temporal Behavior Trees (TBTs) as a unified interface for specification-guided reinforcement learning. A single TBT specifies the task, induces dense robustness-based rewards, defines a curriculum, and supports better understanding of learned policies. By adapting TBT syntax and semantics for specification-guided reinforcement learning, we provide a formalism that is more expressive than popular specification languages such as LTL and inherently hierarchical. We present algorithms to automatically extract a curriculum DAG and derive dense, robustness-based rewards directly from TBT semantics. Furthermore, we utilize TBT trace segmentation to build understandable competence profiles and automatically adapt the reward toward bottlenecks. We evaluate TBTCL on six specifications across discrete and continuous domains and show that Temporal Behavior Trees can serve as a specification, a reward, a curriculum, and a diagnostic.

1 Introduction

Temporal Logics are a powerful tool in reinforcement learning, enabling the formal specification of tasks and serving as the engine of Specification-Guided Reinforcement Learning. Most standard approaches convert a logical specification into a deterministic finite automaton, which is then used to guide the learning process. While this conversion is necessary to translate standard logics such as LTL [Pnueli, 1977] into a form suitable for RL, it incurs a high cost as it loses crucial information at each step. Properties such as subtask boundaries, quantitative performance measures, hierarchical structures, and task orderings and priorities are already present in the logical specification but are discarded by most methods during intermediate conversion steps.

The core cause of these problems is that most specification languages are not designed for purposes beyond task specification. A typical pipeline represents the task specification in one object, derives a reward in another, constructs a curriculum through an additional procedure and yet another representation, and then relies on separate diagnostics and derivations to understand how a learned policy behaves, performs, and fails. As such, a potential solution would be a specification language that provides the task definition, a dense reward, a curriculum, and behavioral diagnostics, without requiring separate conversion algorithms, representations, or user inputs for each. In this paper, we argue that a single formal object can simultaneously serve as the task definition, a dense reward source, a curriculum structure, and a diagnostic model.

*Corresponding Author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

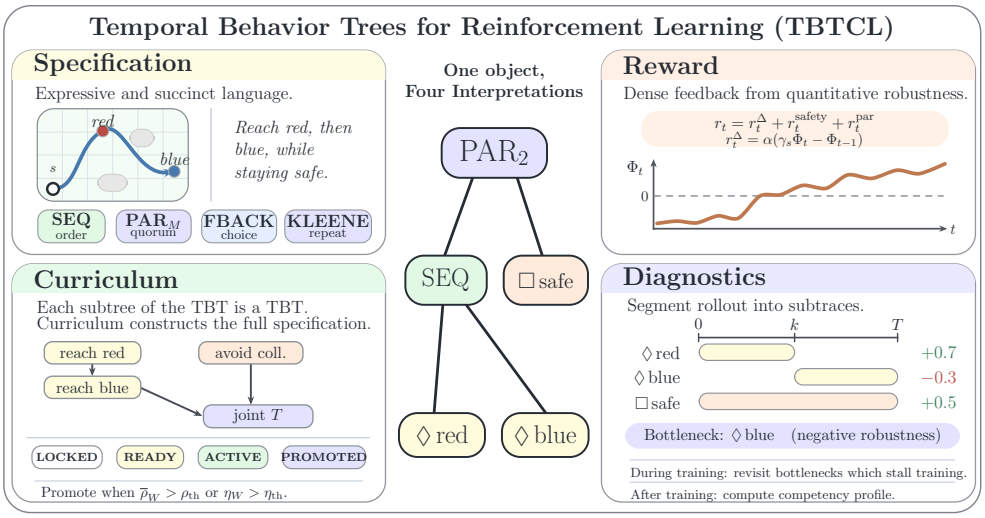


Figure 1: Temporal Behavior Trees provide a single formal object for task specification, dense robustness-based reward shaping, curriculum extraction, and trace-based diagnostics.

We show that Temporal Behavior Trees (TBTs) [Schirmer et al., 2024] have these important properties and provide a flexible formalism for reinforcement learning. Existing work has developed this formalism for monitoring and trace analysis of cyber-physical systems [Schirmer et al., 2024, Schirmer, 2025, Schirmer et al., 2026] and has demonstrated its power and flexibility in these applications.

A TBT composes STL formulas at its leaves with tree-level operators such as sequence, parallel composition, and fallback. The STL leaves provide quantitative robustness, while the tree structure describes how behaviors compose over a trace. This hierarchical structure, which combines STL formulas at the leaves with tree-level operators, enables a concise and expressive definition of subtasks and their interconnections, which are frequently goals in reinforcement learning. This combination further enables more concise and expressive specifications than those of commonly used logics such as LTL. Furthermore, the quantitative semantics encoded by a robustness metric for TBT yields a dense reward signal and builds on previous work in the field that adapted the robustness of logics such as STL for Reinforcement Learning [Várnai and Dimarogonas, 2020]. Finally, TBTs support trace decomposition, connecting policy rollouts directly to the specification expressed as a TBT without additional steps, and can be used to evaluate policy behavior and identify bottlenecks and weak points.

Building on these properties, we adapt the Temporal Behavior Trees for reinforcement learning and demonstrate their use in specification, training, and evaluation. We present TBTCL (Temporal Behavior Tree Centric Learning), a framework that exploits this unified structure. Given a TBT specification, TBTCL extracts a curriculum represented by a directed acyclic graph (DAG) by structural recursion over the tree, trains a single policy through a bottom-up sequence of subformula reward functions, and advances stages based on achieved STL robustness rather than fixed episode budgets. When learning stalls, trace segmentation can identify the bottleneck subtask and redirect training to address it. Moreover, the same segmentation algorithm that drives online curriculum adaptation can also serve as a runtime diagnostic mechanism, decomposing any evaluation trace into per-subtask quality scores directly aligned with TBT semantics, without requiring any additional external mechanism.

We make the following contributions:

- We present Temporal Behavior Trees as a unified interface for specification-guided RL, adapting the semantics to reinforcement learning (Section 3.1).
- We use the quantitative robustness semantics of TBTs to derive a dense reward from the prefix robustness of the TBT (Section 3.2).
- We leverage the tree structure of TBTs and present an algorithm to extract a curriculum directly from their structure and semantics (Section 3.3).
- We demonstrate how the TBT trace segmentation mechanism can be used to understand policies and improve their learning by identifying bottlenecks during training (Section 3.4).

2 Background

Signal Temporal Logic. Signal Temporal Logic (STL) [Maler and Nickovic, 2004, Donzé and Maler, 2010] was developed for monitoring and verification of cyber-physical systems.

An STL formula is defined inductively as $\varphi ::= \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{[l,u]}\varphi \mid \Box_{[l,u]}\varphi \mid \varphi_1 \,\mathcal{U}_{[l,u]}\, \varphi_2$, where $\mu : \mathbb{R}^n \to \mathbb{R}$ is an atomic predicate over the state space and $[l, u]$ are optional time bounds (omitted $\Rightarrow$ full signal). $\Diamond$ (Eventually) requires satisfaction at some point; $\Box$ (Globally) requires it at all points; $\mathcal{U}$ (Until) requires the first operand to hold until the second becomes true.

It extends Boolean temporal logic to real-valued signals, replacing the binary satisfied/violated verdict with a *quantitative robustness* score. The robustness of a given STL formula is a real number whose sign indicates satisfaction and whose magnitude measures the margin. We provide a full definition of STL semantics, syntax and recursive robustness computation in Appendix C.

Temporal Behavior Trees. Temporal Behavior Trees (TBTs) [Schirmer et al., 2024] compose STL leaves with hierarchical operators, yielding a two-level language: tree structure for compositional organization, STL for quantitative leaf-level properties. A TBT leaf $\textsc{Leaf}(\varphi)$ evaluates an STL formula, while a sequence node $\textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2)$, a parallel node $\textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k)$ with quorum M, and a fallback node $\textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k)$ compose child behaviors over trace segments:

$$\mathcal{T} ::= \textsc{Leaf}(\varphi) \mid \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2) \mid \textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k) \mid \textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k).$$

Intuitively, SEQ searches for a cut point such that the prefix satisfies the first child and the suffix satisfies the second. $\textsc{Par}_M$ evaluates all children on the same segment and requires at least M to succeed (quorum). FBACK represents alternatives, selecting the child and suffix with maximal robustness. We provide a detailed description of the TBT syntax and semantics in Appendix C.

Rollouts as traces. We consider episodic reinforcement learning in a Markov Decision Process (MDP) with horizon H and initial-state distribution μ_0. A rollout under policy π, starting from $s_0 \sim \mu_0$, induces a finite trajectory $\sigma = (s_0, \ldots, s_H)$ of environment states, which we call a *trace* following the formal methods literature, optionally augmented with real-valued signals such as distances to goals, hazards, or objects. STL predicates are evaluated on this trace: an atomic predicate μ maps a state or signal value to a real number, with $\mu(s_t) > 0$ indicating that the predicate is satisfied at time t. We write $\sigma[l{:}u]$ to denote the contiguous segment from time l to u.

3 Method

We show that Temporal Behavior Trees can act as the specification *and* as a dense reward *and* as a curriculum *and* as a diagnostic metric. First, in Section 3.1, we argue that TBTs are an expressive and succinct specification language for describing complex behaviors and a powerful formalism for describing reinforcement learning tasks. Afterward, in Section 3.2, we show how STL's robustness metrics are inherited by TBTs and present adaptations that enable effective training directly using TBT semantics. In Section 3.3 we demonstrate how the tree structure of TBTs directly induces a curriculum schedule, and show theoretical results about it. Finally, in Section 3.4 we use TBTs' ability to segment traces (in our case, policy rollouts) to both better understand learned policies and guide the learning process towards better ones.

3.1 Temporal Behavior Trees as Specifications

Temporal Behavior Trees provide an expressive and concise specification language, owing to their hierarchical structure and expressiveness of the underlying semantics. Temporal Behavior Trees provide not only a more succinct way to write out many specifications, but are also known to be more expressive than other commonly used logics. As shown by Schirmer et al. [2024] building on proofs from Wolper [1983], Temporal Behavior Trees are strictly more expressive than STL, whose robustness semantics they inherit.

Each leaf of a TBT contains an STL formula describing a simple task, such as reaching a target or avoiding a hazard. Internal nodes specify how behaviors described by child nodes compose: a

Sequence node (SEQ) imposes temporal order (*reach A before reaching B*), a parallel node (PAR_M) defines simultaneous requirements (*reach A while avoiding C*), and a Fallback node (FBACK) represents alternatives (*reach either target A or target B*). The combination of simple-to-specify leaf properties with powerful operators enables users to specify hierarchical specifications rather than just flat, linear-time formulas.

We further adapt the TBT formalism by Schirmer et al. [2024] for reinforcement learning by providing an operator to describe repeated behaviors, $\text{KLEENE}(n, \mathcal{T}_c)$. This node enables the specification of $n{+}1$ successful repetitions of the child behavior $\mathcal{T}_c$. Its robustness can be computed as the best partition of the trace into $n{+}1$ segments, scored by the weakest repetition:

$$\rho(\sigma[l{:}u], \text{KLEENE}(n, \mathcal{T}_c)) = \max_{\mathcal{P}_{n+1}(l,u)} \min_{i=1}^{n+1} \rho(\sigma_i, \mathcal{T}_c), \tag{1}$$

where $\mathcal{P}_{n+1}(l, u)$ is the set of valid ordered partitions of $\sigma[l{:}u]$ into $n{+}1$ non-empty contiguous segments. This operator enables easier specification of recurring properties (*such as reach A 5 times*) and avoids the need to define the same behavior through lengthy unfoldings of a flat temporal formula [2].

As a concrete example of one task description, in Figure 2 we show an example specification in a Temporal Behavior Tree as well as its equivalent in a formula in Linear Time Temporal Logic (LTL) [Pnueli, 1977]. Due to the linear nature of LTL, many such specifications require a significantly larger number of operators to describe, whereas the tree structure of TBT permits significantly more concise, easier-to-specify task descriptions.

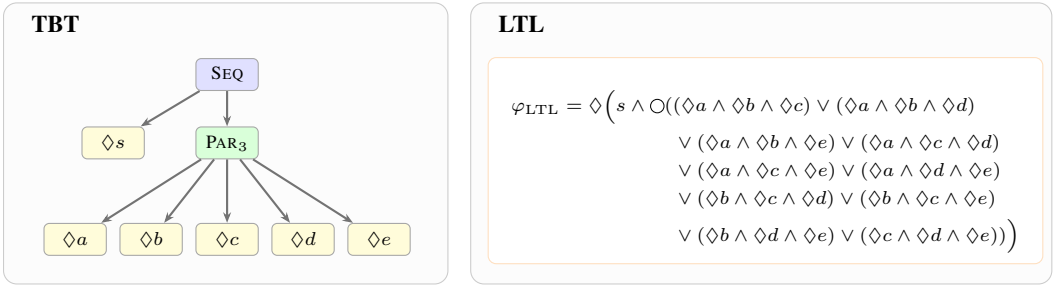


Figure 2: Comparison of equivalent TBT and LTL specifications. Both describe the desired behavior of "*reach S, and afterward at least three of the five possible targets*". While the LTL formula must explicitly enumerate structures, TBT can express them directly because of its tree structure.

3.2 Temporal Behavior Trees as Reward Functions

As TBTs inherit the robustness semantics of STL, we can use the dense reward signal to guide learning, similar to existing work applying STL to RL [Várnai and Dimarogonas, 2020]. A positive robustness score indicates that a rollout satisfies a TBT specification task, whereas a negative robustness score indicates a violation of that same specification, with the magnitude yielding a dense margin score.

Using this, we formulate our reward signal as a robustness-based potential over observed prefixes. For a TBT $\mathcal{T}$, we compute the *prefix potential* as $\Phi_t(\mathcal{T}) = \rho(\sigma[0{:}t], \mathcal{T})$, with $\Phi_t(\mathcal{T}) = 0$ when the prefix is too short to admit a valid decomposition. The dense term for the reward is expressed as the discounted potential difference.

$$r_t^{\Delta} = \alpha\Big(\gamma_s\, \Phi_t(\mathcal{T}) - \Phi_{t-1}(\mathcal{T})\Big), \qquad t = 1, \ldots, N, \tag{2}$$

with α being a reward scale coefficient and $\gamma_s \in (0, 1]$ a shaping discount. This term is the per-step form of quantitative semantics, rewarding policy changes that improve the robustness score of a TBT specification, thereby avoiding the need to convert to a different object or to use external mechanisms to produce a dense reward signal.

[2] The original TBT formalization by Schirmer et al. [2024] includes a $\star_n$ node that accepts *up to* n repetitions of a given behavior. While these semantics are a natural fit for runtime monitoring, they are less so for reinforcement learning, as *up to* specifications are trivial to satisfy - which is why we replace this node with KLEENE

While the STL robustness semantics are directly inherited for leaves, we have to adapt the reward formulation to account for the semantics of the other TBT nodes. For Leaves with safety properties (such as *"never go through A"*), the TBT semantics must account for the fact that, after a violation of a safety property, no future action can repair it [Baier and Katoen, 2008]. Because the robustness score defined in eq. (2) is determined by the previous state, this leads to a situation where no change is possible in the robustness, and the delta-based reward signal disappears [Várnai and Dimarogonas, 2020].

Similarly, the semantics of PAR_M nodes suffer from a similar issue. As the robustness is computed against the weakest child in the quorum of the PAR_M node, a step improvement in any child other than the one currently holding the M-th largest robustness value is masked, and there is no reward for such improvements [Várnai and Dimarogonas, 2020]. This leads to potentially beneficial policy changes not being properly rewarded. We combat these issues by adding additional terms adjusting the reward signal for these situations.

To provide a dense reward signal even after a safety violation, we introduce a term

$$r_t^{\text{safety}} = \beta \sum_{\square\varphi_g \in \mathcal{G}(\mathcal{T})} \min\big(0,\ \rho(\sigma[t{:}t], \varphi_g)\big). \tag{3}$$

where $\mathcal{G}(\mathcal{T})$ ranges over all leaves of $\mathcal{T}$ encoding safety properties. This term imposes a penalty proportional to the extent of the agent's violation of the safety property. Unlike the delta-based approach, r_t^{safety} accumulates the penalty for repeating violations and pushes the policy back towards safe behavior. This provides a dense signal even when there is no way to recover from violating a safety property in a Leaf.

For PAR_M nodes, TBTCL exposes the progress of every child as a second potential term with zero terminal value:

$$r_t^{\text{par}} = \gamma_s \Psi_t - \Psi_{t-1}, \qquad \Psi_t = \lambda_{\text{par}} \sum_{i \in \mathcal{C}(\mathcal{T}_o)} \Phi_t(\mathcal{T}_i) \ \text{ for } 0 \le t < N, \qquad \Psi_N = 0, \tag{4}$$

where $\mathcal{C}(\mathcal{T}_o)$ is the set of immediate children of the parallel objective $\mathcal{T}$ and λ_{par} a shaping scale hyperparameter. While quorum robustness remains the satisfaction score, this formulation ensures that an update that improves any child also improves the rewards, and not only for the one that currently holds the quorum back.

Combining the prefix potential reward and the two terms, TBTCL provides a potential-based reward directly from the TBT semantics

$$r_t = r_t^{\Delta} + r_t^{\text{safety}} + r_t^{\text{par}}, \tag{5}$$

We show several properties of the robustness-based reward, starting by showing that maximizing expected return is equivalent to maximizing the expected terminal robustness net of the safety penalty, U, and then relating U to the satisfaction of the specification. This gives a satisfaction-probability bound and, under additional assumptions, almost-sure satisfaction by return maximizers.

For a trace $\sigma = (s_0, \dots, s_N)$ we write $G(\sigma) = \sum_{t=1}^{N} \gamma_s^{t-1} r_t$ for the return of Eq. 5 and $S(\sigma) = \sum_{t=1}^{N} \gamma_s^{t-1} r_t^{\text{safety}} \le 0$ for the accumulated safety penalty, and write $U(\sigma) = \alpha\gamma_s^N \Phi_N(\mathcal{T}) + S(\sigma)$ for the discounted terminal potential net of this penalty.

Lemma 1 (Return identity)**.** *For every trace, $G(\sigma) = U(\sigma) - c(s_0)$ with $c(s_0) = \alpha\Phi_0(\mathcal{T}) + \Psi_0$ a function of the initial state alone, and $U(\sigma) \le 0$ whenever $\sigma \not\models \mathcal{T}$.*

Proof. The delta term telescopes

$$\sum_{t=1}^{N} \gamma_s^{t-1} \alpha\big(\gamma_s \Phi_t(\mathcal{T}) - \Phi_{t-1}(\mathcal{T})\big) = \alpha \sum_{t=1}^{N} \big(\gamma_s^t \Phi_t(\mathcal{T}) - \gamma_s^{t-1} \Phi_{t-1}(\mathcal{T})\big) = \alpha\big(\gamma_s^N \Phi_N(\mathcal{T}) - \Phi_0(\mathcal{T})\big), \tag{6}$$

the parallel term sums to $\gamma_s^N \Psi_N - \Psi_0 = -\Psi_0$ since $\Psi_N = 0$ and the safety term sums to $S(\sigma)$. Both initial potentials depend only on s_0, and so does $c(s_0)$ as well. As non-satisfying traces produce negative robustness scores [Fainekos and Pappas, 2009], if $\sigma \not\models \mathcal{T}$ it follows that $\Phi_N(\mathcal{T}) \le 0$ which together with $S \le 0$ leads to $U \le 0$. □

As the constant $c(s_0)$ is outside the policy's control, maximizing the expected return maximizes $\mathbb{E}_\pi[U]$. Since $U \leq 0$ on violating traces as per robustness semantics, this expectation yields a bound on the satisfaction probability.

Proposition 1 (Satisfaction-probability bound)**.** *Let $U^* = \sup_\sigma U(\sigma)$ over the feasible traces of a finite-horizon MDP and assume $0 < U^* < \infty$. Every policy π satisfies*

$$\Pr_\pi(\sigma \models \mathcal{T}) \;\geq\; \frac{\mathbb{E}_\pi[U(\sigma)]}{U^*} \;=\; \frac{\mathbb{E}_\pi[G] + \mathbb{E}_{\mu_0}[c(s_0)]}{U^*}.$$

The bound holds for every $\gamma_s \in (0, 1]$, $\alpha > 0$ and $\beta, \lambda_{\text{par}} \geq 0$.

Proof. By Lemma 1, $U \leq 0$ on violating traces, and $U \leq U^*$ on satisfying ones by definition of U^*. Splitting $\mathbb{E}_\pi[U]$ by whether $\sigma \models \mathcal{T}$ therefore gives $\mathbb{E}_\pi[U] \leq U^* \Pr_\pi(\sigma \models \mathcal{T})$. Dividing by $U^* > 0$ proves the bound; the equality is Lemma 1. □

Corollary 1 (Almost-sure satisfaction of return maximizers)**.** *Under the assumptions of Proposition 1, let transitions and termination be deterministic and the action set finite. If μ_0-almost every initial state admits a trajectory with $U > 0$, then every policy maximizing $\mathbb{E}_\pi[G]$ satisfies $\mathcal{T}$ almost surely.*

Proof. Each action sequence determines a trajectory from s_0, so the finite horizon and action set ensure that $U^*(s_0) = \max_\sigma U(\sigma)$ over them is attained. Choosing a maximizing sequence for each observed s_0 attains $\mathbb{E}_{\mu_0}[U^*(s_0)]$, the upper bound on $\mathbb{E}_\pi[U] = \mathbb{E}_{\mu_0}\big[\mathbb{E}_\pi[U \mid s_0]\big]$. Since $\mathbb{E}_\pi[U \mid s_0] \leq U^*(s_0)$, every maximizer of $\mathbb{E}_\pi[G] = \mathbb{E}_\pi[U] - \mathbb{E}_{\mu_0}[c(s_0)]$ has $\mathbb{E}_\pi[U \mid s_0] = U^*(s_0)$ for μ_0-almost every s_0. Applying Proposition 1 conditional on s_0, with $0 < U^*(s_0) < \infty$, gives $\Pr_\pi(\sigma \models \mathcal{T} \mid s_0) \geq 1$. A satisfying trajectory without an instantaneous violation has $S = 0$ and $U = \alpha \gamma_s^N \rho(\sigma, \mathcal{T}) > 0$. □

In stochastic settings, however, this only gives a loose bound as a maximizer of $\mathbb{E}_\pi[U]$ need not maximize the satisfaction probability, since a policy that satisfies with a large margin most of the time and otherwise leads to a violation, contributing a U close to 0, can outscore one that satisfies surely with a small margin.

3.3 Temporal Behavior Trees as Curriculum

Owing to their tree structure, we can extract a curriculum directly from a TBT without requiring additional inputs or representations. As a TBT is a tree, every subtree of a TBT is a TBT as well. We use this fact and construct a curriculum which iteratively builds up from the leaf nodes of the TBT toward the full specification. Using this, TBTCL trains a single policy throughout the curriculum, with progression through the curriculum corresponding to which subtree of the full TBT specification provides the robustness-based reward.

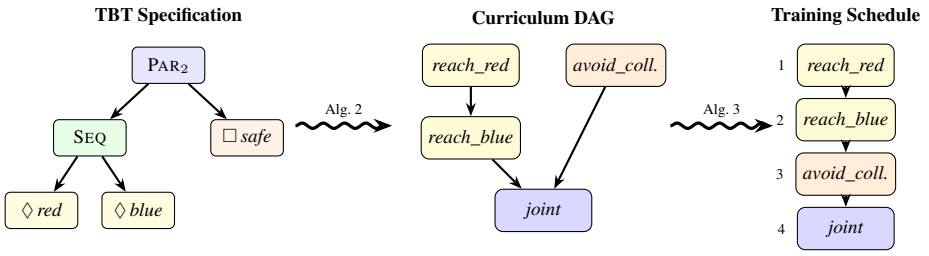


Figure 3: Going from a TBT specification to a curriculum DAG via Algorithm 2. Each leaf first becomes a stage, SEQ induces prerequisite edges and PAR_2 creates a joint stage gated on both children. Finally, topological sort yields the linear training schedule executed by Algorithm 3.

Curriculum Extraction The curriculum extraction begins with a walk-through of the TBT, defining the specification task and constructing a Curriculum Graph directly from the semantics of TBTs. We directly use the hierarchical structure of TBTs, traversing through the tree and building a curriculum based on the leaf nodes and the internal nodes connecting them.

First, we compute a Curriculum Graph in which each node encodes a stage (the subtrees specification being learned), and each edge encodes dependencies between stages (which subtask must be mastered before proceeding to the next stage). We convert a TBT into a Curriculum Graph by traversing it, with its semantics governing its encoding.

Each LEAF(φ) node is converted into a stage, as it corresponds to learning the behavior encoded in the STL formula, such as reaching a target or maintaining distance from a hazard. Sequence nodes SEQ($\mathcal{T}_1, \mathcal{T}_2$) impose order between stages and encode a dependency between stages such that $\mathcal{T}_1$ must be promoted before the stages extracted from $\mathcal{T}_2$ become available. Similarly, FBACK($\mathcal{T}_1, \ldots, \mathcal{T}_k$) encodes a dependency where a single stage becoming PROMOTED is sufficient to unlock the current stage.

PAR and KLEENE nodes encode *composition* stages as they control how rewards from multiple tasks interact. For PAR$_M(\mathcal{T}_1, \ldots, \mathcal{T}_k)$ nodes, every child is first trained as a subobjective, and a joint stage whose objective is the full parallel node becomes available once at least M child stages are promoted. For KLEENE($n, \mathcal{T}_c$) nodes, the stages of $\mathcal{T}_c$ are followed by one stage per repetition count $k = 2, \ldots, n{+}1$, each requiring k repetitions of $\mathcal{T}_c$ and unlocked by the promotion of its predecessor, until the full $n{+}1$ repetitions are reached.

We show a pseudo-code implementation in Algorithm 2.

Scheduling We convert the Curriculum Graph into a learning schedule by traversing the Curriculum Graph. The scheduler then moves through one stage at a time, tracking the status of each stage in the Curriculum Graph following the progression of LOCKED $\rightarrow$ READY $\rightarrow$ ACTIVE $\rightarrow$ PROMOTED.

Stages that have no prerequisites, such as those corresponding to LEAF nodes, start READY while all others start LOCKED. The TBT formula of the stage marked ACTIVE is used as the reward function as described in Section 3.2.

A stage becomes PROMOTED after the policy has successfully learned the corresponding subtask, meaning that over the sliding window of W episodes either the rolling mean robustness $\overline{\rho}_W$ or the success rate η_W is above the respective thresholds ($\overline{\rho}_W > \rho_{\text{thresh}}$ or $\eta_W > \eta_{\text{thresh}}$). We also enforce a minimum number of episodes per stage, E_{min}, before promotion becomes possible.

When all (or, in the case of PAR$_M$, at least M) the dependencies of a stage become PROMOTED , it is marked as READY and available for selection by the scheduler. The next stage to become ACTIVE is selected by a scheduler using an on-the-fly topological sort of the Curriculum Graph.

Once all curriculum stages are promoted, they effectively reconstruct the full TBT specification $\mathcal{T}$. TBTCL then enters a *consolidation phase* that trains on the full TBT specification, recovering any degraded subtask performance. This phase is further augmented by the *adaptive mode* we present in Section 3.4, which actively recovers policies' abilities on specific subtasks by directing the reward towards leaf nodes with the lowest robustness.

We can show that the curriculum that TBTCL builds is a finite, bounded-size DAG (which we discuss in Appendix D). Furthermore, we can show that the decomposition of the TBT into subtrees, as done during curriculum generation, aligns with the task described in the full TBT specification. While this guarantees neither optimality nor convergence, it shows that the tree structure of TBT can be used to derive a curriculum without requiring external mechanisms or difficulty rankings.

Proposition 2 (Semantic Curriculum Characterization)**.** *For any finite TBT specification $\mathcal{T}$ and finite trace σ, $\rho(\sigma, \mathcal{T}) > 0$ holds if and only if the operators of $\mathcal{T}$ can be resolved recursively into satisfied leaf-level STL obligations on the corresponding trace segments.*

Proof. We use structural induction over the TBT. Leaves directly inherit STL satisfaction and internal nodes follow TBT robustness semantics. Applying the induction hypothesis gives the corresponding satisfied leaf obligations in both directions. The full proof is in Appendix D.

□

3.4 Temporal Behavior Trees as Diagnostics

The same semantics of TBTs that make them effective for task specification and learning also enable us to better understand the trained policies. *Trace segmentation* [Schirmer et al., 2024] maps every

subtree of the TBT to the segment of a rollout on which it was evaluated and assigns that segment a robustness value, as shown in Figure 4. While Schirmer [2025] use segmentation to repair autonomous drone behavior [Schirmer et al., 2026], we use it to examine how a learned policy behaves and how it aligns with the specification described by the TBT.

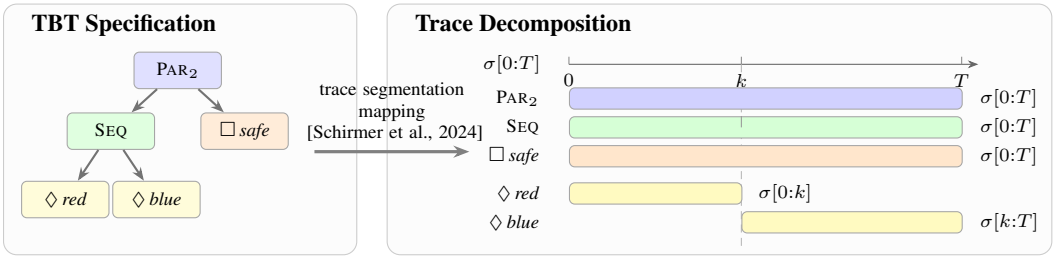


Figure 4: Trace decomposition induced by a TBT $\mathcal{T}$, mapping subtrees onto subtraces and providing a robustness score for each subtree

Aggregating this mapping across several episodes, we can construct a per-leaf competence profile that shows which part of the TBT specification was acted on, when, and how well the policy performed it. This analysis of learned policy behavior follows directly from the formal semantics of the TBTs and their trace segmentation, and provides a more robust analysis of policy actions.

Furthermore, we can use this technique not only for post-training diagnostics but also to guide training by identifying bottlenecks that occur during learning. By tracking the robustness scores of each leaf over a sliding window of sampled episodes, we can identify *bottleneck nodes* with the lowest robustness scores.

Algorithm 1: Adaptive Training

Input : trace σ, active stage s; window W_{stall}, budget R, threshold τ

Segment σ on s; record its bottleneck leaf
if *s was promoted* **then** reset stall; end redirect
else if *redirect budget R is spent* **then** resume s
else if *no promotion for $\geq W_{stall}$ episodes* **then**
  $b^* \leftarrow$ top non-safety bottleneck leaf
  if $\text{freq}(b^*) \geq \tau$ **then** redirect training to b^*

Using this information, during the *consolidation phase*, we can redirect the scheduler to these nodes, using their subspecification to derive the reward and reinforce the policy on this subtask. We call this mechanism *adaptive training* (Algorithm 1), which further mitigates catastrophic forgetting, often observed in compositional learning, by refocusing the agent on specific tasks that degrade during the final phase of learning.

4 Experiments

Setup We evaluate TBTCL on six specifications across two domains, spanning reachability, sequential composition, Kleene repetition, and reach-avoid tasks with safety constraints. We base our implementation on Stable-Baselines3 [Raffin et al., 2021] and PyTorch [Paszke et al., 2019], with NumPy [Harris et al., 2020] for the robustness and TBT engines, NetworkX [Hagberg et al., 2008] for graph operations, and Hydra [Yadan, 2019] for experiment configuration. We ran the evaluations on a single Dell R6525 (2x AMD Epyc 7773x - 128 Cores, 256 Threads, 2TB RAM).

LetterWorld is a discrete gridworld with lettered target cells; the agent observes its normalized position in $[0, 1]^2$ and selects from five actions. Manhattan-distance signals serve as inputs to STL predicates; collisions with obstacles terminate the episode. We use 5×5 grids for compositional tasks and a 10×10 grid with 14 obstacles for reach-avoid. *PointNav* is a continuous 2D point-mass domain in $[-5, 5]^2$ with continuous acceleration in $[-1, 1]^2$; waypoints and circular hazards provide Euclidean distance signals. All methods are trained using PPO [Schulman et al., 2017], 2M environment steps, 20 random seeds and are evaluated over 100 episodes with the stochastic policy.

We evaluate six specifications (Appendix F): single-goal reachability (REACH_RED, REACH_A), sequential composition (RED_THEN_BLUE), repetition (PATROL), and reach-avoid (REACH_AVOID). We compare TBTCL against five baselines: *STL Shaping* [Várnai and Dimarogonas, 2020] (dense robustness reward, no curriculum), *AGCL* [Shukla et al., 2023] (DFA-based curriculum, Boolean

reward), *SPECTRL* [Jothimurugan et al., 2019] (subpolicy decomposition, sparse reward), *Sparse* ($+1$ on task completion) and *LDBA* [Hahn et al., 2019] (using limit-deterministic Büchi automata to derive the reward).

Results We report the *specification satisfaction rate*: the fraction of episodes where the learned policy produces a trace with positive robustness ($\rho > 0$). A rate of 1.00 indicates that the policy reliably produces traces that satisfy the full TBT specification. Table 1 and Figure 5 report satisfaction rates on the six specifications we evaluate.

All methods can solve single-goal reachability in LetterWorld. The dense robustness reward drives the sequential and repetition results: TBTCL and STL Shaping both reach 1.00 on RED_THEN_BLUE and 0.99 and 1.00 on PATROL, whereas AGCL partially solves both (0.50 and 0.60), LDBA solves the sequence (0.95) but not the repetition (0.40), and SPECTRL and Sparse fail. The curriculum separates methods on reach-avoid tasks: in LetterWorld, TBTCL achieves 0.85 versus 0.74 for STL Shaping and 0.38 for Sparse; in PointNav, TBTCL reaches 0.64 while STL Shaping, AGCL, SPECTRL and Sparse remain at 0 and LDBA reaches 0.14. (Figure 5).

Table 1: Mean specification satisfaction rate (fraction of evaluation episodes with $\rho > 0$) over 20 seeds.

	LetterWorld				**PointNav**	
Method	REACH	SEQ	PATROL	AVOID	REACH	AVOID
TBTCL	**1.00** $\pm$ 0.00	**1.00**	0.99 $\pm$ 0.01	**0.85** $\pm$ **0.06**	**1.00**	**0.64** $\pm$ **0.09**
STL Shaping	**1.00** $\pm$ 0.00	**1.00**	**1.00**	0.74 $\pm$ 0.08	**1.00**	0.00 $\pm$ 0.00
AGCL	**1.00**	0.50 $\pm$ 0.07	0.60 $\pm$ 0.10	0.00	0.00	0.00
SPECTRL	**1.00**	0.00 $\pm$ 0.00	0.00	0.38 $\pm$ 0.11	0.24 $\pm$ 0.09	0.00
Sparse	**1.00**	0.02 $\pm$ 0.02	0.00	0.38 $\pm$ 0.11	0.24 $\pm$ 0.09	0.00
LDBA	**1.00**	0.95 $\pm$ 0.05	0.40 $\pm$ 0.11	0.45 $\pm$ 0.11	0.70 $\pm$ 0.11	0.14 $\pm$ 0.08

These results suggest that dense robustness derived from either STL formula or TBTs, and the curriculum mechanism are complementary. On tasks where the reward provides sufficient information, such as sequential composition and patrol, robustness shaping alone suffices, and the curriculum provides no additional benefit. However, when satisfying multiple subtasks simultaneously requires balancing multiple objectives such as both safety and goal completion, the curriculum becomes essential. It allows the policy to first master reachability subtasks before the safety constraint is introduced, avoiding the reward conflict that stalls all non-curriculum baselines that attempt the full specification at once. The complementary nature of the robustness rewards and the curriculum supports the claim that encoding both objects as a single TBT provides benefit at no additional cost. We provide ablations, failure analyses, and parameter-sensitivity analyses in Appendix E.

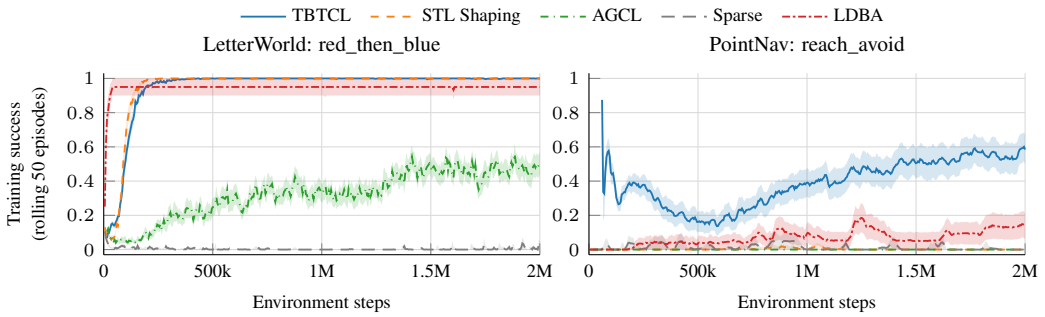


Figure 5: Training success on the full specification vs. environment steps (rolling mean over 50 episodes, mean $\pm$ SEM over 20 seeds)

5 Related Work

Temporal specifications as rewards and automata. STL robustness has been extensively used as a dense reward signal for reinforcement learning and control [Aksaray et al., 2016, Balakrishnan and Deshmukh, 2019, Várnai and Dimarogonas, 2020, Camacho et al., 2019]. Automata-based methods, such as Reward Machines [Icarte et al., 2018, 2022], LDBAs [Hahn et al., 2019] and AGCL [Shukla et al., 2023], expose task structure through automaton states and transitions. We build on this line of work but keep task semantics, reward, curriculum, and diagnostics in a single TBT rather than deriving separate objects or requiring additional inputs.

Compositional specification languages. SPECTRL [Jothimurugan et al., 2019] and DiRL [Jothimurugan et al., 2021] also use logical structure to decompose RL tasks, train subpolicies, and compose them at execution time. TBTCL instead trains a single policy through changing subformula rewards, avoiding execution-time policy composition. The TBT semantics also provide trace segmentation, offering direct evidence for policy diagnostics and adaptive consolidation.

Temporal Behavior Trees. Temporal Behavior Trees were introduced for robust monitoring and segmentation of cyber-physical traces [Schirmer et al., 2024], and later applied to the analysis of autonomous systems and trace repair [Schirmer, 2025, Schirmer et al., 2026]. Concurrent work by Puranic et al. [2026] uses TBTs to shape RL rewards from imperfect demonstrations. Our approach is complementary, as it does not require demonstrations or trace repair; instead, it uses the TBT directly to generate curriculum stages, training rewards, and diagnostics.

Hierarchical and curriculum RL. Hierarchical RL decomposes behavior into skills or options [Sutton et al., 1999, Dietterich, 2000, Nachum et al., 2018, Levy et al., 2019], while curriculum learning orders training problems to improve optimization [Bengio et al., 2009, Narvekar et al., 2020]. TBTCL is closer to curriculum learning: it decomposes the reward and training schedule, not the deployed policy, and obtains the hierarchy from the formal specification rather than from learned option discovery or a task generator [Florensa et al., 2017, Dennis et al., 2020, Jiang et al., 2021].

6 Limitations

We evaluate TBTCL on six specifications across two small domains, aiming to showcase the potential of this research direction and show that TBTCL matches dense STL shaping on reachability, sequence, and repetition tasks, and improves over baselines on reach-avoid tasks where curriculum and bottleneck redirection matter more. While this is sufficient to show that TBTs can drive rewards, curriculum extraction, and diagnostics within a single RL pipeline, future work should address its scalability, optimization, and adaptation to realistic applications. The current reward construction requires additional terms to handle safety properties and parallel operators, as well as introducing additional hyperparameters. While TBTs provide a dense reward signal, their effective usage requires additional tuning via these parameters. As noted by Schirmer et al. [2024], robustness and trace segmentation can become expensive for long traces and partition-searching operators such as SEQ and KLEENE. While our formal results on curriculum structure guarantee a well-formed schedule, they provide no guarantees of convergence or optimal curriculum ordering.

7 Conclusion

We presented TBTCL, a framework for using Temporal Behavior Trees as a unified object for specification-guided reinforcement learning. The central idea is that the same TBT should not only define the task but also provide the dense reward, the curriculum structure, and the diagnostic model for learned policies. By adapting TBT robustness to prefix-based rewards, extracting curriculum stages directly from the tree, and using trace segmentation for bottleneck analysis, TBTCL preserves the structure that is lost in existing methods, where specifications must be compiled into separate, dedicated objects for rewards, curricula, and diagnostic tools. TBTCL matches dense STL shaping on reachability, sequence, and repetition tasks, and improves over baselines on reach-avoid tasks, where curriculum and bottleneck redirection have greater impact. We show the potential of developing a flexible specification language not only for task description but also for derivation of several RL objects at once, providing several benefits at minimal cost.

Acknowledgments and Disclosure of Funding

This work was partially supported by the European Union with ERC Grant HYPER (No. 101055412). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. IEEE Computer Society, 1977.

Sebastian Schirmer, Jasdeep Singh, Emily Jensen, Johann C. Dauer, Bernd Finkbeiner, and Sriram Sankaranarayanan. Temporal behavior trees: Robustness and segmentation. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024, Hong Kong SAR, China, May 14-16, 2024*, 2024.

Sebastian Schirmer. *Specifying Monitors for Autonomous Cyber-Physical Systems*. PhD thesis, Saarland University, Saarbrücken, Germany, 2025. URL `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/41293`.

Sebastian Schirmer, Philipp Schitz, Johann C. Dauer, Bernd Finkbeiner, and Sriram Sankaranarayanan. Trace repair for temporal behavior trees. In *Tools and Algorithms for the Construction and Analysis of Systems - 32nd International Conference, TACAS 2026, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2026, Turin, Italy, April 11-16, 2026, Proceedings, Part I*, Lecture Notes in Computer Science, 2026.

Péter Várnai and Dimos V. Dimarogonas. On robustness metrics for learning STL tasks. In *2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020*. IEEE, 2020.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, Lecture Notes in Computer Science, 2004.

Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, Lecture Notes in Computer Science, 2010.

Pierre Wolper. Temporal logic can be more expressive. *Inf. Control.*, 56, 1983.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009. doi: 10.1016/J.TCS.2009.06.021. URL `https://doi.org/10.1016/j.tcs.2009.06.021`.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22:268:1–268:8, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 2019.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585, September 2020.

Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. Exploring network structure, dynamics, and function using networkx. Technical report, 01 2008. URL https://www.osti.gov/biblio/960616.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

Yash Shukla, Abhishek Kulkarni, Robert Wright, Alvaro Velasquez, and Jivko Sinapov. Automaton-guided curriculum generation for reinforcement learning agents. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, pages 605–613. AAAI Press, 2023. doi: 10.1609/ICAPS.V33I1.27242. URL https://doi.org/10.1609/icaps.v33i1.27242.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13021–13030, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/f5aa4bd09c07d8b2f65bad6c7cd3358f-Abstract.html.

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 395–412. Springer, 2019.

Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*. IEEE, 2016.

Anand Balakrishnan and Jyotirmoy V. Deshmukh. Structured reward shaping using signal temporal logic specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, 2019.

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Proceedings of Machine Learning Research. PMLR, 2018.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Intell. Res.*, 2022. URL https://doi.org/10.1613/jair.1.12440.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10026–10039, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/531db99cb00833bcd414459069dc7387-Abstract.html.

Aniruddh G Puranic, Sebastian Schirmer, John S Baras, and Calin Belta. Learning from imperfect demonstrations via temporal behavior tree-guided trajectory repair. *arXiv preprint arXiv:2604.04225*, 2026.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 1999.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13:227–303, 2000.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3307–3317, 2018.

Andrew Levy, George Dimitri Konidaris, Robert Platt Jr., and Kate Saenko. Learning multi-level hierarchies with hindsight. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. Association for Computing Machinery, 2009.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, 2020.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, Proceedings of Machine Learning Research, pages 482–495. PMLR, 2017.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob N. Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 2021.

A LLM Use Statement

LLMs (Claude Opus 4.6/4.7/Fable 5) and Grammarly were used for editing and formatting. Agentic Coding tools (Claude Code) were used to implement the codebase and baselines. Other LLM tools (namely Claude Opus 4.6, Gemini 3.1, and GPT-5.5) were used to solicit feedback on the paper draft, proofread the results presented in the paper, and conduct background research on the literature.

B Hyperparameters and Sensitivity

Parameter	Value
Algorithm	PPO [Schulman et al., 2017]
Policy	MLP (SB3 `MlpPolicy`)
Learning rate	3×10^{-4}
Rollout length	2048 steps
Minibatch size	256
Epochs per update	10
γ (PPO discount)	0.995
Entropy coefficient	0.01
Max gradient norm	0.5
Training steps	2×10^6
Seeds	20 (1–20)

Curriculum Parameter	Value
ρ_{thresh} (promotion robustness threshold)	0.05
η_{thresh} (promotion success rate threshold)	0.8
W (sliding window size)	80 episodes
E_{min} (minimum episodes before promotion)	50
α (reward scale)	5.0
β (safety penalty scale)	0.5
λ_{par} (PAR shaping scale)	0.5
γ_s (shaping discount)	1.0
PAR term gate	safety-free parallels only
Reward clipping	none
Kleene cycle window	100 steps
W_{stall} (adaptive stall window)	50 episodes
R (redirect budget, episodes)	30 episodes
Redirect frequency threshold τ	0.3

Table 2 shows how the performance on two reach-avoid tasks differs in sensitivity to the defaults above, varying one factor at a time (18 settings $\times$ 2 tasks $\times$ 20 seeds $=$ 720 runs). The discrete task is robust to most parameters, with only the reward scale α showing sensitivity (0.64 at $\alpha = 1$, 0.98 at $\alpha = 10$). However, the continuous domain depends on the safety scale ($\beta = 0.1$: 0.20; $\beta = 1.0$: 0.72), and on the promotion window (160 episodes: 0.42) the shaping discount ($\gamma_s = 0.995$, matching PPO's discount so that the estimated return is exactly G: 0.00) . The effects of the PAR$_M$ gate and the promotion threshold are limited in our evaluation. Similarly, for the stall window and the redirect budget the impact is overall small but helps with stability and avoiding stalls during learning of complex specifications.

Table 2: One-factor-at-a-time sensitivity of TBTCL (satisfaction rate, mean $\pm$ SEM over 20 seeds). Bold value = the setting used everywhere else in the paper.

Factor	Value	LW AVOID	PN AVOID
α (reward scale)	1.0	0.64 ± 0.08	0.52 ± 0.10
	2.5	0.84 ± 0.06	0.76 ± 0.07
	5.0	0.85 ± 0.06	0.64 ± 0.09
	10.0	0.98 ± 0.00	0.47 ± 0.07
β (safety scale)	0.1	0.85 ± 0.06	0.20 ± 0.07
	0.5	0.85 ± 0.06	0.64 ± 0.09
	1.0	0.85 ± 0.06	0.72 ± 0.08
PAR scale	0.1	0.85 ± 0.06	0.64 ± 0.09
	0.5	0.85 ± 0.06	0.64 ± 0.09
	1.0	0.85 ± 0.06	0.64 ± 0.09
γ_s (shaping discount)	0.995	0.76 ± 0.09	0.00
	1.0	0.85 ± 0.06	0.64 ± 0.09
PAR gate	none	0.89 ± 0.05	0.70 ± 0.09
	safety_free	0.85 ± 0.06	0.64 ± 0.09
promotion ρ threshold	0.01	0.91 ± 0.04	0.62 ± 0.10
	0.05	0.85 ± 0.06	0.64 ± 0.09
	0.2	0.90 ± 0.04	0.65 ± 0.09
promotion window	40	0.91 ± 0.04	0.52 ± 0.09
	80	0.85 ± 0.06	0.64 ± 0.09
	160	0.85 ± 0.06	0.42 ± 0.09
stall window	25	0.83 ± 0.07	0.68 ± 0.09
	50	0.85 ± 0.06	0.64 ± 0.09
	100	0.89 ± 0.04	0.56 ± 0.09
redirect budget	15	0.83 ± 0.06	0.54 ± 0.10
	30	0.85 ± 0.06	0.64 ± 0.09
	60	0.93 ± 0.04	0.59 ± 0.10

C STL and TBT definition

Definition 1 (STL Syntax)**.** *An STL formula is defined inductively as* $\varphi ::= \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{[l,u]}\varphi \mid \Box_{[l,u]}\varphi \mid \varphi_1 \,\mathcal{U}_{[l,u]}\, \varphi_2$*, where* $\mu : \mathbb{R}^d \to \mathbb{R}$ *is an atomic predicate over the state space and* $[l, u]$ *are time interval bounds.*

Definition 2 (STL Robustness)**.** *The quantitative robustness* $\rho(\sigma, \varphi, t) \in \mathbb{R}$ *of a formula* φ *on trace* σ *at time* t *follows the standard recursive definition [Fainekos and Pappas, 2009]:*

$$\rho(\sigma, \mu, t) = \mu(\sigma(t)) \tag{7}$$
$$\rho(\sigma, \neg\varphi, t) = -\rho(\sigma, \varphi, t) \tag{8}$$
$$\rho(\sigma, \varphi_1 \wedge \varphi_2, t) = \min\big(\rho(\sigma, \varphi_1, t),\ \rho(\sigma, \varphi_2, t)\big) \tag{9}$$
$$\rho(\sigma, \varphi_1 \vee \varphi_2, t) = \max\big(\rho(\sigma, \varphi_1, t),\ \rho(\sigma, \varphi_2, t)\big) \tag{10}$$
$$\rho(\sigma, \Diamond_{[l,u]}\varphi, t) = \max_{i \in [t+l,\, t+u]} \rho(\sigma, \varphi, i) \tag{11}$$
$$\rho(\sigma, \Box_{[l,u]}\varphi, t) = \min_{i \in [t+l,\, t+u]} \rho(\sigma, \varphi, i) \tag{12}$$
$$\rho(\sigma, \varphi_1 \,\mathcal{U}_{[l,u]}\, \varphi_2, t) = \max_{i \in [t+l,\, t+u]} \min\Big(\rho(\sigma, \varphi_2, i),\ \min_{j \in [t,\, i)} \rho(\sigma, \varphi_1, j)\Big) \tag{13}$$

When interval bounds $[l, u]$ *are omitted, the operators range over the full signal. Maxima and minima over empty index sets are defined as* $-\infty$ *and* $+\infty$ *respectively. Positive robustness of a trace implies Boolean satisfaction and negative robustness implies violation [Fainekos and Pappas, 2009].*

Definition 3 (TBT Syntax and Robustness)**.** *Following Schirmer et al. [2024] and adding explicit repetition, a TBT node* $\mathcal{T}$ *is defined inductively as* $\mathcal{T} ::= \textsc{Leaf}(\varphi) \mid \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2) \mid \textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k) \mid \textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k) \mid \textsc{Kleene}(n, \mathcal{T}_c)$*, where* M *is the* Par *quorum,* $\textsc{Kleene}(n, \mathcal{T}_c)$ *requires* $n{+}1$ *repetitions of* $\mathcal{T}_c$*. Robustness on sub-trace* $\sigma[l{:}u]$*:*

$$\rho(\sigma[l{:}u], \textsc{Leaf}(\varphi)) = \rho(\sigma[l{:}u], \varphi) \tag{14}$$
$$\rho(\sigma[l{:}u], \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2)) = \max_{i \in \{l, \ldots, u-1\}} \min\big(\rho(\sigma[l{:}i], \mathcal{T}_1),\ \rho(\sigma[i{+}1{:}u], \mathcal{T}_2)\big) \tag{15}$$
$$\rho(\sigma[l{:}u], \textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k)) = M\text{-th largest of } \{\rho(\sigma[l{:}u], \mathcal{T}_j)\}_{j=1}^{k} \tag{16}$$
$$\rho(\sigma[l{:}u], \textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k)) = \max_{i \in [l,u]} \max_{j \in [1,k]} \rho(\sigma[i{:}u], \mathcal{T}_j) \tag{17}$$
$$\rho(\sigma[l{:}u], \textsc{Kleene}(n, \mathcal{T}_c)) = \max_{(\sigma_1, \ldots, \sigma_{n+1}) \in P_{n+1}(l,u)} \min_{m=1}^{n+1} \rho(\sigma_m, \mathcal{T}_c) \tag{18}$$

Seq *partitions the trace at an optimal cut point;* Fback *selects the best child on the best suffix. The set* $P_k(l, u)$ *contains the valid ordered partitions of* $\sigma[l{:}u]$ *into* k *non-empty contiguous segments, and* $\max \emptyset = -\infty$ *when a segment is too short to be partitioned.*

For a more detailed explanation of TBTs, their semantics, satisfaction relation, and their use in runtime monitoring, we refer the reader to the foundational work by Schirmer et al. [2024].

D Curriculum Properties

Proof of Proposition 2 (Semantic Curriculum Characterization). We prove the stronger segment-level statement used by structural induction. To avoid side conditions for trace segments that admit no valid cut or partition, we use the standard convention $\max \emptyset = -\infty$. Define $\mathrm{Sat}(\sigma[l{:}u], \mathcal{T})$ to mean that the recursive satisfaction conditions induced by $\mathcal{T}$ hold on segment $\sigma[l{:}u]$:

$$\begin{aligned}
\mathrm{Sat}(\sigma[l{:}u], \textsc{Leaf}(\varphi)) &\iff \rho(\sigma[l{:}u], \varphi) > 0, \\
\mathrm{Sat}(\sigma[l{:}u], \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2)) &\iff \exists i \in \{l, \ldots, u-1\} : \mathrm{Sat}(\sigma[l{:}i], \mathcal{T}_1) \wedge \mathrm{Sat}(\sigma[i{+}1{:}u], \mathcal{T}_2), \\
\mathrm{Sat}(\sigma[l{:}u], \textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k)) &\iff \exists Q \subseteq \{1, \ldots, k\},\ |Q| = M : \forall j \in Q,\ \mathrm{Sat}(\sigma[l{:}u], \mathcal{T}_j), \\
\mathrm{Sat}(\sigma[l{:}u], \textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k)) &\iff \exists j \in \{1, \ldots, k\}, \exists i \in \{l, \ldots, u\} : \mathrm{Sat}(\sigma[i{:}u], \mathcal{T}_j), \\
\mathrm{Sat}(\sigma[l{:}u], \textsc{Kleene}(n, \mathcal{T}_c)) &\iff \exists (\sigma_1, \ldots, \sigma_{n+1}) \in P_{n+1}(l, u) : \forall m,\ \mathrm{Sat}(\sigma_m, \mathcal{T}_c).
\end{aligned}$$

We prove by structural induction on $\mathcal{T}$ that

$$\mathrm{Sat}(\sigma[l{:}u], \mathcal{T}) \iff \rho(\sigma[l{:}u], \mathcal{T}) > 0.$$

Leaf. Equation 14 gives

$$\rho(\sigma[l{:}u], \textsc{Leaf}(\varphi)) = \rho(\sigma[l{:}u], \varphi),$$

so the equivalence holds by the definition of Sat for leaves.

Sequence. Assume the claim holds for $\mathcal{T}_1$ and $\mathcal{T}_2$. By Eq. 15,

$$\rho(\sigma[l{:}u], \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2)) = \max_{i \in \{l,\dots,u-1\}} \min(\rho(\sigma[l{:}i], \mathcal{T}_1), \rho(\sigma[i{+}1{:}u], \mathcal{T}_2)).$$

This value is positive if and only if some cut i makes the displayed minimum positive, which holds iff both child robustness values at that cut are positive. By the induction hypothesis, this is equivalent to $\mathrm{Sat}(\sigma[l{:}i], \mathcal{T}_1)$ and $\mathrm{Sat}(\sigma[i{+}1{:}u], \mathcal{T}_2)$ for some valid cut, which is exactly Sat for the sequence.

Parallel. Assume the claim holds for all children. Equation 16 says that robustness is the M-th largest child robustness value. The M-th largest value is positive if and only if at least M children have positive robustness on the shared segment, equivalently, if and only if there exists a size-M quorum of positive children. By the induction hypothesis, this is exactly the existence of a size-M quorum whose children satisfy Sat on the shared segment.

Fallback. Assume the claim holds for all children. By Eq. 17,

$$\rho(\sigma[l{:}u], \textsc{Fback}(\mathcal{T}_1, \dots, \mathcal{T}_k)) = \max_{i \in [l,u]} \max_{j \in [1,k]} \rho(\sigma[i{:}u], \mathcal{T}_j).$$

This maximum is positive if and only if some child is positive on some suffix. By the induction hypothesis, this is equivalent to $\mathrm{Sat}(\sigma[i{:}u], \mathcal{T}_j)$ for some child j and suffix start i, which is exactly Sat for fallback.

Kleene. Assume the claim holds for $\mathcal{T}_c$. By Eq. 18,

$$\rho(\sigma[l{:}u], \textsc{Kleene}(n, \mathcal{T}_c)) = \max_{(\sigma_1,\dots,\sigma_{n+1}) \in P_{n+1}(l,u)} \min_{m=1}^{n+1} \rho(\sigma_m, \mathcal{T}_c).$$

This value is positive if and only if some valid partition has a positive minimum, which holds if and only if every repetition segment in that partition gives positive robustness for the child. By the induction hypothesis, this is equivalent to every segment in that partition satisfying $\mathrm{Sat}(\sigma_m, \mathcal{T}_c)$, which is exactly Sat for KLEENE nodes. □

Proposition 3 (Curriculum Well-Formedness)**.** *For any finite TBT $\mathcal{T}$, Algorithm 2 returns a finite acyclic curriculum graph $D = (V_D, E_D)$, with vertex set V_D of at most $|V_D| = |Leaf| + |\textsc{Par}| + |\textsc{Fback}| + \sum_{\textsc{Kleene}(n,\cdot) \in \mathcal{T}} n$ nodes and at most $|E_D| \le |V_D|^2$ edges.*

Proof of Proposition 3 (Curriculum Well-Formedness). As EXTRACT visits every node of the finite tree, it leads to a finite curriculum graph. During its traversal, it creates one stage per leaf, one joint stage per $\textsc{Par}_M$ node, one gate stage per FBACK node, none for a SEQ node, and the n repetition stages $r_2, \dots, r_{n+1}$ per $\textsc{Kleene}(n, \cdot)$ node. By construction, this gives the vertex count.

For acyclicity, we observe that every edge runs from a stage that already exists to one created later. SEQ connects the latest stage of $\mathcal{T}_1$ to the sources of $\mathcal{T}_2$, which are created only after $\mathcal{T}_1$ has been processed, while $\textsc{Par}_M$, FBACK and KLEENE connect existing frontiers to the stage created in the same step. By construction, at each step, newly created edges can only point from existing stages to stages created in that step. As they can never encode a dependency in the opposite direction, no directed cycle can exist. The edge bound follows trivially from $E_D \subseteq V_D \times V_D$. □

E Ablations and Failure Modes

We take a closer look at the components of TBTCL and remove one at a time on the two reach-avoid tasks: the safety term of the reward (Section 3.2), the curriculum (Section 3.3) and the adaptive training (Section 3.4). The results in Table 3 show that the reward terms and the curriculum are complementary. In PointNav the curriculum without the safety term does not learn at all (0.00), and

the full reward without a curriculum reaches 0.37, both significantly below the full method (0.64). In LetterWorld the variant without the safety term is identical to the full method over all 20 seeds: a collision ends the episode before the safety leaf is ever violated, so the term never fires and this column measures the curriculum alone, which lifts the satisfaction rate from 0.74 to 0.85.

We apply the trace segmentation mechanism from Section 3.4 to the 100 evaluation episodes of every TBTCL seed on PointNav reach-avoid to analyze how outcomes and bottlenecks vary across seeds. We find that the overall outcome distribution is bimodal: 8 of the 20 seeds satisfy the specification in at least 90% of episodes, 2 never do, and the remaining 10 lie in between. The per-leaf competence profile shows that the safety leaf *avoid_hazard* is the most common bottleneck for all 12 seeds, and the reach leaf *reach_a* contributes to issues for 6 seeds.

Table 3: Reward-component ablation of TBTCL on the two reach-avoid specifications (satisfaction rate, mean $\pm$ SEM over 20 seeds).

Variant	LW AVOID	PN AVOID
TBTCL (full)	0.85 ± 0.06	0.64 ± 0.09
no safety term (r^{Δ} only; PAR term gated off)	0.85 ± 0.06	0.00
shaping only (full reward, no curriculum)	0.74 ± 0.08	0.37 ± 0.10
no adaptive redirect	0.86 ± 0.05	0.60 ± 0.10

F Specification Definitions

Table 4 lists the six specifications evaluated in the main experiments.

Table 4: Evaluated task specifications. *LW*: LetterWorld (discrete); *PN*: PointNav (continuous).

Spec	**TBT Formula**	**Env**	**Grid / Arena**	H
REACH_RED	$\textsc{Leaf}(\Diamond\,(d_red < 1.0))$	LW	5×5	50
RED_THEN_BLUE	$\textsc{Seq}(\Diamond\, red,\ \Diamond\, blue)$	LW	5×5	100
PATROL	$\textsc{Kleene}(2, \textsc{Seq}(\Diamond\, red,\ \Diamond\, blue))$	LW	5×5	200
REACH_AVOID	$\textsc{Par}_2(\Diamond\, red,\ \Box\,(d_obs > 0.5))$	LW	10×10 + obs	200
REACH_A	$\textsc{Leaf}(\Diamond\,(d_a < 0.5))$	PN	5.0	200
REACH_AVOID	$\textsc{Par}_2(\Diamond\, a,\ \Box\,(d_hz > 0.5))$	PN	5.0 + hazards	300

G Algorithm Details

We show the pseudocode implementation of the curriculum extraction and curriculum scheduler algorithms presented in Section 3.3.

Algorithm 2: Curriculum DAG Extraction

Input : TBT $\mathcal{T}$
Output : Curriculum DAG $D = (V_D, E_D)$, its source stages A and its frontier stages F

Initialize empty graph $D = (V_D, E_D)$;
Function `Extract(`$\mathcal{T}$`):`
  if $\mathcal{T} = \textsc{Leaf}(\varphi)$ **then** `// single leaf stage, Globally leaves included`
    Create stage v with formula φ; add v to V_D;
    return $(\{v\}, \{v\})$;
  else if $\mathcal{T} = \textsc{Seq}(\mathcal{T}_1, \mathcal{T}_2)$ **then** `// prefix before suffix`
    $(A_1, F_1) \leftarrow$ `Extract(`$\mathcal{T}_1$`)`; $(A_2, F_2) \leftarrow$ `Extract(`$\mathcal{T}_2$`)`;
    Add edges $f \to v$ for all $f \in F_1$, $v \in A_2$; `//` $\mathcal{T}_2$`'s sources wait for` $\mathcal{T}_1$
    return (A_1, F_2);
  else if $\mathcal{T} = \textsc{Par}_M(\mathcal{T}_1, \ldots, \mathcal{T}_k)$ **then** `// parts then compose`
    $(A_i, F_i) \leftarrow$ `Extract(`$\mathcal{T}_i$`)` for each child $\mathcal{T}_i$;
    Create joint stage j with formula $\mathcal{T}$ and quorum M; add j to V_D;
    Add edges $f \to j$ for all $f \in \bigcup_i F_i$;
    return $(\bigcup_i A_i, \{j\})$;
  else if $\mathcal{T} = \textsc{Kleene}(n, \mathcal{T}_c)$ **then** `// one lap, then` $2, \ldots, n{+}1$
    $(A_c, F_c) \leftarrow$ `Extract(`$\mathcal{T}_c$`)`; $P \leftarrow F_c$;
    for $k \leftarrow 2$ **to** $n{+}1$ **do**
      Create repetition stage r_k with formula "k repetitions of $\mathcal{T}_c$" (r_{n+1} carries $\mathcal{T}$ itself); add r_k to V_D;
      Add edges $p \to r_k$ for all $p \in P$; $P \leftarrow \{r_k\}$;
    return (A_c, P);
  else if $\mathcal{T} = \textsc{Fback}(\mathcal{T}_1, \ldots, \mathcal{T}_k)$ **then** `// any child suffices`
    $(A_i, F_i) \leftarrow$ `Extract(`$\mathcal{T}_i$`)` for each child;
    Create gate stage g with formula $\mathcal{T}$ and quorum 1; add g to V_D;
    Add edges $f \to g$ for all $f \in \bigcup_i F_i$;
    return $(\bigcup_i A_i, \{g\})$;
$(A, F) \leftarrow$ `Extract(`$\mathcal{T}$`)`; **return** (D, A, F);

Algorithm 3: Bottom-Up Curriculum Scheduler

Input : TBT $\mathcal{T}$, curriculum DAG $D = (V_D, E_D)$ with stage formulas X_v and quorums q_v , ρ_{thresh}, η_{thresh}, W, E_{min}
Initialize: source stages $\leftarrow$ READY; all others $\leftarrow$ LOCKED;
while *training budget not exhausted* **do**
  $v \leftarrow$ the ACTIVE stage, else the first READY stage in topological order, set to ACTIVE;
  if *no such* v **then** Train one episode on the reward of $\mathcal{T}$; **continue**; `// consolidation`
  Train one episode on the reward of X_v; record $\rho(\sigma, X_v)$ for the episode trace σ;
  $\overline{\rho}_W \leftarrow$ mean of ρ, $\eta_W \leftarrow$ fraction with $\rho > 0$, over the last W episodes recorded for v;
  if $\text{episodes}(v) \geq E_{min}$ ***and*** $(\overline{\rho}_W > \rho_{thresh}$ ***or*** $\eta_W \geq \eta_{thresh})$ **then** `Promote(`v`)`;
Procedure `Promote(`v`):`
  v.status $\leftarrow$ PROMOTED;
  foreach *dependent* d *of* v *in* D *that is* LOCKED *or* READY **do**
    if *at least* q_d *prerequisites of* d *are* PROMOTED **then** `// quorum met`
      if d *is a* FBACK *gate* **then** Mark the other prerequisites of d SKIPPED; `Promote(`d`)`;
      else Mark the other prerequisites of d PROMOTED; d.status $\leftarrow$ READY;

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We present our contributions in Section 3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in a dedicated section at the end of the paper.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We supply the neccesary information in the Appendix D

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the neccessary details in Section 4 and Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The method is described in enough detail to support replication

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all necessary information in Sections 4 and Appendix B

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report only basic variance measure

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We list the computational resources used in Section 4

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and believe we are compliant to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: We do not see any potentially significant societal impact of our work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work poses no such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We list and cite the relevant software packages that we build our implementation on.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We described our data generation procedure in detail in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No human subjects were involved

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Detailed answer in Appendix A

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.