

Specification-Driven Planning for Safe Autonomy

Parv Kapoor

CMU-S3D-26-111

June 2026

Software and Societal Systems Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Eunsuk Kang (Chair)

Sebastian Scherer

Changliu Liu

Karen Leung (University of Washington)

*Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in Software Engineering.*

Copyright © 2026 Parv Kapoor

This work was supported in part by the National Science Foundation Award CCF-2144860 and CyLab Seed Funding 63235.1.1013066.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution or any other entity.

Keywords: Specification-driven planning, Formal Methods for Robotics, Provable Safety

Abstract

AI-driven planning has greatly expanded the scope of autonomy, transforming how robots perceive, reason, and act in complex environments. Foundation models trained on Internet-scale multimodal data have been a key driver behind this boost in autonomy. However, deploying existing models and techniques in safety-critical applications remains challenging due to the lack of a structured methodology to enforce strict behavioral and safety requirements. These constraints are essential for reliable operation in dynamic, real-world settings.

Historically, formal specifications, particularly linear temporal logic (LTL) and signal temporal logic (STL), have been used to encode safety and mission rules for robotics. Although recent work has integrated such logics into learning-based frameworks, they are poorly aligned with modern pre-training-based paradigms that underpin robot foundation models. Several theoretical and practical challenges must be overcome before these methods can be widely adopted.

My thesis work explores the integration of temporal logic specifications for modern AI-driven planning. It introduces methods to (1) decompose and efficiently evaluate complex STL specifications using modern auto-differentiation frameworks, (2) generalize specification design via Embedding Temporal Logic (ETL), which defines predicates over learned representations instead of explicit states, and (3) enforce STL specifications at inference time using SafeDec, a constrained decoding approach inspired by structured generation in large language models. Together, these contributions enable formal specification-driven planning in AI systems, ensuring safety in robotics and autonomous systems.

For my grandma

Acknowledgments

In my five years in Pittsburgh, I had the privilege to meet some amazing, kind and authentic people who enriched my life in so many ways. I have learned and grown so much due to these people and I will be indebted to them for the rest of my life. My words in this section will not do them justice, but I will try my best.

First, I would like to thank my advisors Eunsuk Kang and Sebastian Scherer. Eunsuk took a chance on me when I started my PhD with one unpublished paper and had immense impostor syndrome. Thank you for teaching me everything I know about research, including writing great papers, identifying research directions, giving amazing presentations, and a lot of other skills that I cannot possibly list. Most of all, I want to thank you for being patient with me as I gallivanted from one research direction to another and believing in me every step of the way. Our weekly one-on-one meetings will be dearly missed. I would also like to thank Sebastian for making me a part of AirLab and imparting me with all his field robotics knowledge. I will forever be in awe of you handling a huge lab and associated projects while still deeply engaging with each project to give concrete feedback on each and every aspect. Your cheerful optimism and support while I tried to make my theory work in the field will never be forgotten.

I would also like to thank my committee members Changliu Liu and Karen Leung. I was lucky enough to collaborate with both of them on multiple projects and I hope I can embody their excellence and vision in my life. Their mentorship has made me a better researcher but most of all a better human being. I would also like to thank other CMU mentors: Andrea Bajcsy, Yonatan Bisk, Henny Admoni, Fei Fang and Ding Zhao to name a few. I had the privilege to collaborate with them and learnt so much in their shadow. I would also like to thank my Microsoft Research and General Robotics internship mentors Siddhartha Sen, John Langford, Ashish Kapoor, Jonathan Huang and Sai Vemprala. Thank you for providing me with the opportunity and teaching me how to apply academic research to a much larger scale than I ever imagined.

I would also like to thank my SoDA and AirLab members: Andy Hammer, Changjian Zhang, Yining She, Ian Dardik, Manuel Barros, Ben-hau Chia, Yiliang Liang, Ian Higgins, Jay Patrikar, Nikhil Keetha, Brady Moon, Zelin Ye and Cherie Ho. I am grateful for all your feedback, collaboration efforts and all the fun we had working on varied projects. I would like to thank my friends in ping pong and life: Vasu Vikram, Kyle Liang, Tejus Bana, Jatan Loya, Steven Wu and Steven's dad. My PhD was probably a bit longer due to our long sessions but I would not trade a single second of it for anything. I would like to thank my gym buddies: Clara Na, Jimin Mun, Amanda Bertsch, Simran Khanuja, and Sophia Roshal for helping me maintain my mental sanity. I would like to thank my CMU Portugal friends: Maria Casimiro, Claudia Mamede,

Teresa Fortes, Catarina Gamboa, Paulo Santos, Hugo Simao, Luis Gomes and Rui Melo for providing me with a special place in their home and in their heart. Thank you Maria and Claudia for always being my agony aunts and listening to me yap for hours. A special thank you to Teresa Fortes for being the supportive and caring person she is, my only regret is that I met you too late in my PhD but I am so grateful for all our time together. Obrigado por tudo. I would like to thank my fellow CMU S3D students: Jenny Liang, Andrea Miller, Shrey Tiwari, Manisha Mukherjee, Angel Cui, Anmol Singhal, Megan Li, Wendy Bickersteth, Chenyang Yang, Courtney Miller, Samantha Phillips, Elizabeth Gilbert, Ian McCormack, Kush Jain, Matthew Davis, Nadia Nahar, Sam Estep, Nikitha Rao, Hobin Kim, and Sarah Santos. My years in this department were so beautiful because of you all. I would also like to thank my MLD friends: Sam Sokota, Jennifer Hsia, and Stephanie Milani for always being down to clown when nobody else was. I would like to thank my best friends: Cooper Lorsung, Nari Johnson, and Mohammad Qadri. All of you have a really special place in my life and I cannot even write in words what you mean to me. My love for you knows no bounds. A special thanks to my friends outside of CMU: Kunal Pratap Singh, Tanya Rodrigues, Mansi Ugale, Arusha Ellias and Shreeya Konnur. You kept me in touch with my roots and for that I am forever grateful. Finally, I would like to thank my family for their love and support. I lost my dad and grandpa at a young age but they instilled the courage and spirit in me that helps me brave this life every day. Thank you papa and dada. Thank you to my sister and aunt for believing in me and adding so much joy to my life. In the last year of my PhD, I had the misfortune of losing my grandma and my uncle but I would not be the person I am without them raising me with their values and all their heart. Thank you for teaching me the meaning of unconditional love and I hope you are proud of me.

Contents

1	Introduction	1
1.1	Challenges	2
1.2	Thesis Statement	4
1.3	Summary of Contributions	4
1.4	Outline	5
2	Background and Related Work	6
2.1	Temporal Logics for Robotics	6
2.2	Signal Temporal Logic	6
2.3	Planning from STL specifications	7
2.4	STL Evaluation and Monitoring Tools	8
2.5	Runtime Safety Enforcement for Robotics	9
2.6	Connections to safe reinforcement learning and control	9
3	STL Decomposition for Efficient Planning	10
3.1	Motivation	11
3.2	STLInc	12
3.2.1	Flattening	14
3.2.2	Symbolic Time Resolution	16
3.2.3	Scheduling	19
3.3	Evaluation	22
3.3.1	Experimental Setup	22
3.3.2	Results	23
3.4	Summary	24
4	STLCG++: Efficient STL Evaluation	25
4.1	Preliminaries	26
4.1.1	Signals and trajectories	26
4.1.2	Smooth max / min approximation	27
4.1.3	Robustness trace	27
4.1.4	Recursive structure of STL formulas	28
4.1.5	STLCG: Recurrent computation	28
4.2	Recurrent v/s Masking based Approach	29

4.2.1	Notation and masking operation	29
4.2.2	Eventually and Always operations	29
4.2.3	Until operation	31
4.2.4	Dependence on smooth approximations	32
4.2.5	Practical benefits	32
4.3	Evaluation	33
4.3.1	Computational Properties of STL _{CG} ++	33
4.4	Smoothing time intervals	34
4.5	Robotics-related applications	35
4.5.1	Trajectory planning with suboptimal STL specifications.	35
4.5.2	Deep generative modeling: STL-guided diffusion policies	37
4.5.3	Machine learning: STL parameter mining from data	37
4.6	Summary	38
5	ETL: Extending STL Beyond State-Based Constraints	39
5.1	Introduction	39
5.2	Background	41
5.2.1	Pretrained Vision Encoders	41
5.2.2	Specification based Runtime Monitoring	41
5.2.3	Conformal Prediction	41
5.3	Embedding Temporal Logic	42
5.3.1	Syntax and Semantics	42
5.3.2	ETL Specification Monitors	45
5.4	Constructing ETL specifications in Practice	47
5.4.1	Specifying Target Embeddings	47
5.4.2	Choice of Encoders and Distance functions	47
5.5	Threshold Calibration for ETL predicates	48
5.5.1	F1-Optimal Threshold	48
5.5.2	Conformal Threshold with Recall Guarantee	49
5.6	Evaluation	50
5.6.1	Simple Navigation Dubins Car	51
5.6.2	Simulated Manipulation Tasks	53
5.6.3	Real World Evaluation via DROID	53
5.7	Summary	54
6	Constrained Decoding for Foundation Models	56
6.1	Background: Constrained Decoding in Transformers	57
6.2	Specification-Guided Constrained Decoding	57
6.2.1	Problem Statement	59
6.2.2	Hard Constrained Decoding	60
6.2.3	Robustness Constrained Decoding	61
6.3	Evaluation	62
6.3.1	Implementational Details	62

6.3.2	Experimental setup	63
6.3.3	Results	64
6.3.4	Comparison with safe reinforcement learning baseline	66
6.3.5	Ablation studies	66
6.4	Summary	67
7	Future Work	68
8	Conclusion	70
A		73
A.1	Additional Evaluation Details for Dubins Car	73
A.1.1	Details for Dubins Car Setup	73
A.1.2	Results for Dubins Car	73
A.2	Additional Details for Manipulation Tasks	74
A.2.1	Tabularized Results	74
A.2.2	Sequential Predicate Trace	74
A.3	Additional DROID Evaluation Details	74
A.3.1	Tabularized Results	74
A.3.2	Ground Truth Construction	75
A.3.3	Encoder and Representation	75
A.3.4	Sequential Evaluation Details	76
A.3.5	Discussion of Failure Modes	77
A.4	SafeDec Visualizations	77
A.5	Results for ϕ_{geofence}	77

List of Figures

3.1	Left: An STL specification ϕ with multiple nested temporal operators and a possible decomposition into subtasks. Right: A sample trajectory that satisfies ϕ in a planar environment.	11
3.2	Overview of the STLINC approach	13
3.3	Flattening rules	15
3.4	An AST for an example STL specification, $F_{[1,20]}(G_{[1,5]}(r_1) \wedge F_{[6,15]}(r_2)) \wedge G_{[1,35]}(\neg r_3)$, with outputs from the flattening and symbolic time resolution steps. Each constraint resulting from the resolution step is assigned an identifier (<i>id</i>); a symbolic time variable that represents the time of satisfying the constraint is annotated with <i>id</i> as the subscript (i.e., t_2 represents the satisfaction of reachability constraint $(2, 21, r_1)$).	18
3.5	A slicing example.	20
4.1	We propose STLGC++ a masking approach to evaluating and backpropagating through signal temporal logic (STL) robustness formulas. The masking approach offers stronger computational, theoretical, and practical benefits compared to STLGC, a recurrent approach.	26
4.2	Illustration of the graph structure of an example STL formula. Input trajectories are first passed through the predicates and then each subformula according to the formula structure.	28
4.3	Comparison of computation time using PyTorch for masking (M, blue) and recurrent (R, orange) approach on CPU (top row) and GPU (bottom row) as signal length increases, across six STL formulas. Solid lines denote robustness computation; dashed lines denote gradient evaluation.	34
4.4	Smooth mask for various values of smoothing parameter c	35
4.5	Trajectory with optimized time interval in orange target region.	36
4.6	Generated trajectories using STL-guided DDIM.	37
4.7	The masking approach enables a smooth approximation of time intervals, enabling the use of gradient descent over time interval parameters. The contour plots show the loss landscape given by (4.4) as a function of a, b , the normalized time interval limits. The solutions at gradient descent steps 0, 500, 1000, and 5000 are shown. Note that the smooth time mask parameter and approximate max / min temperature values are annealed over the steps.	38

5.1	Overview of embedding-based runtime monitoring. Top: Target observations $O_G = \{o_{g1}, \dots, o_{gn}\}$ and the online observation o_t are encoded by a pretrained vision encoder into target embeddings z_G and the current embedding z_t . An embedding predicate then evaluates whether z_t reaches the target set. Bottom left: The embedding trajectory is projected onto its first two principal components, showing its evolution relative to the target embeddings. Bottom right: Runtime monitoring computes the distance from z_t to z_G over time, thresholds it using ϵ , and evaluates the temporal logic specification.	42
5.2	Qualitative and quantitative ETL results.	52
6.1	Overview of our specification aligned decoding framework. Given multimodal observations (RGB images and language goals), a pretrained transformer-based navigation policy generates candidate actions. Our constrained decoder then filters or reweights these actions using robustness scores from a user-defined STL safety specification. Left: Original action probabilities (red) from the pretrained policy are modified by SafeDec using STL robustness: actions predicted to violate the safety specification receive reduced weights, while safer actions are boosted. The chosen safe action and adjusted probabilities (green) are highlighted. Right: An example navigation episode where the robot starts in the kitchen (white dot) and moves toward the bedroom to locate the target object (green marker middle-right), while avoiding user-defined hazardous regions (red zones) throughout the trajectory.	58
6.2	SafeDec trade-off between STL satisfaction and task success. Average STL satisfaction versus task success rate aggregated across all specifications and three policies. RCD achieves the strongest trade-off, substantially improving STL satisfaction while remaining close to the unconstrained policy’s task performance. HCD and filtering achieve near-perfect STL satisfaction, but incur a larger reduction in task success.	64
6.3	Top-down visualization of trajectories for a sample scene under the instruction “ <i>find the sofa</i> ”. The unconstrained model passes through two forbidden regions (red squares) on its way to the sofa (center-left). In contrast, HCD (left) and RCD (right) modify the trajectories to respect STL safety specifications while still reaching the goal. More visualizations are available in Appendix A.4.	65
6.4	Ablation studies. (a) STL satisfaction (%) for HCD and RCD under baseline vs noisy dynamics across base models. (b) Effect of β on success rate and safety satisfaction.	67
A.1	Dual-predicate Boolean timelines for <code>mw-pick-place-wall</code> . Top panels show distances to grasp (z_A) and place (z_B) embeddings with thresholds ϵ^* and ϵ_{CP} . Lower panels show ETL and ground-truth predicate activations.	76
A.2	Top-down views across indoor environments with SafeDec induced trajectories. Red circles mark target objects, green boxes are avoid regions, and orange paths show safe trajectories under constrained decoding.	78

List of Tables

2.1	Summary of existing python-based STL toolboxes that are publicly available. . . .	8
3.1	Benchmark STL specifications created from motion planning patterns. Here, R: Reach, A: Avoid, SV: Sequenced Visit, SB: Stabilization.	22
3.2	STLINC Performance Benchmarking for LinEnv and NonLinEnv against standard MICP [18] and reduced MICP (R-MICP) [79].	24
4.1	Summary of STL specifications used for evaluation inspired by existing literature. I denotes the time interval over which the temporal operators apply.	33
4.2	Relative computation time of masking approach compared to recurrent approach. Median value across different signal lengths. Lower is better.	34
6.1	STL specifications used in our evaluation. Each region R_i denotes a spatial predicate over (x, z)	61
6.2	Comparison by decoding technique across models (SPOC, Flare, PoliFormer) for specifications ϕ_{avoid} , $\phi_{ordered}$, and ϕ_{no_return} . Each cell reports STL satisfaction / success rate (%). Higher is better ($\uparrow$).	63
6.3	Comparison with SafeVLA on the SafetyChores benchmark. Lower cost indicates fewer safety violations, while higher success denotes better task completion. . . .	66
A.1	ETL predicate evaluation on the Dubins car.	74
A.2	ETL predicate evaluation on MetaWorld (test split). Frame-level metrics are reported for all predicates.	74
A.3	F1 score for failure / predicate detection across three environments. All methods use the F1-optimal threshold; ETL (CP) additionally reports the class-conditional split-CP threshold. Per-column best in bold	75
A.4	Comparison of ETL and Qwen2-VL-2B on phasic DROID episodes.	75
A.5	Sequential predicate evaluation on DROID (SVD VAE, wrist camera).	77
A.6	Performance comparison for the geofence specification $\phi_{geofence}$ across decoding techniques and base models. Each cell reports STL satisfaction / success rate (%). Higher is better ($\uparrow$).	78

Chapter 1

Introduction

Artificial intelligence (AI)-driven planning has revolutionized robotics and autonomous decision-making in the past decade. A large part of this revolution has been due to advances in foundation models and world models that have significantly expanded the scope of autonomy. A wide variety of robot form factors can now perform increasingly complex tasks across diverse environments due to these foundation models [40]. These models trained on Internet-scale data pose superior generalization and semantic reasoning abilities, and these large models are increasingly being integrated into all parts of the robotic stack for perception, planning, and control.

Despite these successes, ensuring adherence to strict behavioral and safety requirements for large data-driven models remains an open problem. Although these neural models can adeptly process perception input and follow natural language commands to generate robot commands, they lack explicit mechanisms to enforce logical constraints. These logical constraints are crucial for the deployment of these models in safety-critical applications. As a result, current models can produce unsafe behavior, especially in open-world deployments where formal guarantees are necessary.

In safety-critical robotics and control applications, the need for such guarantees has historically been addressed through formal specifications. Formal specifications have long been used to specify operational rules, such as safety requirements or mission directives for robotic deployments [39, 97]. Specifically, temporal logics such as *linear temporal logic (LTL)* [107] and *signal temporal logic (STL)* [92] provide a mathematically precise way to encode desired robotic behavior. Designers can write temporal logic specifications that are defined over system traces (e.g., state trajectories), and these specifications are used for safe planning. STL has seen particular interest due to its quantitative measure of satisfaction [37]. Recent work has explored integrating these logics into learning-based frameworks [51, 96, 117], but they have yet to catch up with the modern paradigms driving pretraining-based decision-making in robot learning.

My work explores the integration of temporal logic specifications for modern AI-driven planning. The goal is to extend existing temporal logic frameworks to ensure safety-aware planning while being integrated with modern deep learning-based behavior generation.

1.1 Challenges

Despite prior work [96, 117, 130] in employing temporal logics for specification-satisfying planning as a way to ensure safety in robotics, several theoretical and practical challenges remain before such methods can be widely adopted. Here, I elaborate on three main challenges identified by prior work and propose techniques to address them.

Challenge 1: Efficiency There are two key efficiency challenges with existing STL specification mechanisms. First, STL admits nesting temporal operators that can, in turn, encode complex temporally extended goals [11]. Although these specifications can capture complex temporal behavior, planning from these specifications is computationally expensive and sometimes infeasible. For example, a common approach for planning from STL specifications, originally proposed in [109], involves encoding the STL specification and the system dynamics as Mixed Integer Program (MIP) constraints and solving the constrained optimization problem in a receding horizon fashion. MIP-based STL planners scale with the number of binary variables [79]. As the temporal operator nesting increases, the number of binary variables increases exponentially. Hence, while the rich semantics of STL allow expressing long-horizon complex goals by nesting temporal operators, their corresponding MIPs can be computationally inefficient. We call this the "depth" challenge.

Second, evaluating STL's quantitative measure of satisfaction [37] for a large number of trajectories can be prohibitively slow. This inefficiency arises from the recursive nature of STL robustness evaluation. The semantics of temporal operators such as **G** (globally), **F** (eventually), and **U** (until) require aggregating robustness values across multiple timesteps, where each computation depends on the results of previous timesteps. As a result, robustness must be evaluated sequentially along the trajectory, preventing parallelization over the time axis. Due to these sequential dependencies, computational overhead for evaluating robustness for long-horizon trajectories or a large batch of trajectories can be significantly high. This problem is exacerbated for STL formulas with nested temporal operators. Hence, large-scale applications that rely on robustness computation face a fundamental recursion bottleneck, limiting scalability in both offline training and online control.

Approach: We propose two techniques to remedy these challenges. To address the depth challenge, we propose *STL decomposition* techniques that can transform a complex specification with arbitrarily nested temporal operators into a set of reachability and invariance constraints with minimal temporal nesting. These constraints can then be incrementally satisfied, thus reducing computational overhead. Our decomposition technique makes no assumption about the underlying planner and can be applied to long-horizon specifications. To address the breadth challenge, we propose *STLCG++*, a masking-based deep learning library for efficient STL evaluation. Our key insight is to use clever masking mechanisms to evaluate robustness for each timestep in parallel instead of sequentially processing each timestep as proposed in the prior literature.

Challenge 2: Generality – Expanding Temporal Logic Beyond Explicit States While TL specifications are a rich formalism for expressing complex behavior, they are traditionally defined over explicit system state variables that can be observed or estimated through sensors (e.g.,

the velocity of a robot). This limits their application for learning-based robotic systems that rely on high-dimensional data such as perception sensor data. Perpetual access to these system state variables is a limiting assumption that often does not hold true in practice. For example, within navigation domains, sophisticated localization and mapping modules are designed to map high dimensional perceptual inputs to state variables [24]. Additionally, when specifications are defined directly over explicit state variables, designers must manually specify upper and lower bounds for each predicate (e.g., position $x \in [x_{\min}, x_{\max}]$). Writing these precise numeric ranges can be cumbersome and error-prone, as small inaccuracies can yield overly conservative or overly permissive behaviors [110].

Approach: We propose *Embedding Temporal Logic (ETL)* that allows formulating formal specifications with open-ended alphabets by defining predicates over learned representations rather than explicit states. These learned representations are embedded representations of images that are processed by pretrained modern vision transformers such as ViT [35] and CLIP [27]. By lifting the problem to an embedding space, we leverage pre-trained vector representations to overcome the challenge of predefined finite alphabets. In this context, pre-trained embeddings decouple representation learning from downstream tasks and offer high-quality representations without additional training. This allows designers to write these specifications at a higher level of abstraction by defining specifications in terms of semantic concepts or task-relevant features, abstracting away low-level state representations. These higher level specifications can then be used for run-time monitoring of these specifications for learning-enabled systems without precise state localization.

Challenge 3: Applicability: Enforcing Temporal Constraints in Foundation Models While Preserving Base Behavior Although temporal logic has seen success in classical robotic planning for reliable behavior generation, its use for foundation models remains limited. Additionally, re-training or fine-tuning these large pre-trained models to directly embed temporal logic specification is challenging [72]. First, retraining models is a costly endeavor in terms of computational resources and data requirements. Moreover, due to the stochastic nature of these models, it is difficult to guarantee strict satisfaction of temporal constraints through training alone. Another approach to enforcing these constraints is through run-time monitoring [6, 16]. Traditional run-time enforcement of temporal logic constraints relies on filtering: discarding outputs that violate a specification. Although effective in some settings, this approach distorts the output distribution of foundation models (FMs), which are designed to generate coherent, high-probability outputs. Since these models have been pre-trained on Internet scale data and possess rich semantic information, overriding their proposed actions agnostically can lead to degenerate behaviors. Hence, there is a pressing need for methods that can enforce temporal specifications efficiently at inference time without disrupting the model’s pre-trained behavior.

Approach: This thesis aims to address this gap by developing inference-time methods for enforcing temporal properties for these foundation models, inspired by recent work in constrained decoding [2, 19, 127] for large language models. These approaches typically mask out tokens that violate a syntactic constraint defined over token sequences. For example, regular expressions (regex) represent a widely used form of syntactic constraint, requiring that generated token sequences conform to predefined structural patterns [19, 127]. We extend the constrained decoding

paradigm to enforce STL specifications on candidate action trajectories and propose *specification aligned decoding (SafeDec)*. Our key insight is that decoding-time interventions can be used not just to filter unsafe actions, but also to *condition the generation process itself* on specification satisfaction. This conditioning is critical because it steers the model toward generating specification-satisfactory actions rather than relying on post hoc rejection. SafeDec reduces the risk of infeasible outputs while preserving the original action distribution of the model.

1.2 Thesis Statement

Existing temporal logic frameworks face fundamental challenges in efficiency, generality, and applicability, which make them difficult to integrate with modern AI-driven planning systems. This thesis demonstrates that these challenges can be addressed by: (1) More efficient methods for evaluating long-horizon logical specifications through STL decomposition and STLCG++, (2) A novel specification paradigm defined over image features via Embedding Temporal Logic (ETL), and (3) A technique for enforcing invariant STL specifications for robot foundation models through constrained decoding.

These challenges arise in three main forms. First, existing temporal logic frameworks face efficiency limitations: evaluating or planning from deeply nested STL formulas is computationally expensive, and current formulations scale poorly with trajectory length or horizon of the formula. In this thesis, efficiency is measured through wall-clock run-time improvements in robustness evaluation and planning compared to existing baselines such as MICP and STLCG. Second, they lack generality, as existing temporal logics operate over explicit, low-dimensional state variables and cannot naturally express specifications grounded in high-dimensional perceptual representations such as images or language. Generality is assessed by the ability of ETL to express specifications directly in the embedding space without requiring explicit state access. Finally, they suffer from limited applicability in modern AI-driven systems, where enforcing temporal constraints in large pre-trained foundation models requires either expensive retraining or filtering outputs leading to reduction in task success rate. Applicability is evaluated by showing that specification-aligned constrained decoding methods can enforce STL invariants at inference time, without retraining, while preserving the model’s base success rate.

1.3 Summary of Contributions

The contributions of this thesis include:

1. STL decomposition, which improves the efficiency of specification-driven planning by decomposing temporal logic specifications.
2. STLCG++, which improves the efficiency of specification-driven planning by enabling scalable and parallelizable evaluation of temporal logic specifications.

3. Embedding Temporal Logic (ETL), which extends temporal logic to high-dimensional perception-based inputs, broadening the applicability of temporal logics for AI-driven planning.
4. Specification Constrained Decoding for RFMs, a post-processing method that enforces STL constraints in foundation model inference, that generates specification-satisfying actions without retraining.

Together, these contributions enable more structured, interpretable, and computationally efficient planning in AI systems, bridging the gap between logic-based planning and data-driven methods.

1.4 Outline

The remainder of the thesis is outlined as follows. Chapter 2 provides important background and related work about temporal logic and planning used throughout the document. In the following chapters, I will discuss my solutions that addresses the respective challenges. Chapter 3 investigates a decomposition technique for translating complex long horizon STL specifications into a set of reachability and invariance constraints. Chapter 4 investigates a technique for improving STL robustness evaluation using computational graphs. Chapter 5 discusses design and application of a novel embedding space logic called ETL for high-level specification and monitoring. Chapter 6 discusses a technique for runtime enforcement of temporal logic requirements in modern robotic foundation models. Lastly, I discuss future research directions in Chapter 7 and conclude with a summary of my contributions in Chapter 8.

Chapter 2

Background and Related Work

2.1 Temporal Logics for Robotics

There is a long line of work on specifying and verifying complex behaviors in cyber-physical and robotic systems using temporal logics. Temporal logics such as *linear temporal logic (LTL)* [107], *metric temporal logic (MTL)* [77], and *signal temporal logic (STL)* [92] provide a precise way to encode objectives that are expressed in a natural language. These logics have been used for trajectory planning [78, 82, 117], reinforcement learning [3, 4, 7, 140], runtime monitoring [16, 139], and adaptive control [18, 71, 85, 109].

The logics outlined above can struggle with systems that rely on ML for perception, where input data can have a variable number of objects in frames and evolving bounding boxes. This has led to spatial extensions of STL and MTL that allow one to specify properties with geometric interpretations [20, 49]. Specialized logics such as Timed Quality Temporal Logic (TQTL) [33] and Spatio-Temporal Quality Logic (STQL) [13, 54] have been proposed for perception systems that permit reasoning about properties over bounding boxes used in object detection. Recently, Spatiotemporal Perception Logic (STPL) [55] was introduced that combined TQTL with spatial logic and allows quantification over objects, as well as 2D and 3D spatial reasoning. STPL allows expressing properties by specifying relations between objects over time. Most of these logics are grounded in the symbolic outputs of a perception module (object labels, confidences, track IDs).

2.2 Signal Temporal Logic

STL is used specifically to define properties of continuous time real valued signals [92]. A signal $\mathbf{s}$ is a function $\mathbf{s} : \mathbb{T} \rightarrow \mathbb{R}^n$ that maps a time domain $T \subseteq \mathbb{R}_{\geq 0}$ to a real valued vector. Then, an STL formula is defined as:

$$\phi := \mu \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \mathcal{U}_{[a,b]} \psi$$

where μ is a predicate on the signal $\mathbf{s}$ at time t in the form of $\mu \equiv \mu(\mathbf{s}(t)) > 0$ and $[a, b]$ is the time interval (or simply I). The *until* operator $\mathcal{U}$ defines that ϕ must be true until ψ becomes true within a time interval $[a, b]$. Two other operators can be derived from *until*: *eventually* ($F_{[a,b]} \phi := \top \mathcal{U}_{[a,b]} \phi$) and *always* ($G_{[a,b]} \phi := \neg F_{[a,b]} \neg\phi$).

Definition 1. Given a signal s_t representing a signal starting at time t , the Boolean semantics of satisfaction of $s_t \models \phi$ are defined inductively as follows:

$$\begin{aligned}
s_t \models \mu &\iff \mu(s(t)) > 0 \\
s_t \models \neg\phi &\iff \neg(s_t \models \phi) \\
s_t \models \phi_1 \wedge \phi_2 &\iff (s_t \models \phi_1) \wedge (s_t \models \phi_2) \\
s_t \models F_{[a,b]}(\phi) &\iff \exists t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi \\
s_t \models G_{[a,b]}(\phi) &\iff \forall t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi
\end{aligned}$$

Apart from the Boolean semantics, quantitative semantics are defined for a signal to compute a real-valued metric indicating *robustness*, i.e., the strength of satisfaction or violation.

Definition 2. Given a signal s_t representing a signal starting at time t , the quantitative semantics of satisfaction of $s_t \models \phi$ are defined inductively as follows:

$$\begin{aligned}
\rho(s_t, \mu_c) &= \mu(x_t) - c \\
\rho(s_t, \neg\phi) &= -\rho(s_t, \phi) \\
\rho(s_t, \phi_1 \wedge \phi_2) &= \min(\rho(s_t, \phi_1), \rho(s_t, \phi_2)) \\
\rho(s_t, F_{[a,b]}(\phi)) &= \max_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi) \\
\rho(s_t, G_{[a,b]}(\phi)) &= \min_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi)
\end{aligned}$$

For example, suppose that we are given (1) $\phi \equiv G_{[0,3]}(\text{distToR3}(t) \geq 3.0)$, which states that the agent should maintain at least 3.0 meters away from region R_3 for the next 4 time steps and (2) signal s_t that contains sequence $\langle 3.0, 2.5, 3.0, 3.5 \rangle$ for distToR3 . Evaluating the robustness of satisfaction of ϕ over s_t would result in a value of -0.5 , implying that the agent violates the property by a degree of 0.5 (i.e., it fails to stay away from R_3 by 0.5 meters).

2.3 Planning from STL specifications

Planning from STL specifications is an active area of research for which multiple approaches have been proposed in the past few years [3, 4, 85, 109, 111]. One of the first papers in this direction involved translating STL specifications into constraints within a Mixed Integer Linear Program (MILP) [109]. This approach is sound and complete but faces scalability challenges for long-horizon specifications. To remedy this drawback, the original encoding has been modified by focusing on abstraction-based techniques [117] and reducing binary variables via logarithmic encoding [79]. Most of these techniques focus on reducing the MILP's complexity to observe performance benefits. Recently, the focus has shifted to developing techniques that leverage robustness feedback as a heuristic for trajectory synthesis instead of using MILP. These techniques

Table 2.1: Summary of existing python-based STL toolboxes that are publicly available.

Name	AutoDiff	Vectorize	GPU
STLCG [82]	PyTorch	✓	✓
Argus [12]	✗	✗	✗
stlpy [79]	✗	✗	✗
PyTeLo [23]	✗	✗	✗
pySTL [120]	✗	✗	✗
RTAMT [132]	✗	✗	✗

involve using reinforcement learning [3, 66], search-based techniques [4] and control barrier functions [85] to generate STL satisfying trajectories. While these methods offer greater scalability, they are not complete and frequently struggle to accommodate complex specifications because of the intrinsically non-convex optimization problem posed by robustness semantics.

To overcome challenges of planning from complex STL specifications, Decomposition of STL specifications has been studied before in [80, 134]. In [134], the authors restrict themselves to an STL fragment that does not allow nesting of temporal operators. In [80], the authors perform structural manipulation using a tree structure. However, their focus is on multi-agent setups and they handle nested operators conservatively, especially for the eventually operator. This conservative notion generates specification satisfying behavior but it can be overly restrictive.

2.4 STL Evaluation and Monitoring Tools

As STL has been used for various applications, a variety of STL libraries across different programming languages have been developed, including Python, C++, Rust, and Matlab. Given that Python is commonly used for robotics research, Tab. 2.1 compares recent STL Python packages regarding automatic differentiation (AD), vectorization, and GPU compatibility. Most libraries offer evaluation capabilities of a single signal, or their design is tailored towards a specific use case, making it difficult to extend or apply them to new settings. If users want to perform an optimization utilizing STL robustness formulas, a separate optimization package (e.g., CVXPY [32], Drake [118]) is often required.

RTAMT [132] was introduced as a unified tool for offline and online STL monitoring with an efficient C++ backend. It has received widespread support and has superseded other alternatives in terms of usage. However, RTAMT performs CPU-based signal evaluation and lacks differentiation and vectorization capabilities, limiting its efficiency in handling and optimizing over large datasets, where AD and GPU compatibility are crucial. STLCG was the first to introduce vectorized STL evaluation and backpropagation by leveraging modern AD libraries. However, STLCG faced scalability challenges due to its underlying recurrent computation.

2.5 Runtime Safety Enforcement for Robotics

Constraint satisfaction for robotics has been an active area of research that involves techniques such as control barrier functions (CBFs) [8], safe reinforcement learning [43], and temporal logic-based shielding approaches [6]. Recently, with the advent of vision language action models and their impressive generalizable capabilities for manipulation, navigation and other tasks, there are growing concerns about ensuring safety and correctness without retraining these large models. Although classical methods offer formal guarantees, they either require pretraining/fine-tuning stage interventions or designing a new classical controller for each safety specification, which can be restrictive. For example, SafeVLA [138] fine-tunes pre-trained foundation models with task-specific safety costs, achieving strong performance in Safety-CHORES tasks. However, the safety specification is expected to be embedded in the training data and loss, meaning the model cannot generalize to new safety constraints at test time. In contrast, ASIMOV [113] explores rewriting dangerous instructions with better human-aligned alternatives to steer model behavior without modifying model parameters, but lacks trajectory-level formal guarantees. Our technique achieves a middle ground with the ability to adapt to novel specifications at test time without modifying model parameters while requiring minimal assumptions about the underlying model. The closest to our work is SELP [128] that proposes LTL-constrained decoding for language model-based plan generation. However, SELP is unsuitable for STL because its Boolean predicate-based LTL cannot encode numeric bounds (e.g. $\|x - x_{goal}\| < 0.1m$) and does not possess quantitative semantics, which is crucial for ranking actions. These techniques are also tailored to high-level plans from LLMs, not to the per-step low-level actions generated online by policies.

2.6 Connections to safe reinforcement learning and control

We adopt STL as the specification formalism for runtime safety enforcement because it offers several advantages over barrier functions, Hamilton-Jacobi reachability, and safe reinforcement learning [25]. Prior work emphasizes that STL is especially well suited to robotics because it explicitly reasons over real-valued signals and dense time, while also providing quantitative robustness semantics that measure the degree of satisfaction or violation of a specification. This makes STL substantially more expressive than barrier-based methods, which are effective for enforcing instantaneous invariance [8] but are not naturally designed to represent temporally extended requirements such as “eventually,” “until,” or ordered task progression. Relative to HJ reachability, STL provides a richer compositional language for specifying complex behaviors, rather than restricting reasoning to individual reachability or reach-avoid computations [25]. Relative to safe RL, STL offers a transparent and human-interpretable safety interface that does not depend on reward shaping or training distribution coverage [3, 14]. Thus, STL provides a practical middle ground: it retains the structure and interpretability of formal methods while remaining compatible with continuous robotic signals and runtime quantitative evaluation.

Chapter 3

STL Decomposition for Efficient Planning

Most autonomous robots interacting with the physical world need to achieve complex objectives while dealing with uncertainty and stochasticity in their environment. This problem is exacerbated by short response times expected while ensuring runtime efficiency. Hence, formulating these complex objectives accurately is a crucial step in realizing the desired behaviors for robotic operations.

Temporal logics such as LTL and STL provide a precise way to encode objectives that are expressed in a natural language. STL has received special attention in the community due to its rich quantitative semantics that can quantitatively measure satisfaction of a given property that encodes an objective. Additionally, it can be used to describe complex properties over real valued signals such as state trajectories arising from continuous dynamical systems. For robotic planning, STL can be used to describe complex behaviors with concrete time deadlines such as those found in trajectory planning and task planning. Planners can use these specifications to generate specification-conforming behavior.

A significant amount of common robotic objectives can be interpreted as a sequence of sub-tasks. It has been shown that incremental subtask planning can be done more efficiently compared to planning for a composite task [21, 30, 99]. However, when STL is used to represent these composite tasks, incremental planning becomes challenging. This issue is because STL semantics can encode the sequential nature of tasks but does not expose this structure to the planner. In such cases, the planners are forced to work with complex long-horizon specifications. When the horizon of the specification is longer than the planning horizon, planners can often generate suboptimal or violating plans. This problem is exacerbated when planning occurs at runtime with computational constraints and compounding modeling errors [4]. Due to the exacerbation of this problem with temporal nesting, we term this challenge the "depth" challenge.

To overcome this challenge, we propose a theory to decompose long-horizon, arbitrarily nested specifications into sub-specifications that can be satisfied incrementally. We define recursive rules for decomposition and propose a novel scheduling algorithm for incremental task planning. The key insight here is to "divide and conquer" STL requirements while ensuring, by construction, that the resulting plan satisfies the original composite specification. We illustrate the effectiveness of our proposed approach over an experiment involving robot exploration problems with linear and non-linear dynamics. Our experiment shows that our approach is able to more efficiently generate

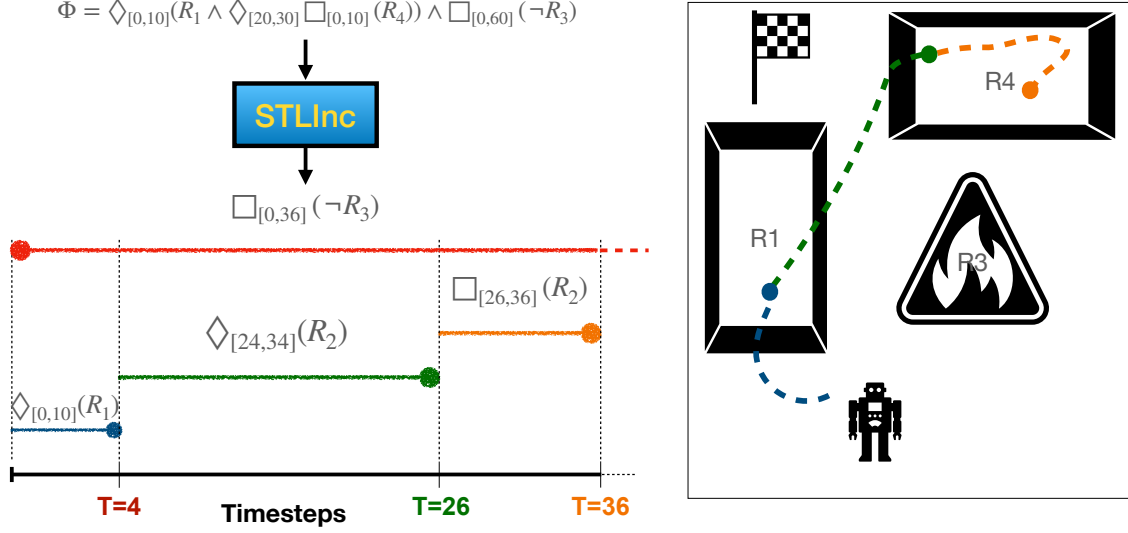


Figure 3.1: **Left:** An STL specification ϕ with multiple nested temporal operators and a possible decomposition into subtasks. **Right:** A sample trajectory that satisfies ϕ in a planar environment.

plans for complex, composite specifications in comparison to the existing state-of-the-art STL-based planning methods. In addition, our decomposition technique is agnostic to the underlying system dynamics and the choice of planner, and can potentially be adapted by different planners.

3.1 Motivation

We illustrate the problem of planning from complex specifications using an example from the motion planning domain. We use a planning problem similar to the one defined in [82].

As illustrated in Figure 3.1, the goal of the agent (robot) is to visit regions R_1 and R_4 sequentially while avoiding an unsafe region, R_3 . Additionally, upon reaching R_4 , the agent needs to stay in it for 10 time steps. We combine three common motion planning patterns such as sequenced visit, stabilization, and global avoidance to create a specification with timed deadlines as follows:

$$\begin{aligned}\Phi &= \phi_1 \wedge \phi_2 \\ \phi_1 &= \Diamond_{[0,10]}(R_1 \wedge \Diamond_{[20,30]}\Box_{[0,10]}(R_4)) \\ \phi_2 &= \Box_{[0,60]}(\neg R_3)\end{aligned}$$

The state-of-the-art (SOTA) technique for planning from Φ , originally proposed in [109], involves encoding the STL specification and the system dynamics as Mixed Integer Program (MIP) constraints and solving the constrained optimization problem in a receding horizon fashion. A new binary decision variable is introduced for each atomic proposition per time step in the STL specification. A known drawback of this technique is its exponential worst-case complexity with respect

to the number of binary variables [79]. Various encoding modifications have been suggested to enhance the efficiency of the technique by reducing the number of variables and constraints [79, 111].

However, even with reduced variable encoding, current methods excel primarily with short-horizon specifications. When encoding nested temporal operators, a large number of additional variables and constraints are needed to capture the relationship between different temporal operators, in contrast to non-nested operators, where temporal constraints associated with each operator are considered independently. For example, let us take the subformula $\Diamond_{[20,30]}\Box_{[0,10]}(R_4)$ from ϕ_1 . For encoding this subformula into an MIP, we would need 11 (outer eventually) + 121 (inner always) = 132 binary variables.¹ In general, as the nesting depth increases, the number of variables can increase exponentially.

In this work, we propose a technique to improve the scalability of STL planning algorithms through decomposition of STL specifications. Our idea is inspired by human planning, where long-term goals are achieved by breaking tasks into incremental sub-goals [34]. Concretely, by decomposing the specification, we effectively remove the complexity of nested operators and also reduce the length of the lookahead horizon.

A possible decomposition of the specification Φ into four subtasks is as follows:

$$\begin{aligned} sch_1 &= \Diamond_{[0,10]}(R_1) \wedge \Box_{[0,10]}(\neg R_3) \\ sch_2 &= \Diamond_{[t_{R1}+20, t_{R1}+30]}(R_4) \wedge \Box_{[t_{R1}+20, t_{R1}+30]}(\neg R_3) \\ sch_3 &= \Box_{[t_{R4}+0, t_{R4}+10]}(\neg R_3 \wedge R_4) \\ sch_4 &= \Box_{[t_{R42}, 60]}(\neg R_3) \end{aligned}$$

Here, the symbolic time variables $(t_{R1}, t_{R4}, t_{R42})$ indicate when those subtasks get satisfied. More specifically, t_{R4} indicates when the agent reaches Region 4 and t_{R42} indicates when the agent has been inside Region 4 for 10 timesteps after reaching it. These variables then shift the time intervals of the other constraints that depend on them (e.g., time t_{R1} from sch_1 is used to concretize the time intervals for sch_2 , whose time of satisfaction, in turn, influences sch_3). These subtasks have shorter time horizons and no nested temporal operators, resulting in MIP constraints that are less complex than those that would result from composite specifications. As shown later in Section 3.3, this decomposition-based approach has potential to significantly improve the efficiency of planning.

3.2 STLInc

The overview of our planning framework (STLINC) is shown in Figure 3.2. The key idea behind this approach is that a bounded STL formula in our fragment can be decomposed into a finite set of the following two types of *task constraints*, each of which is associated with time interval $I = [a, b]$ and state proposition p :

¹For the outer $\Diamond$ clause, 11 binary variables are introduced to encode that the inner $\Box$ clause is satisfied within interval [20,30]; for each time point in [0,10] interval, another set of 11 variables are introduced, thus resulting in $11*11 = 121$ variables.

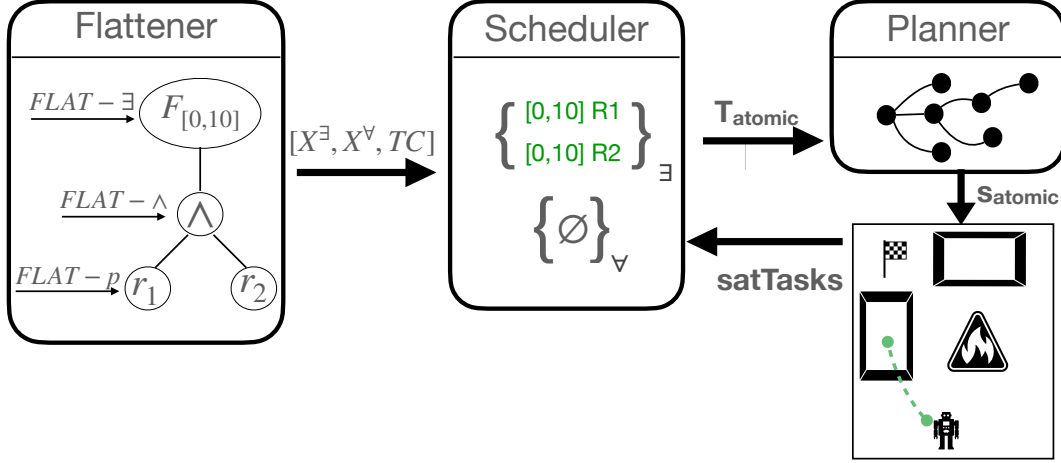


Figure 3.2: Overview of the STLINC approach

Reachability: The system ensures that p holds over at least one time step t within I .

Invariance: The system ensures that p holds over every step t within I .

Based on this idea, the framework carries out the incremental planning process over three steps. First, the *flattener* takes a user-specified STL specification (ϕ) and decomposes it into two sets of task constraints, $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$, which contain the reachability and invariance constraints, respectively. The decomposition is performed such that satisfying all of the constraints in these two sets implies the satisfaction of the original formula ϕ .

Next, the *scheduler* takes the two sets, $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$, and generates a sequence of *atomic tasks*, $\sigma = \langle at_0, at_1 \dots at_k \rangle$, where (1) each atomic task at is a non-nested STL formula consisting of $G_{[a,b]}(p)$ or $F_{[a,b]}(p)$ (where p is a propositional formula) and (2) two different atomic tasks do not overlap in their time intervals. After this sequence is generated, the *planner* executes these atomic tasks one by one in the designated order. Once all tasks are executed, the system will have fulfilled $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$, thus satisfying the original goal of ϕ . The rest of this chapter describes each of the three steps in detail.

STL fragment Our approach is designed to handle specifications written in the following fragment of STL:

$$\begin{aligned} \phi &::= F_{[a,b]}(\phi) \mid G_{[a,b]}(\phi) \mid \phi_1 \wedge \phi_2 \mid p \\ p &::= p_1 \wedge p_2 \mid p_1 \vee p_2 \mid \neg p \mid \mu \end{aligned}$$

where p is a propositional formula that does not contain any temporal operator, and μ is an atomic proposition. We target this STL fragment as (1) it significantly simplifies the decomposition process and (2) it is still expressive enough to capture many common behavioral patterns in the robotics planning domain [97]. Note that the fragment allows a nesting of temporal operators of an arbitrary depth, which is important for specifying sequential tasks. For example, the objective

of *visiting* locations in a particular order can be defined as:

$$\phi \equiv F_{[1,5]}(l_1 \wedge F_{[1,10]}(l_2 \wedge F_{[1,7]}(l_3)))$$

where l_1, l_2, l_3 represent the locations to be visited.

Task constraints As mentioned at the beginning of the section, an STL formula imposes two types of constraints over system behavior: *reachability* and *invariance* constraints, which are formally defined as follows:

Definition 3 (Reachability set). A *reachability set*, $\mathcal{X}^\exists \in \mathbb{P}(\mathbb{T} \times \mathbb{T} \times \mathbb{U})$ is a set of tuples of form (l, h, p) , each stating that there exists some time t in interval $[l, h]$, inclusively, such that proposition p holds over the system state at time t .

Definition 4 (Invariance set). An *invariance set*, $\mathcal{X}^\forall \in \mathbb{P}(\mathbb{T} \times \mathbb{T} \times \mathbb{U})$, is a set of tuples of form (l, h, p) , each stating that for every time t in interval $[l, h]$, inclusively, proposition p holds over the system state at time t .

We introduce an additional set of constraints that specify intervals over symbolic time variables that are introduced during the flattening of F formulas:

Definition 5 (Time variable intervals). A *time variable interval set*, $TC \in \mathbb{P}(\mathbb{T} \times \mathbb{T} \times \mathbb{T})$, is a set of tuples of the form (l, h, v) , each stating that symbolic time variable v takes a value in the interval $[l, h]$, inclusively.

3.2.1 Flattening

Given an input STL specification, ϕ , the goal of *flattening* is to construct two sets of constraints—reachability and invariance sets—whose satisfaction also implies the satisfaction of ϕ . Flattening is applied recursively based on the structure of ϕ , as shown through the rules in Figure 3.3. Along with $\mathcal{X}^\forall$ and $\mathcal{X}^\exists$, each recursive step produces an additional auxiliary output TC , which is later used by the scheduler to resolve symbolic time variables.

FLAT- p

In the basic case where $\phi = p$, flattening generates one reachability constraint that requires p to be satisfied at the current time (i.e., $l = h = 0$). Hence, the reachability set for ϕ is $(0, 0, p)$.

FLAT- $\wedge$

Given conjunctive formula $\phi = \phi_1 \wedge \phi_2$, the invariance set for ϕ is the union of $\mathcal{X}_1^\forall$ and $\mathcal{X}_2^\forall$; i.e., every invariance constraint in ϕ_1 and ϕ_2 must be satisfied. Similarly, the reachability set for ϕ is the union of $\mathcal{X}_1^\exists$ and $\mathcal{X}_2^\exists$; i.e., every reachability constraint in ϕ_1 and ϕ_2 must be satisfied.

FLAT- $\exists$

Given $\phi = F_{[a,b]}(\phi_1)$, for each constraint $(l_1, h_1, p_1) \in \mathcal{X}_1^\exists$, flattening involves shifting interval $[l_1, h_1]$ by a symbolic time variable $t \in [a, b]$. This bound on t is encoded by adding a tuple (a, b, t)

$$\begin{array}{c}
\frac{\phi = p}{\mathcal{X}^\exists = \{(0,0,p)\} \quad \mathcal{X}^\forall = \emptyset} \text{FLAT-}p \quad \frac{\begin{array}{l} \phi = \phi_1 \wedge \phi_2 \\ (\mathcal{X}_1^\exists, \mathcal{X}_1^\forall, TC_1) = \text{FLAT}(\phi_1) \\ (\mathcal{X}_2^\exists, \mathcal{X}_2^\forall, TC_2) = \text{FLAT}(\phi_2) \end{array}}{\begin{array}{l} \mathcal{X}^\exists = \mathcal{X}_1^\exists \cup \mathcal{X}_2^\exists \\ \mathcal{X}^\forall = \mathcal{X}_1^\forall \cup \mathcal{X}_2^\forall \\ TC = TC_1 \cup TC_2 \end{array}} \text{FLAT-}\wedge \\
\\
\frac{\phi = F_{[a,b]}\phi_1 \quad (\mathcal{X}_1^\exists, \mathcal{X}_1^\forall, TC_1) = \text{FLAT}(\phi_1)}{\begin{array}{l} \mathcal{X}^\exists = \{(t+l, t+h, p) \mid \exists(l, h, p) \in \mathcal{X}_1^\exists\} \\ \mathcal{X}^\forall = \{(t+l, t+h, p) \mid \exists(l, h, p) \in \mathcal{X}_1^\forall\} \\ TC = TC_1 \cup \{(a, b, t)\} \end{array}} \text{FLAT-}\exists \\
\\
\frac{\phi = G_{[a,b]}\phi_1, \quad (\mathcal{X}_1^\exists, \mathcal{X}_1^\forall, TC_1) = \text{FLAT}(\phi_1)}{\begin{array}{l} \mathcal{X}^\exists = \{(k+l, k+l, p) \mid \exists(l, h, p) \in \mathcal{X}_1^\exists \wedge k \in [a, b]\} \\ \mathcal{X}^\forall = \{(a+l, b+h, p) \mid \exists(l, h, p) \in \mathcal{X}_1^\forall\} \\ TC = TC_1 \end{array}} \text{FLAT-}\forall
\end{array}$$

Figure 3.3: Flattening rules

to TC . Intuitively, this can be understood from the following transformation similar to the one in [80]:

$$F_{[a,b]}(F_{[c,d]}(\phi_1)) = F_{[a+c, b+d]}(\phi_1)$$

Here, instead of adding a and b directly to the upper and lower bound, respectively of time interval $[c, d]$, we add a symbolic variable that can take a value between a and b . For example, given specification $\phi = F_{[2,5]}(p)$, $\mathcal{X}^\exists$ would contain constraint $(t+0, t+0, p)$ and TC would contain $(2, 5, t)$.

As another example, consider specification $\phi = F_{[2,5]}(p_2 \wedge F_{[1,3]}(p_1))$. When the $\text{FLAT-}\exists$ rule is applied, the resulting $\mathcal{X}^\exists$ contains $(t_2 + t_1 + 0, t_2 + t_1 + 0, p_1)$ and (t_2, t_2, p_2) , and TC contains $\{(1, 3, t_1), (2, 5, t_2)\}$. Here, t_1 is introduced when $\text{FLAT-}\exists$ rule is applied to the innermost F operator ($F_{[1,3]}$); t_2 is then introduced when $\text{FLAT-}\exists$ is applied for outermost F operator ($F_{[2,5]}$).

The flattening of an F formula applied to an invariance set is handled similarly. Consider a specification of the form

$$\phi \equiv F_{[a,b]}(\phi_1) = F_{[a,b]}(G_{[c,d]}(p))$$

with $\mathcal{X}_1^\forall = \{(c, d, p)\}$. Here, according to the Boolean STL semantics, there exists a $t \in [a, b]$ such that $\forall t' \in [t+c, t+d]$, p must be satisfied. Hence, in the resulting invariance set for ϕ , the time interval for the existing invariance constraint is shifted by t . This bound on t is encoded by adding a tuple (a, b, t) to TC . For example, given specification $\phi = F_{[2,5]}(G_{[3,10]}(p))$, $\mathcal{X}^\forall$ would contain constraint $(t+3, t+10, p)$ and TC would contain $(2, 5, t)$.

FLAT- $\forall$

The flattening of an $\forall$ formula over $\mathcal{X}_1^\exists$ is handled in the following way. Consider a specification of the form:

$$\phi \equiv G_{[a,b]}(\phi_1) = G_{[a,b]}(F_{[c,d]}(p))$$

with $\mathcal{X}_1^\exists = \{tc\} = \{(t_1, t_1, p)\}$ and $TC = \{(c, d, t_1)\}$. Since ϕ states that constraint tc must hold at every time step between $[a, b]$, the idea is to create multiple reachability constraints of form $(t_1 + k, t_1 + k, p)$, one for each time value k in interval $[a, b]$. For example, let $\phi = G_{[1,100]}(F_{[1,5]}(p))$. After flattening, $\mathcal{X}^\exists$ for ϕ would contain 100 reachability constraints, each in form of $(t_1 + k, t_1 + k, p)$ for $1 \leq k \leq 100$.

The flattening of an $\forall$ over $\mathcal{X}_1^\forall$ is handled in the following way. Consider a specification of the form

$$\phi \equiv G_{[a,b]}(\phi_1) = G_{[a,b]}(G_{[c,d]}(p))$$

with $\mathcal{X}_1^\forall = \{tc\} = \{(c, d, p)\}$. Since ϕ states that constraint tc must hold at every time step between $[a, b]$, by shifting the interval of each constraint by $[a, b]$, the resulting formula $\mathcal{X}^\forall$ is constructed as $\{(a+c, b+d, p)\}$. Intuitively, this can be understood from the following transformation similar to the one in [80]:

$$G_{[a,b]}(G_{[c,d]}(\phi_1)) = G_{[a+c, b+d]}(\phi_1)$$

We now introduce a theorem that establishes the soundness of our flattening operation with respect to a given STL specification.

Lemma 1. *Let ϕ be an input STL specification, and let $(\mathcal{X}^\exists, \mathcal{X}^\forall, TC)$ be the output of $\text{flatten}(\phi)$. Then, for every signal s_t :*

$$\forall v \in \mathcal{V}(\text{vars}(TC)) \bullet \forall x \in \mathcal{X}^\exists \cup \mathcal{X}^\forall \bullet s_t \models \text{Inst}(x, v) \implies s_t \models \phi$$

where vars is a function that returns the set of all symbolic variables in TC , $\mathcal{V}(\mathcal{X})$ is the set of all possible assignments of values to variables in $\mathcal{X}$ (restricted to values in their respective time intervals), and $\text{Inst}(x, v)$ instantiates symbolic variables in constraint x with the values from v .

In other words, if every possible reachability or invariance constraint is satisfied over a particular signal, then the original specification ϕ must also hold over the same signal.

3.2.2 Symbolic Time Resolution

The $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$ constraints generated from flattening may contain multiple symbolic variables. To enable scheduling, STLINC first attempts to resolve as many of these variables as possible by applying substitution rules to the constraints.

Algorithm 1 shows the sketch of the *symbolic time resolution (STR)* process. As inputs, it takes the output of the flattening procedure—the $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$ constraints, and the time intervals over symbolic time variables. As outputs, it produces (1) a new pair of $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$, with some of the time variables replaced by concrete time values, and (2) time variable intervals (TC') that specifies, for each concrete constraint produced in (1), an interval during which the constraint

Algorithm 1 Symbolic Time Resolution (STR)

```
1: Input: Flattened constraints  $\mathcal{X}^\exists, \mathcal{X}^\forall$ , time variable intervals  $TC$ 
2: Output: Modified constraints  $\mathcal{X}^\exists, \mathcal{X}^\forall$ , new time variable intervals  $TC'$ 
3:  $TC' = \emptyset$ 
4: for  $tc = (a, b, t)$  in  $TC$  do
5:    $\mathcal{X}_t^\forall := \text{FILTER}(\mathcal{X}^\forall, t)$ ,  $\mathcal{X}_t^\exists := \text{FILTER}(\mathcal{X}^\exists, t)$ 
6:   for  $x = (l, h, p)$  in  $\mathcal{X}_t^\forall$  do
7:      $(y^\exists, y^\forall) := \text{APPLYFG}(x, tc)$ 
8:      $\mathcal{X}^\exists := \mathcal{X}^\exists \cup \{y^\exists\}$ 
9:      $\mathcal{X}^\forall := (\mathcal{X}^\forall - \{x\}) \cup \{y^\forall\}$ 
10:     $TC' := TC' \cup \{(t+l, t+l, \text{T-SAT}(y^\exists)), (t+l+1, t+h, \text{T-SAT}(y^\forall))\}$ 
11:   end for
12:   for  $x = (l, h, p)$  in  $\mathcal{X}_t^\exists$  do
13:      $y^\exists := \text{APPLYFF}(x, tc)$ 
14:      $\mathcal{X}^\exists := (\mathcal{X}^\exists - \{x\}) \cup \{y^\exists\}$ 
15:      $TC' := TC' \cup \{(t+l, t+h, \text{T-SAT}(y^\exists))\}$ 
16:   end for
17: end for
18: return  $(\mathcal{X}^\exists, \mathcal{X}^\forall, TC')$ 
```

must be satisfied. This latter set of time intervals are used to enforce conjunctive constraints (i.e., $\phi_1 \wedge \phi_2$) to be satisfied simultaneously.

The algorithm iterates through the time variables in the bottom-up order (i.e., starting with the variables that appear in the lower part of an AST for a given STL expression). For each constraint that contains variable t (line 4), STR applies two types of substitutions, depending on whether the constraint is a reachability or an invariance constraint.

Invariance constraints

An invariance constraint, $x = (l, h, p)$, with time variable $t \in [a, b]$ represents an STL expression $F_{[a,b]}(G_{[l,h]} p)$ (where t appears in l and h , and p is an atomic proposition). Intuitively, this expression can be regarded a kind of “reach and stay” task, where the system must first reach a state where p holds within time interval $[a+l, b+l]$, and then continue to satisfy p for the following $(h-l)$ time steps. The rule APPLYFG takes constraint x and interval tc , and produces a new pair of reachability and invariance constraints, as follows:

$$\text{APPLYFG}(x = (l, h, p), tc = (a, b, t)) \equiv ((a+l, b+l, p), (t_{sat}, t_{sat} + h-l, p))$$

where t_{sat} is a new symbolic variable that represents the time at which p is satisfied between $(a+l, b+l)$.

In addition, STR adds two time variable intervals (line 10) to ensure that: (1) the new reachability constraint, $y^\exists$, is satisfied exactly at $(t+l)$ and (2) the invariance constraint, $y^\forall$, is satisfied

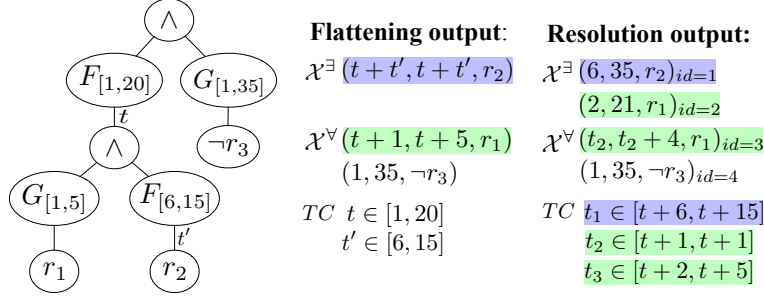


Figure 3.4: An AST for an example STL specification, $F_{[1,20]}(G_{[1,5]}(r_1) \wedge F_{[6,15]}(r_2)) \wedge G_{[1,35]}(\neg r_3)$, with outputs from the flattening and symbolic time resolution steps. Each constraint resulting from the resolution step is assigned an identifier (*id*); a symbolic time variable that represents the time of satisfying the constraint is annotated with *id* as the subscript (i.e., t_2 represents the satisfaction of reachability constraint $(2, 21, r_1)$).

subsequently for the following $(l - h)$ steps. Here, $\text{T-SAT}(y)$ returns a symbolic time variable representing the time of satisfaction of constraint y .

Consider the example in Figure 3.4. One of the invariance constraints that flattening generates is $(t + 1, t + 5, r_1)$, which depends on time variable $t \in [1, 20]$. The application of APPLYFG results in constraints stating that (1) the system must satisfy p within $[2, 21]$, and (2) from the point of the satisfaction of this constraint (t_2), it must hold p true for the following $(5 - 1) = 4$ steps. Note that the other invariance constraint, $(1, 35, \neg r_3)$, remains untouched, as it does not depend on any time variable.

Reachability constraints

A reachability constraint, $x = (l, h, p)$, with time variable $t \in [a, b]$ represents an STL expression $F_{[a,b]}(F_{[l,h]} p)$. When a pair of F operators are nested in this manner, they can be simplified by using the following substitution rule:

$$\text{APPLYFF}(x = (l, h, p), tc = (a, b, t)) \equiv (l + a, h + b, p)$$

The resulting reachability constraint, $y^\exists$, replaces the existing constraint in $\mathcal{X}^\exists$ (line 14). In addition, a new time variable interval is added to ensure that $y^\exists$ is satisfied between $(t + l)$ and $(t + h)$.

For the example in Figure 3.4, flattening produces one reachability constraint, $(t + t', t + t', r_2)$. During the first iteration of the outermost loop (line 4), STR selects $tc = (5, 15, t')$ and applies APPLYFF to the constraint, producing a new reachability constraint $(t + 5, t + 15, r_2)$. In next iteration, STR selects $tc = (1, 20, t)$ and applies APPLYFF to $(t + 5, t + 15, r_2)$, producing an additional reachability constraint $(6, 35, r_2)$.

Note that at the end of resolution for the example (Figure 3.4), variable t appears in multiple intervals in TC' . In the following section, we describe how STLINC schedules tasks to generate concrete values for the time variables incrementally one-by-one, ultimately synthesizing a plan that satisfies the original STL formula.

Algorithm 2 Schedule

```
1: Input:  $\mathcal{X}^\exists, \mathcal{X}^\forall, TC'$ 
2: Output: Signal  $s_{plan}$  for synthesized plan
3:  $s_{plan} = \langle \rangle$ 
4:  $\prec := \text{COMPUTEORDER}(\mathcal{X}^\exists, \mathcal{X}^\forall, TC')$ 
5:  $currTasks := \text{NEXTTASKS}(\mathcal{X}^\exists, \mathcal{X}^\forall, \prec, \emptyset)$ 
6: while  $currTasks \neq \emptyset$  do
7:    $currTasks := \text{SLICE}(currTasks)$ 
8:    $atomicTask := \text{NEXTATOMIC}(currTasks)$ 
9:    $currTasks := currTasks - \{atomicTask\}$ 
10:   $s_{atomic} := \text{PLAN}(atomicTask)$ 
11:  if  $s_{atomic} = \langle \rangle$  then break end if
12:   $s_{plan} := s \hat{\ } s_{atomic}$ 
13:   $satTasks := \text{EXTRACTTIME}(s_{atomic})$ 
14:   $currTasks := currTasks \cup \text{NEXTTASKS}(\mathcal{X}^\exists, \mathcal{X}^\forall, \prec, satTasks)$ 
15: end while
16: return  $s_{plan}$ 
```

Lemma 2. Let $(\mathcal{X}^\exists, \mathcal{X}^\forall, TC)$ be the output of $\text{flatten}(\phi)$ and $(\mathcal{X}_2^\exists, \mathcal{X}_2^\forall, TC')$ be the output of $\text{STR}(\mathcal{X}^\exists, \mathcal{X}^\forall, TC)$. Then, for every signal s_t :

$$\begin{aligned} \forall v \in \mathcal{V}(\text{vars}(TC')) \bullet (\forall x_2 \in \mathcal{X}_2^\exists \cup \mathcal{X}_2^\forall \bullet s_t \models \text{Inst}(x_2, v)) \implies \\ (\forall x \in \mathcal{X}^\exists \cup \mathcal{X}^\forall \bullet s_t \models \text{Inst}(x, v)) \end{aligned}$$

In other words, if every reachability and invariance constraint generated from STR is satisfied under some instantiation (v) of symbolic time variables in TC' , then the original set of flattened constraints must also be satisfied under the same condition.

3.2.3 Scheduling

Given the output from the resolution step (the two constraint sets, $\mathcal{X}^\exists, \mathcal{X}^\forall$ and the time interval variables, TC'), the goal of the scheduler is synthesize a plan that satisfies the original STL specification ϕ . To achieve this, the scheduler iteratively interacts with a *planner* that is capable of synthesizing a plan to satisfy an *atomic task* formula of form $F_{[a,b]}(\phi_1) \wedge G_{[a,b]}(\phi_2)$, where ϕ_1 and ϕ_2 are quantifier-free STL expressions. The scheduling process is incremental: The scheduler generates a sequence of atomic tasks formulas and invokes the planner to solve them one-by-one, using information (i.e., the time of satisfaction of a reachability or invariance constraint) generated by the planner to resolve any dependencies on symbolic time variables that were introduced during the flattening and resolution steps. The scheduling algorithm (Alg. 2) comprises of three major parts: ordering, slicing of constraints, and planning of atomic tasks.

Ordering In the first step (line 4), the scheduler computes a partial order ($\prec$) among the given reachability and invariance constraints, to determine which of these constraints must be satisfied before others. In particular, given a pair of constraints, x_1 and x_2 , $x_1 \prec x_2$ if and only if the time of satisfaction of x_1 necessarily precede that of x_2 , based on the time intervals that are assigned to those constraints in $\mathcal{X}^\exists$ or $\mathcal{X}^\forall$.

If one or more of x_1 and x_2 depends on another symbolic variable, t , for satisfaction, then the information in TC' is used to determine the presence of a precedence relationship. Consider t_1 and t_3 from Figure 3.4; after resolution, the satisfaction of these two constraints depend on the symbolic variable t (as specified in TC'). Although the value of t is unknown, it can be determined that for any possible value of t , t_3 will necessarily be satisfied before t_1 (i.e., $t_3 \prec t_1$). Overall, for this example, COMPUTEORDER determines that $t_2 \prec t_3 \prec t_1$. Note that t_4 does not appear in this ordering as it needs to be satisfied in parallel with these other constraints.

Slicing The scheduler then determines the first set of constraints (or tasks) to be carried out based on the order $\prec$ (line 5). In general, the time intervals over these constraint may overlap with each other in an arbitrary way. Recall, however, that each atomic task to be solved by the planner must be in form $F_{[a,b]}(\phi_1) \wedge G_{[a,b]}(\phi_2)$. Thus, the scheduler must first convert the constraints in *currTasks* into a set of atomic tasks constraint; this step involves *slicing* one or more constraints in *currTasks* (line 7).

Our slicing algorithm is illustrated in Algorithm 3. We illustrate its application using an example, shown in Figure 3.5. For our running example, the first set of constraints to be fulfilled is $\{t_2 = (2, 21, r_1), t_4 = (1, 35, \neg r_3)\}$. In the first step (line 3), we identify the lowest and the highest time bounds out of all the constraints in *currTasks* (which, for the example, would be 1 and 35). Then, in lines 5 to 11, for the horizon, we identify which constraints are active at each given time step. After this step, in lines 14 to 29, we first create “slices”, which involves generating multiple time intervals out of time steps where similar constraints are active. Then, we create constraints of type $\mathcal{X}^s$ and $\mathcal{X}^\forall$ out of them by combining propositions of constraints of the same type. For the running example, these time slices would be $[1, 1]$, $[2, 21]$ and $[22, 35]$. Concretely the SLICE operation slices t_4 into three constraints, t_{4a}, t_{4b}, t_{4c} ; this, in turn, results in the following three atomic tasks:

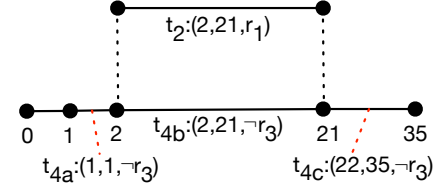


Figure 3.5: A slicing example.

$$G_{[1,1]}(\neg r_3) \quad F_{[2,21]}(r_2) \wedge G_{[2,21]}(\neg r_3) \quad G_{[22,35]}(\neg r_3)$$

However, the slicing step may introduce dependencies among the atomic tasks, especially for the constraints in $\mathcal{X}^\exists$. For example, suppose that task $F_{[2,21]}(r_2)$ is further split into two slices, $F_{[2,12]}(r_2)$ and $F_{[12,21]}(r_2)$. Since r_2 needs to be satisfied only once in interval $[2, 21]$, we would need to ensure that we are not over-constraining the space of possible behaviors by requiring r_2 to be satisfied twice, as the slices would imply. To achieve this, we keep track of dependencies among constraints, which are used by the scheduler to remove unnecessary atomic tasks (e.g., remove $F_{[12,21]}(r_2)$ once r_2 is satisfied between $[2, 12]$). This dependency management is handled inside NEXTTASKS (Algorithm 2, Line 14).

Algorithm 3 SLICE

```
1: Input: Set of currTasks  $\mathcal{X}^c$ 
2: Output: Set of atomic tasks constraints  $\mathcal{X}^{c'}$ 
3:  $t_{min}, t_{max} := \text{HORIZON}(\text{currTasks})$ 
4:  $active := \langle \rangle$ 
5: for  $t$  in  $[t_{min}, t_{max}]$  do
6:    $active_t := \langle \rangle$ 
7:   for  $x = (l, h, p)$  in  $\mathcal{X}^c$  do
8:     if  $t$  in  $[l, h]$  then  $active_t := active_t \widehat{\cap} \langle x \rangle$  end if
9:   end for
10:   $active := active \widehat{\cap} active_t$ 
11: end for
12:  $t_{low} := t_{min}$ 
13:  $\mathcal{X}^{c'} = \{ \}$ 
14: for  $t$  in  $[t_{min} + 1, t_{max}]$  do
15:   if  $active[t - 1] \neq active[t]$  then
16:      $\mathcal{X}^{temp} := active[t - 1]$ 
17:      $\mathcal{X}_{\exists}^{temp} := \{(t_{low}, t - 1, \top)\}$ 
18:      $\mathcal{X}_{\forall}^{temp} := \{(t_{low}, t - 1, \top)\}$ 
19:     for  $x = (l, h, p)$  in  $\mathcal{X}^{temp}$  do
20:       if  $x$  in  $\mathcal{X}_{\exists}^{temp}$  then
21:          $\mathcal{X}_{\exists}^{temp}.p := \mathcal{X}_{\exists}^{temp}.p \wedge p$ 
22:       else
23:          $\mathcal{X}_{\forall}^{temp}.p := \mathcal{X}_{\forall}^{temp}.p \wedge p$ 
24:       end if
25:     end for
26:      $\mathcal{X}^{c'} := \mathcal{X}^{c'} \cup \mathcal{X}_{\exists}^{temp} \cup \mathcal{X}_{\forall}^{temp}$ 
27:      $t_{low} := t$ 
28:   end if
29: end for
30: return  $\mathcal{X}^{c'}$ 
```

	STL Specification	Pattern
ϕ_1	$F_{[0,15]}(R_1) \wedge F_{[5,25]}(R_2) \wedge F_{[20,30]}(R_3) \wedge G_{[0,40]}(\neg O_1)$	R+A
ϕ_2	$F_{[0,15]}(R_1 \wedge F_{[0,15]}(R_2)) \wedge G_{[0,40]}(\neg O_1)$	SV+A
ϕ_3	$F_{[0,15]}(R_1 \wedge F_{[0,15]}(R_2 \wedge F_{[0,20]}(R_3 \wedge F_{[0,15]}(R_1))))$	SV
ϕ_4	$F_{[0,15]}G_{[0,10]}(R_1) \wedge F_{[0,35]}(R_2) \wedge G_{[0,40]}(\neg O_1)$	R+A+SB
ϕ_5	$F_{[0,15]}(R_1 \wedge F_{[0,20]}G_{[0,10]}(R_2))$	SV+SB

Table 3.1: Benchmark STL specifications created from motion planning patterns. Here, R: Reach, A: Avoid, SV: Sequenced Visit, SB: Stabilization.

Planning atomic tasks Once atomic tasks have been generated through slicing, the scheduler selects the next atomic task and invokes the planner (lines 8-10). If the planner is able to synthesize a plan that satisfies the atomic task, it returns a signal that represents the satisfying trajectory, which is then appended to the cumulative signal s_{plan} (line 12); if not, the scheduler terminates by returning the signal that contains a partially satisfactory plan (line 11).

From the synthesized signal, the scheduler extracts the constraints from $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$ that were satisfied, along with the concrete time values for their satisfaction (line 13). This information ($satTasks$) is then used to determine the next increment of constraints to be solved.

Finally, when all of the constraints in $\mathcal{X}^\exists$ and $\mathcal{X}^\forall$ have been satisfied, the scheduler terminates by returning s_{plan} as the final plan.

Lemma 3. *Let s_{plan} be the output of $Schedule(\mathcal{X}^\exists, \mathcal{X}^\forall, TC')$ and v be an assignment of values to symbolic time variables in TC' , which are determined during the scheduling step. Then, the following statement holds:*

$$\forall x \in \mathcal{X}^\exists \cup \mathcal{X}^\forall \bullet s_{plan} \models Inst(x, v)$$

In other words, the synthesized plan, s_{plan} , satisfies all of the reachability and invariance constraints that were generated from the preceding flattening and STR steps.

Building on Lemmas 1, 2, and 3, we finally introduce a theorem to state that our proposed approach generates a plan that satisfies the given specification ϕ :

Theorem 4. *Given specification ϕ and s_{plan} as the output of the scheduling algorithm, $s_{plan} \models \phi$.*

3.3 Evaluation

3.3.1 Experimental Setup

Specifications.

We investigated multiple motion planning STL specifications from [26, 97]. Based on the most common planning patterns, such as Reach (R), Avoid (A), Stabilisation (SB), Sequenced Visits (SV) etc., we created representative STL benchmark specifications as outlined in Table 3.1. These specifications are defined over STL subformulas of the form R_i or O_i where R_i / O_i is satisfied

if the agent is inside Region i or Obstacle i . These subformulas are defined in a similar fashion using conjunction of linear and nonlinear predicates as done in [79]. Please refer to [79] for more information on how these are defined for rectangular/circular regions.

Implementation Details.

We investigate planning from benchmark specifications in two robot exploration environments (similar to Figure 3.1), namely LinEnv and NonLinEnv created using STLPY [79]. STLPY has the functionality to encode any arbitrary STL formula, dynamics and actuation limits into constraints and use existing state-of-the-art solvers (Gurobi [46], SNOPT [41], etc.,) to generate satisfying plans. Our two environments are both planar but differ in underlying dynamics governing the robot. LinEnv has linear dynamics (Double Integrator) whereas NonLinEnv has nonlinear dynamics (Unicycle). We benchmark our technique against existing MICP methods (for linear dynamics) and other gradient-based techniques like SNOPT (for non-linear dynamics). We use Python to implement our tool² while using stlpy and Drake [118] to encode the STL constraints. Additionally, we use Gurobi or SNOPT to solve the final constrained optimization problem. All experiments were run on a workstation with an Intel Xeon W-1350 processor and 32 GB RAM.

Benchmarks and Research Questions.

We compare against the state-of-the-art techniques proposed in [18] (which we call *standard MICP*) and [79] (*reduced MICP*). Since the standard MICP encoding is only defined for environments with linear dynamics, we compare our technique against reduced MICP encoding for NonLinEnv. Reduced MICP claims better performance over standard MICP for long horizon and complex specifications due to their efficient encoding of disjunction and conjunction with fewer binary variables. However, standard MICP is faster for short-horizon specifications due to solver-specific presolve routines that leverage the additional binary variables for simplification.

Since our focus is on both short- and long-horizon specification with deep levels of temporal operator nesting, we benchmark against both techniques. The two main metrics we are concerned with are the time taken for solving and the final robustness values. To make the comparison fair, the total time taken by our technique includes the time taken by the flattener, scheduler, and solvers.

The two main research questions we investigate in this work are:

1. **RQ1:** Does our decomposition technique result in shorter solve times?
2. **RQ2:** Does our decomposition technique result in higher robustness scores?

3.3.2 Results

Table 3.2 summarizes the results for STLINC performance compared to the baselines. In the tables, N represents the horizon of the specification and D represents the maximum depth of temporal nesting; TO represents a timeout, which means the solver did not terminate despite running it for

²<https://github.com/parvkpr/MCTSTL>

Spec	N	D	Solve time (s)					Robustness				
			LinEnv			NonLinEnv		LinEnv			NonLinEnv	
			MICP	R-MICP	STLINC	R-MICP	STLINC	MICP	R-MICP	STLINC	R-MICP	STLINC
ϕ_1	40	0	0.845	2.698	0.891	0.890	1.464	0.500	0.500	0.500	0.430	0.572
ϕ_2	30	1	2.459	TO	0.402	12.674	0.892	0.491	-inf	0.491	-inf	0.594
ϕ_3	60	3	TO	TO	0.874	15.829	1.554	-inf	-inf	0.228	-inf	0.065
ϕ_4	40	1	0.318	0.330	0.629	1.049	1.131	0.494	0.500	0.500	0.470	0.364
ϕ_5	40	2	2.829	28.490	0.694	83.193	1.776	0.500	0.500	0.500	-inf	0.596

Table 3.2: STLINC Performance Benchmarking for LinEnv and NonLinEnv against standard MICP [18] and reduced MICP (R-MICP) [79].

30 minutes. In those cases, the solver’s output plan robustness is represented as -inf (which means no solution was found in the given time).

For LinEnv for all the specifications, our robustness values are comparable to the two techniques but our solve times are either lower or comparable to the baselines. Additionally, for specification ϕ_3 , which has the deepest temporal nesting, our method significantly outperforms both baseline methods that experience timeouts.

For NonLinEnv for ϕ_1 and ϕ_4 , the baseline encoding performs better in terms of solving time but STLINC only does slightly worse. However, for specification ϕ_2, ϕ_3 and ϕ_5 , STLINC significantly outperforms the baselines.

3.4 Summary

STL has been used extensively to describe complex properties over real valued signals such as state trajectories arising from continuous dynamical systems. For robotic planning, STL can be used to describe complex behaviors with concrete time deadlines such as those found in trajectory planning and task planning. However, when STL is used to represent these composite tasks, incremental planning becomes challenging. This issue is because STL semantics can encode the sequential nature of tasks but does not expose this structure to the planner. In such cases, the planners are forced to work with complex long-horizon specifications. To overcome this challenge, we propose a structural manipulation-based technique for the temporal decomposition of STL specifications, enabling the incremental fulfillment of these specifications. We show our method generates correct-by-construction trajectories that satisfy deeply nested specifications with long time horizons for which existing baseline STL planning techniques struggle. Our experiments suggest that our technique is more efficient for multistep tasks with deep temporal nesting, outperforming baselines by an order of magnitude.

Chapter 4

STLCG++: Efficient STL Evaluation

STL presents an attractive formalism to describe spatio-temporal specifications as it is designed to operate over *real-valued* time-series input rather than discrete propositions. In particular, STL’s quantitative semantics, or robustness formulas, which measure how well a given robot trajectory satisfies a requirement are a powerful feature. With some smoothing approximations in place, it becomes efficient and stable to *differentiate* STL robustness within gradient-based optimization methods—the key to many robot control and learning applications. As such, we have seen a growing interest in the inclusion of STL objectives/constraints in various optimization-based robotics problems utilizing gradient descent as a solution method, such as trajectory optimization [42, 103], deep learning [91, 95], and control synthesis [88, 131]. Recently, STLCG [82] was introduced as a general framework to encode any STL robustness formula as a computation graph and leveraged modern automatic differentiation (AD) libraries for evaluation and backpropagation. The STLCG (PyTorch) library made STL accessible to the broader robotics and deep learning communities, supporting a growing body of work that relies on gradient-based optimization with STL objectives/constraints [31, 73, 83, 89, 90, 121, 122, 142].

To construct the computation graph for any STL robustness formula, STLCG processes the time-series input *recurrently* (see Fig. 4.1 right), primarily inspired by how recurrent neural networks (RNNs) [56] process sequential data. While consistent with the semantics of STL robustness, recurrent processing leads to the forward and backward passes being comparatively slower than other non-recurrent operations—a widely observed drawback of RNNs. These sequential operations limit STLCG’s capability for efficiently handling long sequence lengths in offline and online settings, especially when combined with other demanding computations, e.g., running foundation models. We call this challenge the “breadth” challenge.

To overcome this challenge, we leverage lessons from the language modeling community. Attention-based neural architectures, such as transformers [119], have demonstrated superior performance in processing sequential data, particularly on GPU hardware. The key to the transformer architecture is the self-attention operation, which operates on all input values *simultaneously* rather than recurrently. Inspired by the masking mechanism in transformer architectures, we present STLCG++, a masking approach to evaluate and backpropagate through STL robustness for long sequences more efficiently than STLCG, a recurrent-based approach (see Fig. 4.1 left). Furthermore, STLCG++ enables the differentiability of STL robustness values with respect to time interval

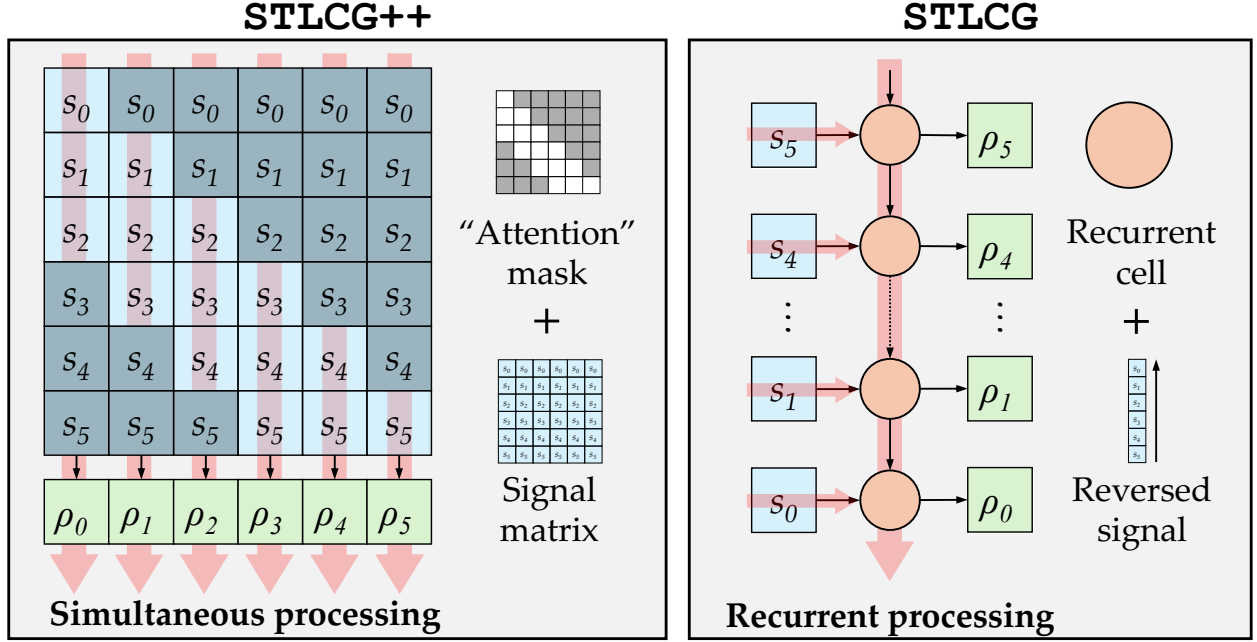


Figure 4.1: We propose **STLCG++** a masking approach to evaluating and backpropagating through signal temporal logic (STL) robustness formulas. The masking approach offers stronger computational, theoretical, and practical benefits compared to **STLCG**, a recurrent approach.

parameters via smooth masking. We provide *two* open-source **STLCG++** libraries, one in JAX and another in PyTorch, and demonstrate their usage via several robotics-related problems ranging from unsupervised learning, trajectory planning, and deep generative modeling. **STLCG++** opens new possibilities for using STL requirements in long-sequence contexts, especially for online computations, paving the way for further advancements in spatio-temporal behavior generation, control synthesis, and analysis for robotics applications.

4.1 Preliminaries

We provide a brief introduction to mathematical terminologies used in this chapter. See [82, 92] for a more in-depth description.

4.1.1 Signals and trajectories

Signals and subsignals. STL formulas are interpreted over one-dimensional *signals* $\mathbf{s} = (s_0, \dots, s_T)$, a sequence of scalars sampled at uniform timesteps Δt (i.e., continuous-time outputs sampled at finite time intervals) from any system of interest. Given a signal $\mathbf{s} = (s_0, \dots, s_T)$, a *subsignal* is a contiguous fragment of a signal. By default, we assume a subsignal will start at timestep t and end at the last timestep T . We denote such a subsignal by $\mathbf{s}_t = (s_t, \dots, s_T)$. If a subsignal ends at a dif-

ferent timestep from T , we denote it by $\mathbf{s}_t^K = (s_t, \dots, s_K)$. *Note:* The absence of a sub(super)script on $\mathbf{s}$ implies that the signal starts (ends) at timestep 0 (T).

States, trajectories, and subtrajectories. Given a system of interest, let $x_t \in \mathbb{R}^n$ denote the state at timestep t . Let $\mathbf{x} = (x_0, \dots, x_T)$ denote a sequence of states sampled at uniform time steps Δt . Similar to how subsignals are defined, we denote a subtrajectory by $\mathbf{x}_t = (x_t, \dots, x_T)$

From trajectories to signals. In this work, we focus specifically on signals computed from a robot’s trajectory. As we will see in the next section, core to any STL formula are predicates which are functions mapping state to a scalar value, $\mu : \mathbb{R}^n \rightarrow \mathbb{R}$, with $s_t = \mu(x_t)$, $t = 0, \dots, T$. A signal can represent, for example, a robot’s forward speed.

4.1.2 Smooth max / min approximation

It is often desirable to use STL robustness within gradient-based optimization (e.g., training deep neural networks or trajectory optimization). Since the robustness formulas consist of nested max / min operations, smooth approximations $\widetilde{\max}$ / $\widetilde{\min}$ are often used to aid numerical stability. Two typical smooth approximations are softmax and logsumexp.

$$\begin{aligned}\widetilde{\max}_{\text{soft}}(\mathbf{x}) &= \sum_{i=1}^N \frac{x_i \exp(\tau x_i)}{\sum_{j=1}^N \exp(\tau x_j)}, \\ \widetilde{\max}_{\text{LSE}}(\mathbf{x}) &= \frac{1}{\tau} \log \sum_{i=1}^N \exp(\tau x_i), \quad \widetilde{\min}(\mathbf{x}) = -\widetilde{\max}(-\mathbf{x})\end{aligned}$$

The smoothness can be controlled by a temperature parameter τ ; as $\tau \rightarrow \infty$, the smooth approximations approach the true max / min value. For deep learning, common practice is to anneal the temperature value so that a more accurate smooth max / min is used towards the end of the optimization routine. Soundness of the smooth approximation may be an important consideration, depending on the specific application. Recent works proposed sound approximations [94] which STLGG++ can easily support. However, we view these efforts as complementary to this work.

4.1.3 Robustness trace

As STL formulas are composed of nested operations, evaluating the robustness value necessitates first determining the robustness value for each subtrajectory and subformula. For instance, consider $F\phi$ where ϕ is the subformula. Evaluating $\rho(\mathbf{x}, F\phi)$ requires the robustness value of ϕ for all subtrajectories, i.e., $\rho(\mathbf{x}_t, \phi)$ for all t . Suppose that ϕ is another temporal formula, e.g., $G\phi$, then we would also require $\rho(\mathbf{x}_t, \phi)$ for all t . Subsequently, we introduce the concept of a *robustness trace*.

Definition 6 (Robustness trace). *Given a trajectory $\mathbf{x} = (x_0, \dots, x_T)$ and an STL formula ϕ , the robustness trace $\tau(\mathbf{x}, \phi)$ is a sequence of robustness values of ϕ for each subtrajectory of $\mathbf{x}$. That is,*

$$\tau(\mathbf{x}, \phi) = \rho(\mathbf{x}_0, \phi), \rho(\mathbf{x}_1, \phi), \dots, \rho(\mathbf{x}_T, \phi). \quad (4.1)$$

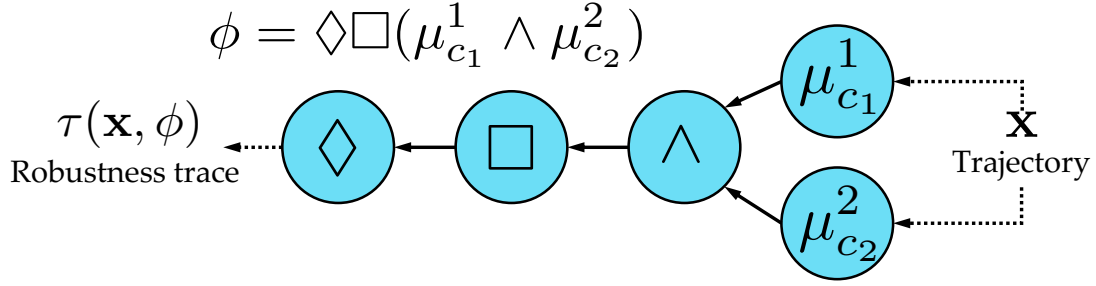


Figure 4.2: Illustration of the graph structure of an example STL formula. Input trajectories are first passed through the predicates and then each subformula according to the formula structure.

Computing the robustness trace of non-temporal operations is straightforward, as there is no need to loop through time, and for brevity, we omit that description in this chapter.

4.1.4 Recursive structure of STL formulas

To compute the robustness value of any STL formula ϕ with arbitrary formula depth¹, we must first calculate the robustness trace of its subformula(s), and so forth, since STL formulas are constructed recursively. Fig. 4.2 illustrates how the trajectories are passed through each operation of an STL formula to compute the robustness trace of each subformula. As such, we need to develop the “building blocks” for computing robustness corresponding to each STL operation. Then, by stacking these operations according to the formula structure, we can compute the robustness trace for *any* STL formula regardless of formula depth.

4.1.5 STLCG: Recurrent computation

We briefly outline the recurrent operations underlying STL_{CG}; for more details, we refer the reader to [82]. Illustrated in Fig. 4.1 (right), STL_{CG} utilizes the concept of dynamic programming to calculate the robustness trace. The input signal is processed *backward* in time, and a *hidden state* is maintained to store the information necessary for each recurrent operation at each time step. The choice of recurrent operation depends on the temporal STL formula (either a max or min). The size of the hidden state depends on the time interval of the temporal operator and is, at most, the length of the signal. Although the use of a hidden state to summarize past information helps reduce space complexity, the recurrent operation leads to slow evaluation and backpropagation due to sequential dependencies. Next, we present a masking-based approach that bypasses the sequential dependency, leading to faster computation times.

¹Assuming the formula consists of at least one temporal operation. Although STL formulas without temporal operators are possible, it is the temporal operations that make computing robustness challenging.

4.2 Recurrent v/s Masking based Approach

STLCG’s primary performance bottleneck stemmed from its use of recurrent operations for computing robustness of temporal operators. Illustrated in Fig. 4.1 (right), STLCG utilizes the concept of dynamic programming to calculate the robustness trace. The input signal is processed *backward* in time, and a *hidden state* is maintained to store the information necessary for each recurrent operation at each time step. The choice of recurrent operation depends on the temporal STL formula (either a max or min). The size of the hidden state depends on the time interval of the temporal operator and is, at most, the length of the signal. Although the use of a hidden state to summarize past information helps reduce space complexity, the recurrent operation leads to slow evaluation and backpropagation due to sequential dependencies. In contrast STLCG++ employs a *masking* approach to compute STL robustness traces. Fig. 4.1 illustrates the masking approach, highlighting the idea that each value of a robustness trace is computed *simultaneously*, rather than sequentially, as we saw with STLCG.

Mirroring the concept of attention masks in transformer architectures [119], we introduce a mask M to select relevant parts of a signal that can be later processed simultaneously rather than sequentially.

4.2.1 Notation and masking operation

Assume we have a 1D input signal $\mathbf{s}$ of length $T + 1$, and let K denote the number of time steps contained (inclusive) in the time interval $[a, b]$ given an STL temporal operator. A mask $\mathcal{M}$ is a 2 or 3 dimensional array, depending on the temporal operator, whose entries are either 1 or 0. A mask $\mathcal{M}$ is applied via element-wise multiplication to an array $\mathcal{S}$ of the same size. E.g., for a 2D mask, if $\mathcal{M}_{ij} = 0$, the corresponding entries $\mathcal{S}_{ij}$ will be ignored, or is *masked out*. We also replace the entries masked out with an arbitrary value, typically a large positive or negative value. Let $\mathcal{M} \odot_{\pm} \mathcal{S}$ denote the application of mask $\mathcal{M}$ on $\mathcal{S}$ and masked out by either a large positive ($\odot_{+}$) or negative ($\odot_{-}$) number.

4.2.2 Eventually and Always operations

For brevity, we describe the procedure for the Eventually operation but note that the procedure for the Always operation is almost identical except a min operation is used instead of max. Consider the formula $\psi = F_{[a,b]}\phi$, and let $\mathbf{s} = \tau(\mathbf{x}, \phi)$ denote the robustness trace of ϕ . Specifically, $s_t = \rho(\mathbf{x}_t, \phi)$. Here, ϕ is an STL subformula, typically representing a predicate or a more complex temporal logic expression. Then, we seek to find the robustness trace,

$$\tau(\mathbf{x}, F_{[a,b]}\phi) = \left\{ \max_{i \in [a,b]} s_{t+i} \right\}_{t=0}^T$$

Intuitively, we want to slide a time window $[a, b]$ along $\mathbf{s}$ and take the max of the values within the window. Rather than performing this set of operations sequentially, we **first** “unroll” the sequential operation—turning the 1D signal into a 2D array, **second**, we apply a 2D mask to mask

out the irrelevant entries dictated by the time window, and **third**, we use the max operation over the unmasked values.

Step 1: “Unrolling” the signal. First, we *pad* the end of the signal by the size of the time window K ; the reason will be apparent in Step 3. The padding value $\bar{s}$ can be set by the user, such as extending the last value of the signal, or setting it to a large negative number. Then, we turn the padded signal into a 2D array $\mathcal{S}$ by repeating the signal along the second dimension T times, resulting in a $(T + K) \times T$ 2D array. An example is provided in (4.2). Given $\mathbf{s}$ and padding value $\bar{s}$, the unrolled 2D array $\mathcal{S}$ is,

$$\mathcal{S} = \begin{array}{c} \overbrace{\begin{bmatrix} s_0 & s_0 & s_0 & \dots & s_0 & s_0 & s_0 \\ s_1 & s_1 & s_1 & \dots & s_1 & s_1 & s_1 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ s_T & s_T & s_T & \dots & s_T & s_T & s_T \\ \bar{s} & \bar{s} & \bar{s} & \dots & \bar{s} & \bar{s} & \bar{s} \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ \bar{s} & \bar{s} & \bar{s} & \dots & \bar{s} & \bar{s} & \bar{s} \end{bmatrix}}^{\text{\# timesteps}} \end{array}$$

Step 2: Construct 2D mask. We construct a mask $\mathcal{M}$ that will be applied to the unrolled signal $\mathcal{S}$. The mask $\mathcal{M}$ comprises of two sub-masks: (i) subsignal mask $\mathcal{M}_{\text{subsig}}$ and (ii) time interval mask $\mathcal{M}_{\text{time}}$.

Subsignal mask $\mathcal{M}_{\text{subsig}}$: This mask incrementally masks out the start of the signal as we move along the horizontal (second) dimension of $\mathcal{S}$. Essentially, $\mathcal{M}_{\text{subsig}}$ masks out the upper triangular region with an offset of one to exclude the diagonal entries. After applying the $\mathcal{M}_{\text{subsig}}$ on $\mathcal{S}$, the columns of the unmasked entries correspond to all the subsignals of $\mathbf{s}$. The entries masked out by $\mathcal{M}_{\text{subsig}}$ are denoted by the red shaded entries in (4.2).

Time interval mask $\mathcal{M}_{\text{time}}$: This mask masks out all the entries outside of time interval $[a, b]$ for each subsignal. $\mathcal{M}_{\text{time}}$ consists of an off-diagonal strip that masks out entries *before* the start of the interval (determined by a), and a lower triangular matrix with an offset that masks out entries *after* the interval (determined by b). The entries masked out by $\mathcal{M}_{\text{time}}$ are denoted by the blue shaded entries in (4.2).

Final mask $\mathcal{M}$: The final mask is $\mathcal{M} = \mathcal{M}_{\text{subsig}} + \mathcal{M}_{\text{time}}$, resulting in a mask that retains all the entries within the time interval $[a, b]$ for each sub-signal (unshaded entries in (4.2)).

Step 3: Apply 2D mask and max operation. Given $\mathcal{S}$ and $\mathcal{M}$ from the previous steps, we can compute $\mathcal{M} \odot_- \mathcal{S}$, similar to what is done in the mask attention mechanism in transformer architectures. Then, we apply the max operation along each column. Note the padding we did back in step 1 becomes relevant when the time interval exceeds the length of the subsignal. This is similar to the approach taken in [82] to handle incomplete signals. For the Always operation, min and $\mathcal{M} \odot_+ \mathcal{S}$ are used instead.

Example 1. Consider the STL formula $\psi = F_{[1,3]}(s > 0)$ and a signal with 8 time steps, $\mathbf{s} = [s_0, \dots, s_7]$. The corresponding unrolled 2D array is shown (4.2) with padding value $\bar{s}$, and the red and blue shaded entries denote the entries masked out by $\mathcal{M}_{\text{subsig}}$ and $\mathcal{M}_{\text{time}}$, respectively.

$$\begin{bmatrix}
s_0 & s_0 & s_0 & s_0 & s_0 & s_0 & s_0 & s_0 \\
s_1 & s_1 & s_1 & s_1 & s_1 & s_1 & s_1 & s_1 \\
s_2 & s_2 & s_2 & s_2 & s_2 & s_2 & s_2 & s_2 \\
s_3 & s_3 & s_3 & s_3 & s_3 & s_3 & s_3 & s_3 \\
s_4 & s_4 & s_4 & s_4 & s_4 & s_4 & s_4 & s_4 \\
s_5 & s_5 & s_5 & s_5 & s_5 & s_5 & s_5 & s_5 \\
s_6 & s_6 & s_6 & s_6 & s_6 & s_6 & s_6 & s_6 \\
s_7 & s_7 & s_7 & s_7 & s_7 & s_7 & s_7 & s_7 \\
\bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} \\
\bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} \\
\bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s} & \bar{s}
\end{bmatrix} \tag{4.2}$$

After filling the masked entries with a large negative value, and taking the max column-wise. Let $s = [0, 1, 2, 3, 4, 5, 6, 7]$ and $\bar{s} = 7$, then $\tau(s, \psi) = [3, 4, 5, 6, 7, 7, 7, 7]$. If we set $\bar{s} = -10^5$, then $\tau(s, \psi) = [3, 4, 5, 6, 7, -10^5, -10^5, -10^5]$.

4.2.3 Until operation

We apply a similar masking strategy for the Until operation, following a similar three-step process described in Sec. 4.2.2. The main differences are (i) the signal needs to be “unrolled” into three dimensions, and (ii) the operations in Step 3 correspond to the Until robustness formula.

Consider the formula $\phi \mathbf{U}_{[a,b]} \psi$ where ϕ and ψ are STL subformulas. Let $\mathbf{s}^\phi = \tau(\mathbf{x}, \phi)$ and $\mathbf{s}^\psi = \tau(\mathbf{x}, \psi)$ denote the robustness trace of ϕ and ψ respectively for trajectory $\mathbf{x}$. Specifically, $s_t^\phi = \rho(\mathbf{x}_t, \phi)$ and $s_t^\psi = \rho(\mathbf{x}_t, \psi)$. Then, we seek to find the robustness trace,

$$\tau(\mathbf{x}, \phi \mathbf{U}_{[a,b]} \psi) = \left\{ \max_{i \in [a,b]} \left\{ \min \left(\min_{\tau \in [0,i]} s_{t+\tau}^\phi, s_{t+i}^\psi \right) \right\} \right\}_{t=0}^T$$

Step 2: Construct 3D mask. In the Until robustness formula, we need $\min_{\tau \in [0,i]} s_{t+\tau}^\phi$ and s_{t+i}^ψ for $i \in [a,b]$ and for all t . Given i , notice that $\rho(\mathbf{x}_t, G_{[0,i]} \phi) = \min_{\tau \in [0,i]} s_{t+\tau}^\phi$ and $\rho(\mathbf{x}_t, G_{[i,i]} \psi) = s_{t+i}^\psi$ which we can compute given the procedure outlined in Sec. 4.2.2. However, the outer max requires us to consider all $i \in [a,b]$. Thus for each formula $G_{[0,i]} \phi$ and $G_{[i,i]} \psi$, we stack all the 2D masks for each value of $i \in [a,b]$ across the third dimension, resulting in two 3D masks, $\mathcal{M}^\phi$ and $\mathcal{M}^\psi$ of size $(T+K) \times T \times K$.

Step 3: Apply the Until robustness operations. We can now apply all the operations needed to compute the robustness trace for the Until operation. After Step 2, we have $\mathcal{S}^\phi$, $\mathcal{S}^\psi$, and the 3D masks $\mathcal{M}^\phi$ and $\mathcal{M}^\psi$. We can compute $\mathcal{M}^\phi \odot_+ \mathcal{S}^\phi$ and $\mathcal{M}^\psi \odot_+ \mathcal{S}^\psi$. Then, we can apply the following set of operations (where the first dimension = 1).

$$\begin{aligned}
& \text{Size } [2 \times T \times K] \\
S_1 &= \text{stack}_{\text{dim}=1} \left(\left[\min_{\text{dim}=1} \mathcal{M}^\phi \odot_+ \mathcal{S}^\phi, \min_{\text{dim}=1} \mathcal{M}^\psi \odot_+ \mathcal{S}^\psi \right] \right) \\
S_2 &= \min_{\text{dim}=1} S_1, \quad (S_2 \text{ has size } [T \times K]) \\
S_3 &= \max_{\text{dim}=2} S_2, \quad (S_3 \text{ has size } [T])
\end{aligned}$$

We see that this set of operations is relatively simple, a series of min and max operations along various dimensions, and these operations remain the same regardless of the choice of time intervals, unlike the recurrent approach.

4.2.4 Dependence on smooth approximations

Both masking and recurrent approaches output the correct robustness values when the true min / max operations and logsumexp approximation are used. However, if the softmax/softmin approximation is used with a recurrent approach, the output robustness and gradient values do not correctly reflect the correct values. This is because, when recurrently applying softmin/softmax, values applied earlier in the recurrence will be “softened” more than values applied later in the recurrence. Not only does nesting the softmin/softmax operations give a very poor approximation of the true min / max value, but it also artificially reduces the gradient for values passed earlier in the recurrence. With the logsumexp approximation, applying it recursively over multiple values is the same as applying it once over all the values. That is, $\widetilde{\max}_{\text{LSE}}([\widetilde{\max}_{\text{LSE}}(\mathbf{x}), y]) = \widetilde{\max}_{\text{LSE}}([\mathbf{x}, y])$ where as $\widetilde{\max}_{\text{soft}}([\widetilde{\max}_{\text{soft}}(\mathbf{x}), y]) \neq \widetilde{\max}_{\text{soft}}([\mathbf{x}, y])$. This issue is not observed when using the masking-based approach due to the way subsignals are processed without recurrence.

4.2.5 Practical benefits

STLCG++ offers several practical benefits over STLCG. We highlight several reasons why this is the case. **(i) Software design.** The recurrent approach required signals to be passed *backward* in time. This increased the complexity of software design and was not often intuitive for end users. With the masking approach, there is no need to reverse the signal. **(ii) Static graph structure.** With the recurrent approach, the choice of time interval would fundamentally change the dimension of hidden states, whereas the graph structure remains the same for the masking approach. Keeping the same graph structure, even if the intervals change, is essential in compilation. **(iii) Vectorization and just-in-time compilation.** Related to the second point above, the static graph structure afforded by the masking approach enables the ability to easily *vectorize* and *just-in-time compile* (JIT) the computation over not just various signal inputs but also various time intervals. The ability to vectorize and JIT over different time intervals can be particularly useful in formula mining applications where we may want to evaluate robustness formulas with different time intervals simultaneously. **(iv) Conformity to popular deep learning libraries.** By leveraging AD libraries that empower popular frameworks, STLCG++ becomes accessible to the broader community and seamlessly integrates into existing tools.

Table 4.1: Summary of STL specifications used for evaluation inspired by existing literature. I denotes the time interval over which the temporal operators apply.

Spec	Depth	Description	STL Formula
ϕ_1	0	Invariance	$\mathbf{F}(\varphi \wedge \psi)$
ϕ_2	1	Stabilization	$\mathbf{FG}(\varphi \wedge \psi)$
ϕ_3	1	Strict ordering	$\varphi \mathbf{U} \psi$
ϕ_4	3	Sequenced visit pattern	$\mathbf{F}_I(\varphi_4 \wedge \psi_4 \wedge \mathbf{F}_I(\varphi_3 \wedge \psi_3 \wedge \mathbf{F}_I(\varphi_2 \wedge \psi_2 \wedge \mathbf{F}_I(\varphi \wedge \psi))))$
ϕ_5	2	Sequenced visit + stabilization	$\mathbf{F}_I((\varphi_2 \wedge \psi_2) \wedge \mathbf{F}_I(\mathbf{G}_I(\varphi \wedge \psi)))$
ϕ_6	0	Reach regions in any order	$\mathbf{F}_I(\varphi_9 \wedge \psi_9) \wedge \dots \wedge \mathbf{F}_I(\varphi_0 \wedge \psi_0)$

4.3 Evaluation

4.3.1 Computational Properties of STLGG++

We analyze the computational properties of the approaches STLGG++ (masking-based) and STLGG (recurrent-based) by measuring the computation time required to compute robustness values and their gradient. We seek to answer the following research questions.

RQ1: Does STLGG++ compute robustness traces faster than STLGG as measured by median computation time?

RQ2: How does STLGG++’s computation time scale with sequence length compared to STLGG?

We perform experiments on CPU and GPU. Since STLGG++ (masking) involves large matrix computations, we anticipate STLGG++ to scale favorably on a GPU. As STLGG (recurrent) utilizes a recurrent structure, it scales with sequence length. Tab. 4.1 describes six different STL formulas with varying complexities [52] that we test on. We evaluate computation time for increasing signal lengths up to $T = 512$ time steps with a batch size of 8. We present our results in Fig. 4.3 and Tab. 4.2 and make the following observations.

CPU backend. We observe that STLGG++ generally achieves lower computation times than STLGG. The exception is in ϕ_3 , where STLGG++ is slower than STLGG for longer sequences. This is because the space complexity for the Until operation is $\mathcal{O}(T^3)$. In Tab. 4.2, we see that STLGG++ on JAX struggled with ϕ_3 for long sequence lengths, but was fine with PyTorch. We hypothesize that it is due to how JAX allocates memory, especially during just-in-time compilations. Additionally, both libraries see increasing computation time with sequence length. For STLGG the computation time increase is expected since the time complexity is $\mathcal{O}(T)$. For STLGG++, the increased computation time can be explained by the space complexity $\mathcal{O}(T^2)$ (except for ϕ_3), and such computations are not handled as efficiently on a CPU.

GPU backend. STLGG++ outperforms STLGG for all formulas. Additionally, STLGG++ exhibits essentially constant computation time except for ϕ_3 , which has $\mathcal{O}(T^3)$ space complexity. From Tab. 4.2, we see that **STLGG++ on GPU provides around 95% and 85% improvement on JAX and PyTorch, respectively**. For STLGG, moving from CPU to GPU gives virtually the same scaling, with no significant improvement/reduction in computation times.

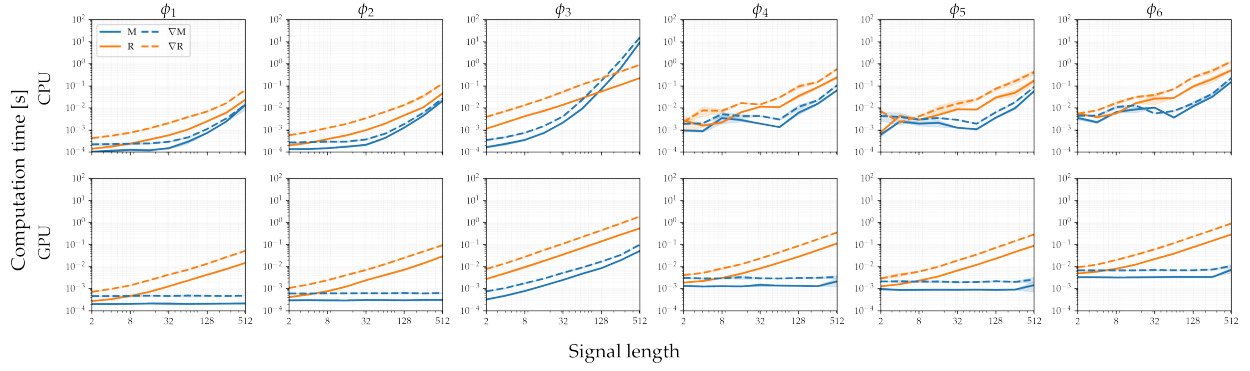


Figure 4.3: Comparison of computation time using PyTorch for masking (M, blue) and recurrent (R, orange) approach on CPU (top row) and GPU (bottom row) as signal length increases, across six STL formulas. Solid lines denote robustness computation; dashed lines denote gradient evaluation.

Table 4.2: Relative computation time of masking approach compared to recurrent approach. Median value across different signal lengths. Lower is better.

Device	CPU		GPU	
Formula	JAX + JIT		PyTorch	
ϕ_1	−37.99%	−93.43%	−76.05%	−85.76%
ϕ_2	−51.51%	−96.65%	−82.50%	−89.03%
ϕ_3	822.01%	−91.31%	−88.22%	−94.15%
ϕ_4	−68.22%	−96.00%	−73.48%	−77.91%
ϕ_5	−68.58%	−95.32%	−67.91%	−80.86%
ϕ_6	−72.93%	−98.11%	−54.14%	−84.36%

4.4 Smoothing time intervals

By using the masking operation to capture parts of the signal within the specified time interval, we can build a smooth approximation of the mask and *differentiate* robustness values with respect to the parameters of the mask that determine which values are selected. As such, it becomes possible to perform gradient descent over time interval parameters to find, for example, a time interval that best explains time-series data. We showcase two examples in Sec. 4.5 utilizing this new capability. Notably, this differentiation was not possible with `STLCG` since the choice of time intervals impacted the size of the hidden state used within the recurrent computations.

Rather than using a mask with 0 and 1’s as described in Sec. 4.2.2 and 4.2.3, we use the following smooth mask approximation. For a sequence length of T , we have,

$$\widetilde{\text{mask}}(i; a, b, c, \varepsilon) = \max(\sigma(c(i - aT)) - \sigma(c(i - bT)) - \varepsilon, 0), \quad (4.3)$$

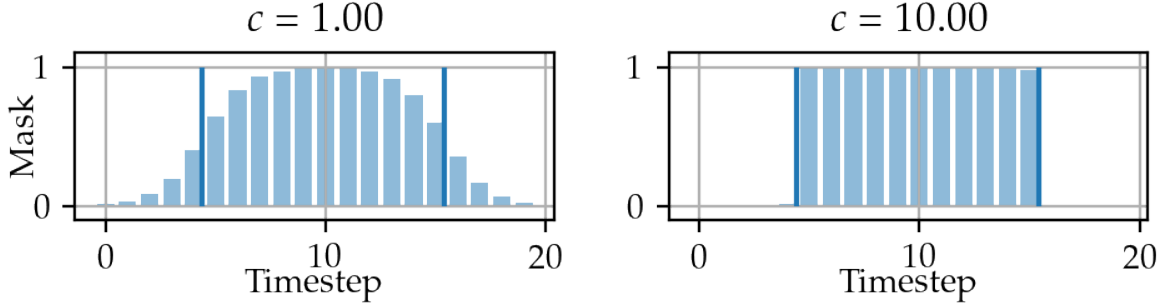


Figure 4.4: Smooth mask for various values of smoothing parameter c .

where i is the time index, $\sigma(x)$ is the sigmoid function, the parameters $0 \leq a < b \leq 1$ denote the fraction along the signal of the start and end of the time interval, c denotes the mask smoothing parameter, and ε denotes a user-defined tolerance. Fig. 4.4 shows a visualization of the smooth mask and how the smoothing parameter c affects the mask.

When optimizing over time interval parameters, we can anneal the smooth mask parameter c to help with convergence to a (local) optima on a generally nonlinear loss landscape (see Sec. 4.5.3 for more details). As noted before, a benefit of the masking approach is that the operations can be easily vectorized. We can simultaneously evaluate the robustness for multiple time intervals and perform gradient descent on multiple values. As such, we can envision a use case where we perform a coarse global search via sampling and then a local refinement via gradient descent.

For reference, using the vectorized mapping function in JAX+JIT, the computation time to evaluate 90,000 different values for (a, b) with a signal of length $T = 20$ for the formula $G_{[aT, bT]}(s > 0)$ is $1.87 \pm 0.0164\text{ms}$ on a M2 MacBookPro. This averages to around $20.78\mu\text{s}$ per time interval value. In contrast, (sequentially) searching over all possible valid time intervals for a signal length of 20 (i.e., 190 intervals with integer interval limits) using the recurrent approach takes about $5.25 \pm 0.0497\text{s}$, or roughly 27.6ms each evaluation. This means that **STLCG++ offers more than 1000 $\times$ improvement in computation time** when evaluating STL robustness over multiple time interval values.

4.5 Robotics-related applications

We showcase a variety of use cases that demonstrate the computational advantages of the masking approach. We developed *two* Python STLCG++ libraries, one in JAX [22] and another in PyTorch [105] and discuss how STLCG++ opens up new possibilities for robot planning and control.

4.5.1 Trajectory planning with suboptimal STL specifications.

Trajectory optimization is essential in robotic systems, enabling precise navigation and task execution under complex constraints. These trajectories must balance user-defined goals with feasibility and efficiency. Since goal misspecification can lead to erroneous behavior, infeasibility, or sub-par performance, we show how suboptimal goals (captured through STL specifications) can be refined in addition to solving for an optimal trajectory. Consider a trajectory optimization problem

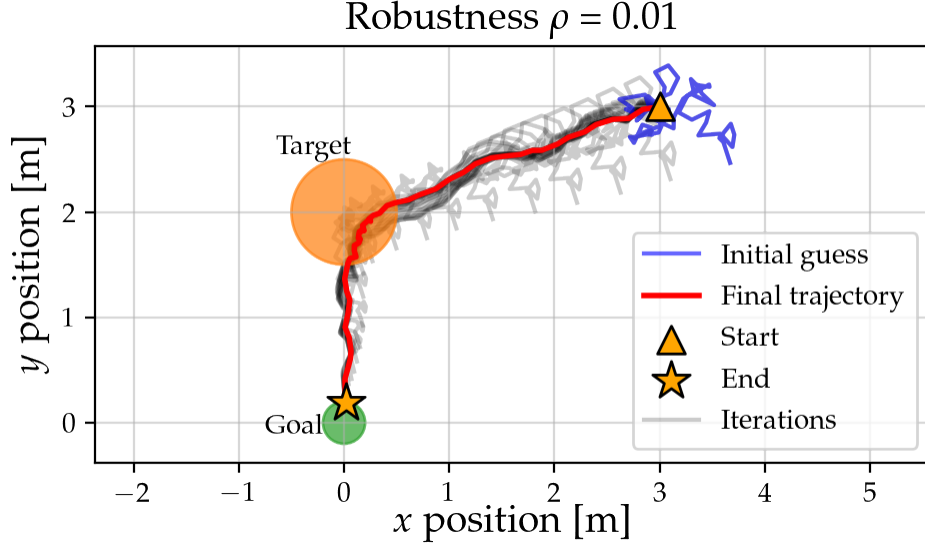


Figure 4.5: Trajectory with optimized time interval in orange target region.

where we would like to reach a goal region $\bullet$ while visiting a target region $\bullet$ within a fixed time horizon. Additionally, the visit duration of $\bullet$ is to be maximized. Consider a specification $\phi = G_{[aT, bT]}(\text{inside } \bullet) \wedge F(\text{inside } \bullet)$ which specifies a time window that the trajectory should be inside $\bullet$. We cast this trajectory planning problem as an unconstrained optimization problem,

$$\begin{aligned} \min_{\mathbf{u}, a, b} & \gamma_1 J_{\text{STL}}(\mathbf{x}) + \gamma_2 J_I(a, b) + \gamma_3 J_{\text{lim}}(\mathbf{u}) + \gamma_4 J_{\text{eff}}(\mathbf{u}), \\ J_{\text{STL}}(\mathbf{x}) &= \text{ReLU}(-\rho(\mathbf{x}, \phi)), \quad J_I(a, b) = \exp(2(\tilde{I} - b + a)), \\ J_{\text{lim}}(\mathbf{u}) &= \frac{1}{T} \sum_{t=0}^{T-1} \text{ReLU}(\|\mathbf{u}_t\|_2 - \bar{u}), \quad J_{\text{eff}}(\mathbf{u}) = \frac{1}{T} \sum_{t=0}^{T-1} \|\mathbf{u}_t\|_2^2, \end{aligned}$$

where $\mathbf{x}$ is the state trajectory from executing $\mathbf{u}$ with single integrator discrete-time dynamics. We use a timestep of $\Delta t = 0.1$, $T = 51$, $\bar{u} = 2$ is the system's maximum control limits, and $\tilde{I} = 0.2$ is a nominal (normalized) time interval size that we would like to improve upon. We randomly initialized the control inputs and $(a, b)_{\text{init}} = (0.14, 0.82)$. Using coefficients $\gamma_1 = 1.1$, $\gamma_2 = 0.05$, $\gamma_3 = 2$, $\gamma_4 = 0.5$, the resulting solution is shown in Fig. 4.5 with final values $(a, b)_{\text{final}} = (0.37, 0.84)$. The final optimized STL formula is $\phi = G_{[20, 43]}(\text{inside } \bullet) \wedge F(\text{inside } \bullet)$. Using JAX, each gradient step took about $92.3 \pm 3.99 \mu\text{s}$ on an M2 MacBookPro. With unconstrained multiobjective optimization, selecting objective weights that achieve desirable behavior can be tedious. With STLGG++, we can perform gradient descent computation over various coefficient values simultaneously and then select the best one.

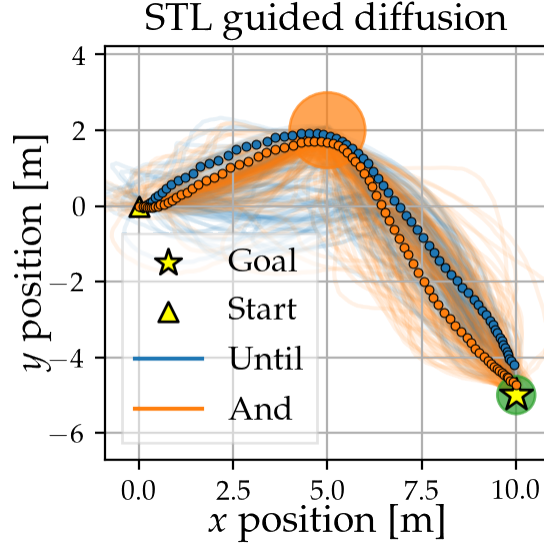


Figure 4.6: Generated trajectories using STL-guided DDIM.

4.5.2 Deep generative modeling: STL-guided diffusion policies

We demonstrate the use of STL specifications for guided diffusion models [116], a recent type of deep generative models that present a promising approach for robot policy learning and behavior generation [63]. We build upon [98], which uses Control Barrier and Lyapunov functions (CBFs and CLFs) to guide the denoising process of a diffusion model for safe control sequence generation. In this application, we use STL robustness in place of CBFs and CLFs as the guidance function. We trained a Denoising Diffusion Implicit Model (DDIM) model [116], generating trajectories with 80 time steps.

We used 200 denoising steps and a batch size of 128. We generated trajectory samples using two different STL guidance functions,

$$\begin{aligned}\phi_U &= \text{FG}_{[0,10]}(\text{inside } \bullet) \cup \text{FG}_{[0,5]}(\text{inside } \bullet) \\ \phi_\wedge &= \text{FG}_{[0,10]}(\text{inside } \bullet) \wedge \text{F}_{[60,80]} \text{G}_{[0,5]}(\text{inside } \bullet).\end{aligned}$$

The resulting trajectories are shown in Fig. 4.6. Although guided diffusion does not guarantee that the STL specifications are strictly satisfied, we observe a satisfaction rate of 52.34%, 62.50% for ϕ_U and $\phi_\wedge$, respectively. This shows how STLGG++ can enhance diffusion policies to promote safer behavior generation.

4.5.3 Machine learning: STL parameter mining from data

Formal STL specifications play a crucial role in building robust robot systems, contributing to controller synthesis [70], fault localization [17], and anomaly detection and resolution [65]. However, specifications are not often readily available, making the task of mining specifications from data a vital problem to study [52]. In this example, we perform STL specification mining from data.

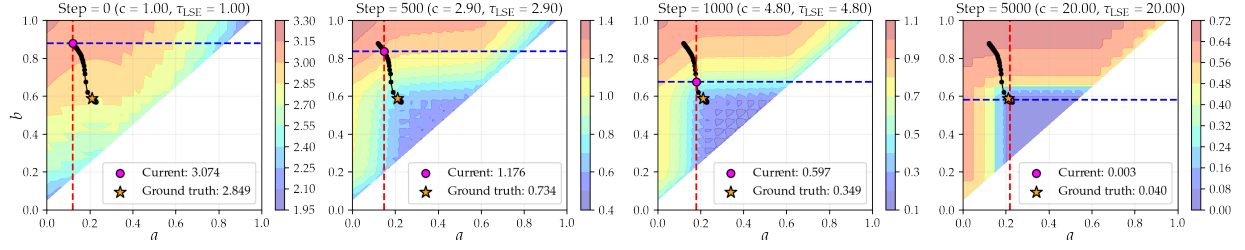


Figure 4.7: The masking approach enables a smooth approximation of time intervals, enabling the use of gradient descent over time interval parameters. The contour plots show the loss landscape given by (4.4) as a function of a, b , the normalized time interval limits. The solutions at gradient descent steps 0, 500, 1000, and 5000 are shown. Note that the smooth time mask parameter and approximate max / min temperature values are annealed over the steps.

Specifically, we leverage the differentiability and vectorization over smooth time intervals to find a time interval that best fits the observed data.

Consider a dataset of (noisy) signals with $T = 20$ timesteps. The signals have value 1 between the normalized time interval of $(a, b) = (0.23, 0.59)$ and zero elsewhere. Some noise is added around (a, b) and to the signal itself. The goal is to learn the largest time interval $[\bar{a}, \bar{b}]$ such that the signals from the dataset satisfy $\phi = G_{[\bar{a}, \bar{b}]}(s > 0)$. We frame this STL mining problem as an optimization problem,

$$\min_{0 \leq a < b \leq 1} \frac{1}{N} \sum_{\mathbf{s} \in \mathcal{D}} \max(-\rho(\mathbf{s}, G_{[a, b]}(s > 0)), 0) + \gamma(a - b) \quad (4.4)$$

where γ is a coefficient on the term $a - b$ that encourages the interval to be larger. We solve (4.4) via gradient descent, using the smooth time interval mask discussed in Sec. 4.4. Specifically, we annealed the time interval mask scaling factor and the temperature with a sigmoid schedule, and performed 5000 gradient steps with a step size of 10^{-2} . To ensure $0 \leq a < b \leq 1$, we passed them through a sigmoid function first. Fig. 4.7 shows a few snapshots of the gradient steps and the loss landscape as the scaling parameter and temperature increase. We can observe that our solution converges to the global optimum and is consistent with the ground truth values (subject to the injected noise). To give a sense of the computation time, using JAX on a M2 MacBookPro, it took about $122 \pm 0.479 \mu\text{s}$ per gradient step.

4.6 Summary

This chapter highlights STL_{CG++}, a masking-based approach for computing STL robustness using automatic differentiation libraries. STL_{CG++} mimics the operations that underpin transformer architectures, and outperforms the proposed masking approach over STL_{CG}, which uses a recurrent approach. STL_{CG++} offers significant computational advantages over STL_{CG}, thus presenting new and exciting opportunities for incorporating STL specifications into various online robot planning and control tasks that require fast computation and inference speeds.

Chapter 5

ETL: Extending STL Beyond State-Based Constraints

5.1 Introduction

Modern autonomous systems—from self-driving vehicles to robotic manipulators—increasingly rely on learned representations for perception, prediction, and decision making [47, 50, 62, 75, 133, 143]. These learned representations allow autonomous systems to overcome the challenges of operating on explicit state space representations such as object poses, velocities, etc. that often require explicit state estimation pipelines and auxiliary localization modules. We refer to such systems that operate on learned representations as perception-based systems. Perception-based systems map high-dimensional sensor streams such as images, video, or lidar to compact latent representations $z \in \mathcal{Z} \subseteq \mathbb{R}^d$, which are then consumed by downstream policies, planners, or world models [15, 101]. Although these latent representations are more compact than raw sensory observations, they remain substantially higher-dimensional and less interpretable than traditional state-space representations. As a result, specifying and assuring safe behavior for such systems remains a central challenge in trustworthy autonomy, particularly as they are increasingly deployed in open-world environments.

A standard approach to achieve high assurance for autonomous systems involves writing formal specifications and using techniques such as verification [29] and run-time monitoring [92] to check whether system executions satisfy those specifications [114]. Among these, runtime monitoring has attracted particular interest because it can be deployed online without requiring exhaustive offline analysis of all possible behaviors. Specification runtime monitors observe a system online and raise an alert when a safety specification is violated [29]. This ability to provide lightweight, online assurance has led to successful applications in domains such as autonomous vehicles [112], drone swarms [44], and robotic manipulators [84]. However, runtime monitoring relies on the availability of *formal specifications* that capture the desired properties of a system. For systems with low-dimensional state representations, temporal logics such as STL provides a natural formalism to express these properties over predicates. Here, a *predicate* denotes a simple state-based condition that can be evaluated as true or false at each time step, such as whether

position, velocity, or force satisfies a given threshold. Such predicates can be written directly as arithmetic conditions over interpretable signals and composed using temporal operators to express richer behavioral requirements.

For perception-based systems, writing such formal specifications over learned representations remains an open challenge [114]. For these systems, low-dimensional state representations are often not available or require specialized localization and mapping modules that can be inaccurate and computationally intensive. Hence, translating a property such as “the robot is near the obstacle” or “the gripper is holding the object” into logical predicates over system variables requires either an additional perception module, such as a classifier or detector, or a handcrafted feature extractor tailored to the task [55]. Both of these design decisions introduce new sources of brittleness, calibration error, and domain dependence. Fundamentally, they create a mismatch between modern learned perception systems and formal monitoring frameworks, which still assume access to state-based propositions.

In this work, we propose a new approach for formally specifying the behavioral properties of perception-based autonomous systems. The key idea is to employ embeddings—pretrained vector representations of observations—as a first-class concept in a specification notation, and express a property in terms of distances between a target embedding (an ideal representation of the world for the system to reach or avoid, given as part of the specification) and an observed embedding (a representation observed and generated through a sensor during system execution). The key insight is that *pretrained encoders already embed semantic proximity in geometry* [102, 108]: observations of semantically similar scenes map to nearby vectors in latent space. This makes perceptual properties directly expressible as geometric ones; for instance, “being near the obstacle” can be represented as “ $\|z_t - z_{\text{obstacle}}\|_2$ is small,” where z_{obstacle} is the encoder representation of a reference image of that obstacle. This perspective is especially useful for tasks whose correctness depends on high-level perceptual concepts that are difficult to localize or enumerate symbolically. For example, consider a rescue robot tasked with satisfying the requirement: “Locate a potential victim while avoiding areas with fire.” Such a property is difficult to specify in existing formal languages, since it depends on perception and the exact locations and appearances of relevant objects may be unknown or change over time. In an embedding-based formulation, however, this task can instead be expressed as reaching states whose observations closely resemble a potential victim while avoiding states whose observations resemble fire. Such specifications can then be used for run-time monitoring to help ensure that the system satisfies the desired property.

Although this idea is conceptually simple, making it useful for formal specification is not straightforward. First, there is the question of how target embeddings are generated: they can come from a reference image, a demonstration, or a set of both, and different choices can induce meaningfully different predicates. Additionally, embeddings are learned, continuous, and model-dependent representations, and geometric proximity is not guaranteed to align perfectly with the logical distinctions required for monitoring. A central challenge, therefore, is to turn embedding-space similarity into a well-defined specification primitive: one must decide which geometric relationships correspond to predicate satisfaction, how to calibrate decision thresholds, and how these predicate evaluations are composed for multiple requirements. These issues make embedding-based specification substantially more challenging than simply reusing learned features inside an

existing monitor.

We propose *Embedding Temporal Logic* (ETL), a temporal logic for specifying the behavior of perception-based systems directly over learned representations. ETL replaces Boolean state-based propositions with *embedding predicates*, which express semantic conditions through geometric relations in latent space. By composing embedding predicates with standard temporal operators, ETL can express temporally extended perceptual requirements [21].

5.2 Background

5.2.1 Pretrained Vision Encoders

Recent advances in representation learning have produced pretrained vision encoders that are sufficiently expressive to serve as general-purpose perceptual representations across a wide range of visual domains [102]. As a result, these models provide a practical foundation for extending temporal logic beyond low-dimensional state space representations. Pretrained vision encoders such as CLIP [108] and DINOv2 [102] provide a shared embedding space in which perceptual similarity can be measured, making them a natural choice for defining specification predicates over observations. Trained on large-scale image corpora, these encoders produce rich and transferable representations whose embedding spaces exhibit a useful property for monitoring: perceptually similar observations (e.g., depicting the same object or scene state) cluster together, while semantically distinct observations are well-separated [102]. While general-purpose encoders capture coarse perceptual similarity, task-conditioned world models can further shape embedding geometry toward distinctions relevant for downstream tasks.

5.2.2 Specification based Runtime Monitoring

Given a logical specification φ encoding a desired system property, a runtime monitor is a function $M(\sigma_{\leq t}, \varphi)$ that is evaluated online at each timestep t to assess whether the observed execution trace $\sigma_{\leq t}$ satisfies the specification [16, 92]. In its simplest form, the monitor produces Boolean outputs in $\{-1, +1\}$, where $+1$ indicates that the trace is consistent with φ so far, and -1 indicates a violation. A correct monitor must raise a violation (-1) exactly when the observed trace can no longer satisfy φ under any continuation.

In practice, runtime monitoring often goes beyond Boolean verdicts and employs quantitative semantics, such as robustness measures in temporal logic, which provide a real-valued signal indicating how strongly a trace satisfies or violates the specification. These quantitative monitors are particularly useful in continuous and stochastic systems, as they enable graded feedback and can be integrated with optimization or control algorithms for real-time decision making.

5.2.3 Conformal Prediction

Conformal prediction [123] is a distribution-free calibration framework that turns a held-out calibration set into a finite-sample statistical guarantee under only an *exchangeability* assumption. At

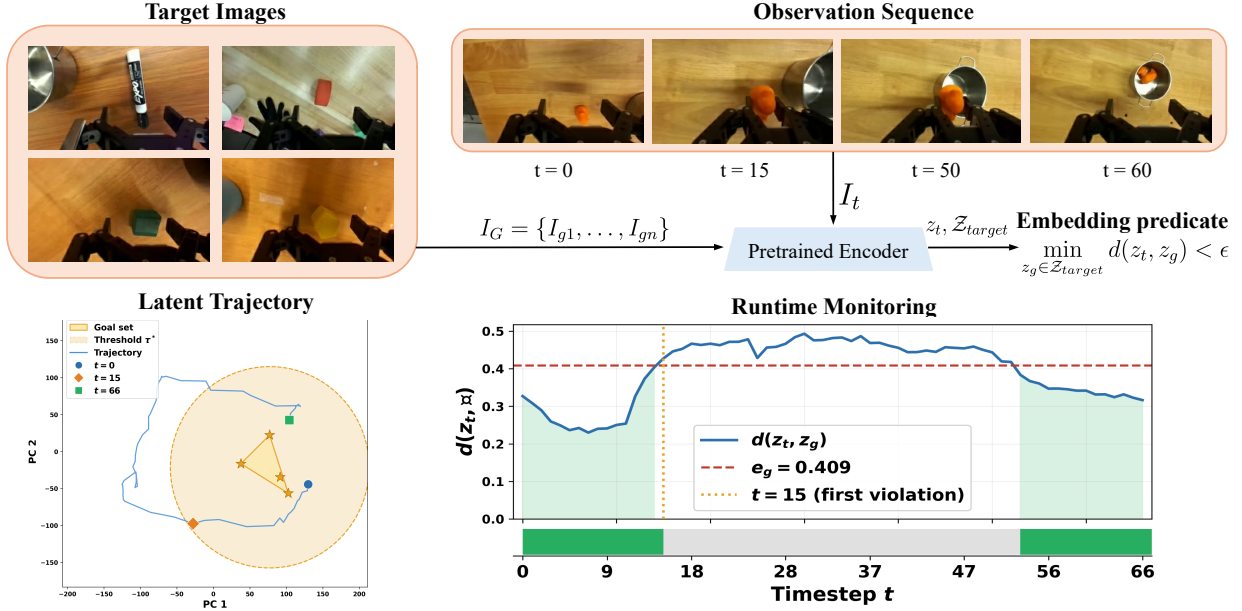


Figure 5.1: Overview of embedding-based runtime monitoring. **Top:** Target observations $O_G = \{o_{g1}, \dots, o_{gn}\}$ and the online observation o_t are encoded by a pretrained vision encoder into target embeddings z_G and the current embedding z_t . An embedding predicate then evaluates whether z_t reaches the target set. **Bottom left:** The embedding trajectory is projected onto its first two principal components, showing its evolution relative to the target embeddings. **Bottom right:** Runtime monitoring computes the distance from z_t to z_G over time, thresholds it using ϵ , and evaluates the temporal logic specification.

at a high level, it is a method for calibrating a model so that its predictions on unseen examples come with a reliability guarantee. It involves splitting the data into two parts: one part is used to train the model, and the other is held out for calibration. The trained model is then evaluated on the calibration split, and the resulting prediction errors are recorded. These errors are used to choose a cutoff that serves as a buffer around future predictions. In this way, the model's outputs are adjusted using the size of past errors, making the final prediction more reliable on new data. We employ conformal prediction theory for calibrating thresholds for ETL predicates in Section 5.5.

5.3 Embedding Temporal Logic

In this section, we first outline the syntax and semantics of ETL. Then, we highlight how ETL specification based monitors can be defined for runtime monitoring.

5.3.1 Syntax and Semantics

In our approach, a perception-based system is assumed to make an observation about the real world through a sensor (e.g., camera) at each step in its execution. This observation is then passed

through an encoder that translates the observation into an embedding. We formally model such a system as an *Embedding Temporal Structure*.

Definition 7 (Embedding Temporal Structure). *An Embedding Temporal Structure is a tuple*

$$\mathcal{M} \equiv (\mathcal{S}, \mathcal{O}, \mathcal{Z}, \phi_{obs}, \psi_{enc}, \mathcal{D}_{\mathcal{Z}}, AP_z)$$

where

- $\mathcal{S}$ is the set of ground truth states
- $\mathcal{O}$ is the set of observation states
- $\mathcal{Z}$ is the set of embeddings
- $\phi_{obs} : \mathcal{S} \rightarrow \mathcal{O}$ is the observation function that maps the ground truth states to the states that can be observed with the given sensor(s)
- $\psi_{enc} : \mathcal{O} \rightarrow \mathcal{Z}$ is the embedding function that converts an observation into an embedding
- $\mathcal{D}_{\mathcal{Z}}$ is a set of admissible distance or similarity functions, where each $d \in \mathcal{D}_{\mathcal{Z}}$ computes a distance between a pair of embeddings; i.e., $d : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathbb{R}_{\geq 0}$.
- AP_z is the set of predicates over embeddings, specified in Definition 11

Conceptually, z is an approximation of a latent variable; i.e., the state of the world that is only indirectly observable by the system. The *organization* of latent representations of observations within the embedding space $\mathcal{Z}$ is induced by the encoder ψ_{enc} and the downstream models that subsequently consume this embedding for further tasks (e.g., object identification and prediction) and shape the embedding organization as part of their training. This organization of embeddings in turn has a pronounced effect on the distances between embeddings.

To reason about a system's temporal behavior, we must consider not only individual states but sequences of states over time. We therefore define an execution of the system and the corresponding embedding-space representation induced by the observation and encoder mappings.

Definition 8 (Execution). *An execution of the system $\mathcal{M}$ is a finite or infinite sequence of states*

$$\varsigma = s_0, s_1, s_2, \dots$$

such that each state s_t belongs to the state space $\mathcal{S}$ for every time index $t \in \mathbb{N}$.

Each state in an execution can be mapped through ϕ_{obs} and ψ_{enc} , yielding a corresponding representation in embedding space.

Definition 9 (Representation Map). *The mapping from the ground truth states to embeddings is the representation map $\eta \equiv \psi_{enc} \circ \phi_{obs} : \mathcal{S} \rightarrow \mathcal{Z}$.*

Definition 10 (Trace). *Given an execution $\varsigma = (s_i)_{i \in \mathbb{N}}$, the associated trace is the sequence $\sigma = (z_i)_{i \in \mathbb{N}}$, where $z_i = \eta(s_i)$ for all $i \in \mathbb{N}$.*

Embedding Predicates Unlike classical temporal logics where atomic propositions are Boolean predicates over S , ETL predicates are defined over the embedding space $\mathcal{Z}$.

Definition 11 (Embedding Predicate). *An embedding predicate $ap \in AP_z$ is a tuple*

$$ap \equiv (\mathcal{Z}_{target}, d, \varepsilon, \bowtie, a)$$

where:

- $\mathcal{Z}_{target} \subseteq \mathcal{Z}$ is a set of target embeddings,
- $d \in D_{\mathcal{Z}}$, where $d : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathbb{R}_{\geq 0}$, is a distance function,
- $\varepsilon \in \mathbb{R}_{\geq 0}$ is a threshold,
- $\bowtie \in \{<, \leq, >, \geq\}$ is a comparison operator,
- a is an aggregation operator (e.g., min, max).

The set of target embeddings identifies inputs that sensors should either reach or avoid in order for the system to be considered safe. Additional details for target embeddings are discussed in Section 5.4.1.

We now define how an Embedding Predicate can be evaluated.

Definition 12 (Predicate Satisfaction). *For a given embedding predicate $ap \in AP_{\mathcal{Z}}$ and embedding z , the evaluation of an embedding predicate is defined as:*

$$\delta_{ap}(z) = a(\{d(z, z_g) \mid z_g \in \mathcal{Z}_{target}\})$$

Then, the predicate satisfaction is the Boolean value computed by $\delta_{ap}(z) \bowtie \varepsilon$.

Example 1. Consider an embedding predicate ap as illustrated in Figure 5.1, where the target images depict a desired visual concept (e.g., object not grasped) and are encoded into a target set $\mathcal{Z}_{target} = \{z_1^g, \dots, z_m^g\}$. Given the embedding z_t of the current observation, predicate evaluation measures the distance between z_t and the target set. Let d be the L2 distance, $a = \min$, and $\bowtie = \leq$. Then,

$$\delta_{ap}(z_t) = \min_{z_g \in \mathcal{Z}_{target}} d(z_t, z_g).$$

The predicate is satisfied if $\delta_{ap}(z_t) \leq \varepsilon$, meaning that the current observation embedding lies sufficiently close to at least one target embedding, indicating that the desired visual concept is present.

Now that we have defined embedding predicates, we next define the syntax and semantics of ETL to reason over sequences of embeddings over time.

Definition 13 (ETL Syntax). *The ETL syntax of formula φ is recursively defined as follows:*

$$\varphi ::= ap \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 U \varphi_2$$

As in LTL, the until operator U can be used to express the *eventually* (**F**) and *always* (**G**): $\mathbf{F}\varphi = \text{True} U \varphi$ and $\mathbf{G}\varphi = \neg \mathbf{F}\neg \varphi$.

Definition 14 (ETL Semantics). *For a given trace σ , the ETL semantics for timestep i are defined similarly to those of LTL, but over embedding traces:*

$$\begin{aligned} \sigma, i \models ap &\iff \delta_{ap}(z_i) \bowtie \varepsilon \\ \sigma, i \models \neg \varphi &\iff \sigma, i \not\models \varphi \\ \sigma, i \models \varphi_1 \wedge \varphi_2 &\iff \sigma, i \models \varphi_1 \text{ and } \sigma, i \models \varphi_2 \\ \sigma, i \models \varphi_1 U \varphi_2 &\iff \exists j \geq i \text{ such that } \sigma, j \models \varphi_2 \\ &\quad \text{and } \forall k \in [i, j), \sigma, k \models \varphi_1 \end{aligned}$$

Robustness of ETL Certain extensions of LTL, such as *signal temporal logic* (STL) [92], allow for a quantitative notion of satisfaction — called *robustness* [37] — that represents the degree to which the system satisfies or violates a specification. To enable ETL to be used for quantitative monitoring, we also introduce a notion of robustness for ETL.

Definition 15 (ETL Robustness). *For a trace σ , the robustness of an ETL formula at timestep i is defined as follows:*

$$\begin{aligned}\rho(ap, \sigma, i, \mathbf{b}) &= \begin{cases} \varepsilon - \delta_{ap}(z_i) & \bowtie \in \{\leq, <\} \\ \delta_{ap}(z_i) - \varepsilon & \bowtie \in \{\geq, >\} \end{cases} \\ \rho(\neg\varphi, \sigma, i, \mathbf{b}) &= -\rho(\varphi, \sigma, i, \mathbf{b}) \\ \rho(\varphi_1 \wedge \varphi_2, \sigma, i, \mathbf{b}) &= \min(\rho(\varphi_1, \sigma, i, \mathbf{b}), \rho(\varphi_2, \sigma, i, \mathbf{b})) \\ \rho(G\varphi, \sigma, i, \mathbf{b}) &= \inf_{k \in [i, \mathbf{b}]} \rho(\varphi, \sigma, k, \mathbf{b}) \\ \rho(F\varphi, \sigma, i, \mathbf{b}) &= \sup_{k \in [i, \mathbf{b}]} \rho(\varphi, \sigma, k, \mathbf{b})\end{aligned}$$

where **inf** and **sup** are the infimum and supremum operators, respectively. Note that the computation of the satisfaction score is restricted to a subsequence of σ between i to $\mathbf{b}$ (i.e., $z_i, z_{i+1}, \dots, z_{\mathbf{b}}$). The bound $\mathbf{b}$ may be determined by, for example, the planning horizon used by a planner (i.e., the length of action sequence used by the planner for behavioral prediction). For brevity, the definition for the until (U) operator is omitted, but similar to the one described in [37].

Example 2 Reusing the embedding predicate ap as defined in Example 1, consider the ETL specification $\varphi = G(ap)$, which requires that the visual concept encoded by predicate ap is always observed. Suppose the system generates the embedding trace $\sigma = z_{12}, z_{13}, z_{14}, z_{15}$, with minimum distances to $\mathcal{Z}_{target}$ as $[\delta_{ap}(z_t)]_{t=12}^{15} = [0.327, 0.374, 0.403, 0.427]$.

Using the robustness semantics for predicates, $\rho(ap, \sigma, t, 3) = \varepsilon - \delta_{ap}(z_t)$, and setting $\varepsilon = \varepsilon_g = 0.409$, the predicate robustness at each timestep is: $[0.082, 0.035, 0.006, -0.019]$. The robustness of the temporal specification is then $\rho(\varphi, \sigma, 0, 3) = \min_{t \in [0, 3]} \rho(ap, \sigma, t, 3) = -0.019$. Intuitively, although the system remains close to the object not being grasped for most of the window, the specification is violated because at timestep 15 the embedding moves outside the threshold ε , corresponding to the frame $t=15$ in Figure 5.1 where the object is grasped.

5.3.2 ETL Specification Monitors

An ETL monitor evaluates the satisfaction of a specification over an embedding trace for a given system. ETL monitors are defined only over the finite trace of a system in order to be evaluated during the execution of the system, and thus are only defined for specifications that are well-defined over finite traces.

Definition 16 (Finite Trace). *For a given trace σ , a finite trace $\sigma \leq t$ is a trace consisting of the first $t + 1$ states of σ .*

We refer to a finite trace as a trace when clear from context. An ETL monitor is formally defined over a finite trace of embeddings by the following definition.

Definition 17 (ETL Monitor). *For a trace $\sigma \leq t$, system $\mathcal{M}$, and specification ϕ , an ETL Monitor is $M_\phi(\sigma \leq t) = (r_0^\phi, \dots, r_t^\phi) \in \{-1, +1\}^{t+1}$, where $r_i^\phi = \text{sgn}(\rho(\phi, \sigma \leq i, 0, i))$ for each $0 \leq i \leq t$, with $\text{sgn}(x) = +1$ if $x \geq 0$ and -1 otherwise.*

Intuitively, when the monitor is positive for a timestep, the system satisfies the ETL specification at that timestep. However, when the robustness of the system becomes negative at timestep i , the monitor raises an alert to indicate that the observed execution violates ϕ over the prefix $[0, i]$.

Semantic Correctness We denote an ETL monitor to be *semantically correct* iff it is equivalent with evaluation over ground truth executions. To compare ETL monitor to ground truth executions, we define a ground truth monitor over finite executions.

Definition 18 (Finite Execution). *For a given execution ζ , a finite execution $\zeta_{\leq t}$ is an execution consisting of the first $t + 1$ states of ζ .*

We refer to a finite execution as an execution when clear from context.

Definition 19 (Ground-Truth Monitor). *Let ω be a temporal logic formula over symbolic predicates, and let $\zeta_{\leq t}$ be a finite execution prefix. The ground-truth monitor for ω on $\zeta_{\leq t}$ is the binary-valued trace*

$$GT_\omega(\zeta_{\leq t}) = (b_0^\omega, b_1^\omega, \dots, b_t^\omega) \in \{-1, +1\}^{t+1},$$

where, for each prefix $\zeta_{\leq i}$ with $0 \leq i \leq t$,

$$b_i^\omega = \begin{cases} +1 & \text{if } \zeta_{\leq i} \models \omega, \\ -1 & \text{otherwise.} \end{cases}$$

Here, satisfaction $\zeta_{\leq i} \models \omega$ is defined by the semantics of the underlying temporal logic over symbolic predicates.

We define the semantic correctness of monitor M using the following definitions.

Definition 20 (Semantic Correspondence). *Let M be an embedding temporal structure with representation map $\eta = \psi_{\text{enc}} \circ \phi_{\text{obs}}$. Let ω be a specification over ground-truth state trajectories $\zeta_{\leq t}$ and let ϕ be an ETL specification over finite embedding traces $\sigma_{\leq t}$. We say that ϕ corresponds semantically to ω with respect to M if, for every finite execution prefix $\zeta_{\leq t} = (s_0, \dots, s_t)$, letting*

$$\sigma_{\leq t} = (\eta(s_0), \dots, \eta(s_t)),$$

we have

$$\sigma_{\leq t} \models \phi \iff \zeta_{\leq t} \models \omega.$$

Intuitively, we say that ϕ is a latent space specification *semantically corresponds* with the ground-truth specification ω if both are intended to capture the same behavioral property, with ω defined over state-based predicates and ϕ defined over embedding-space predicates.

Definition 21 (Semantic Correctness of an ETL Monitor). *Let ω be a ground-truth specification and let ϕ be an ETL specification that corresponds semantically to ω . An ETL monitor M is semantically correct with respect to (ϕ, ω) if, for every execution prefix,*

$$M_\phi(\sigma_{\leq t}) = GT_\omega(\zeta_{\leq t}).$$

This notion of semantic correctness provides the basis for evaluating ETL monitors in practice. In our experiments (Section 5.6), we approximate semantic correctness by comparing the outputs of the ETL monitor against ground-truth monitors derived from state-based specifications.

5.4 Constructing ETL specifications in Practice

The formal semantics proposed in the preceding section define ETL as a specification language over embedding traces. To use ETL in practice, however, several design choices must be instantiated concretely: one must determine how target embeddings are specified, how observations are mapped into the representation space, which distance function is used to compare embeddings, and how predicate thresholds are calibrated so that they align with the intended semantic concept. In this section, we first outline how target embeddings are specified. Then, we highlight the different choices of encoders and distance functions.

5.4.1 Specifying Target Embeddings

ETL is specified using predicates over *target embeddings* that correspond to parts of the physical world against which current observations are evaluated. In practice, the specifier (e.g., a system engineer) would not directly specify the mathematical representations of embeddings. Instead, the targets would be provided in the same format as the sensor’s input — e.g., for a camera, a target would be provided as a reference image — that is then translated into the target embeddings via the pre-trained encoder. This allows specifications such as “*eventually reach a state similar to this image*” or “*always avoid states resembling fire*” without requiring explicit symbolic labels. By enabling specification directly over learned representations rather than explicit symbolic labels, our approach removes the need to manually define a finite predicate vocabulary from observations, a process that has traditionally depended on substantial expert knowledge and task-specific engineering in robotics.

5.4.2 Choice of Encoders and Distance functions

The choice of distance function is another key practical consideration in evaluating ETL satisfaction. Distances over pretrained image embeddings capture perceptual similarity, but lack temporal context because these encoders are trained on static observations. World models mitigate this limitation by mapping observations into a latent space that evolves as a function of past states and actions. When trained with reconstruction-based objectives, such models often preserve much of the geometry of the original embedding space while enriching it with temporal structure. Consequently, similarity relationships defined over observation embeddings can often be carried over to the latent space in a meaningful way.

Accordingly, the choice of d should align with the geometry induced by the representation. For instance, cosine distance is a natural choice for contrastive embeddings, whereas L2 distance is often more appropriate for reconstruction-based latent representations. More generally, the se-

lected distance should be one that best reflects semantic similarity in the underlying space. We empirically study the effect of this design choice in Section 6.

5.5 Threshold Calibration for ETL predicates

The preceding section described how ETL predicates are instantiated in practice through target embeddings, encoders, and distance functions. We now turn to threshold calibration. Recall from Definition 11 that ETL predicates are evaluated by comparing embedding distances against a threshold ε . These thresholds play a critical role: they determine when a continuous similarity measure corresponds to predicate satisfaction. As a result, ETL specifications are not only parameterized by the encoder and distance function, but also by calibrated thresholds that defines when a predicate holds.

Intuitively, ε defines the *boundary of a concept* in the representation space (e.g., how close an observation must be to a goal embedding to count as “reached”). Unlike state-based predicates, thresholds in embedding spaces depend on the geometry induced by the encoder and the choice of distance function. Poorly chosen thresholds can lead to high false negatives (overly strict predicates) or false positives (overly permissive predicates), making calibration essential.

We propose a data-driven approach to calibrate thresholds for ETL predicates. We assume access to a dataset of trajectories (e.g., simulator rollouts or demonstrations) with ground-truth signals $\zeta_{\leq t}$ that satisfy a TL specification ω at timestep t in the underlying state space. Here, ω is a TL specification that semantically corresponds to the ETL specification being instantiated. Then, each trajectory is mapped to an embedding trace using the representation map η , yielding embeddings $z_t = \eta(s_t)$. From these embeddings, we compute distances to the target set as $d_t = d_u(z_t, T_u)$. We then select thresholds using a held-out calibration set of size n_{cal} .

We propose two concrete approaches for calibrating thresholds.

5.5.1 F1-Optimal Threshold

We calibrate each predicate threshold on a held-out set – $\mathcal{D}_{\text{cal}}$ – of trajectories with ground-truth labels. For each timestep t , we compute the embedding distance d_t to the target embedding and, for a candidate threshold ε_1 , predict that the predicate holds whenever $d_t \leq \varepsilon_1$. We then search over possible threshold values based on the observed calibration distances and select the threshold that maximizes the F1 score with respect to the ground-truth labels.

Definition 22 (F1-Optimal Threshold). *Let $\mathcal{D}_{\text{cal}} = \{(d_t, y_t)\}_{t=1}^N$ be the calibration set, where $d_t \in \mathbb{R}_{\geq 0}$ is the embedding distance at timestep t and $y_t \in \{0, 1\}$ is the corresponding ground-truth predicate label. For any candidate threshold $\varepsilon \in \mathbb{R}_{\geq 0}$, define the induced prediction*

$$\hat{y}_t(\varepsilon) = \mathbf{1}[d_t \leq \varepsilon].$$

Let

$$\text{F1}(\varepsilon) = \frac{2\text{TP}(\varepsilon)}{2\text{TP}(\varepsilon) + \text{FP}(\varepsilon) + \text{FN}(\varepsilon)},$$

where $\text{TP}(\varepsilon)$, $\text{FP}(\varepsilon)$, and $\text{FN}(\varepsilon)$ denote the numbers of true positives, false positives, and false negatives, respectively, obtained by comparing $\hat{y}_t(\varepsilon)$ against y_t over $\mathcal{D}_{\text{cal}}$. The F1-optimal threshold is defined as

$$\varepsilon_{F1} \in \arg \max_{\varepsilon \in \{d_1, \dots, d_N\}} \text{F1}(\varepsilon).$$

If multiple thresholds achieve the same maximum F1 score, ties may be broken arbitrarily.

Intuitively, this chooses the threshold that best matches the intended concept by balancing false positives and false negatives on calibration data.

5.5.2 Conformal Threshold with Recall Guarantee

In safety-critical monitoring, missing a true positive event is often more costly than raising a false alarm. To bias threshold selection toward high recall, we compute a second threshold, ε_{CP} , using *split conformal prediction* [9, 81]. We employ conformal prediction as it allows us to make no assumptions about the underlying distribution of embedding distances, and we want a guaranty that transfers directly to deployment-time predicate evaluation.

In this work, we propose conformal *ETL predicate calibration* by adapting conformal prediction to select predicate thresholds from embedding-distance scores. Here, the goal is not to produce a prediction interval around a model output, but to calibrate the threshold used for ETL predicate evaluation.

We do this in two stages. Within each calibration trajectory, we compute a single score from all satisfying timesteps by taking the largest embedding distance among the frames that satisfy the predicate. This score represents the *hardest positive case* in that trajectory: any threshold smaller than this value would miss at least one true positive frame in that same trajectory. Then across trajectories, we collect one such score from each calibration trajectory and apply split conformal prediction to these held-out scores. The resulting threshold ε_{CP} is therefore calibrated not to individual frames, but between different trajectories. As a result, for a new trajectory drawn from the same distribution, with probability at least $1 - \alpha$, the threshold is large enough to cover that trajectory’s hardest positive case, and hence will detect every ground-truth positive timestep in that trajectory. Here, $\alpha \in (0, 1)$ is a user-specified error level.

Setup Let $\varsigma^1, \dots, \varsigma^{n_{\text{cal}}}$ be n_{cal} calibration demonstrations, disjoint from training, and let z_g denote the target latent embedding that semantically corresponds to a ground-truth specification ω (see Definition 20). For each calibration demonstration ς^i , define the nonconformity score

$$\text{score}_i = \max_{t \in [0, |\varsigma^i|]} \begin{cases} d(z_t^{(i)}, z_g) & \text{if } \delta_{\omega}^{GT}(s_t^{(i)}) = 1 \\ -\infty & \text{otherwise.} \end{cases}$$

for a distance function $d \in \mathcal{D}_{\mathcal{X}}$ and where $\delta_{\omega}^{GT}(s_t^{(i)}) = 1$ indicates that the ground-truth state at time t satisfies predicate ω . Intuitively, score_i is the most difficult positive timestep in demonstration ς_i : any threshold below score_i would fail to classify at least one ground-truth positive frame in that demonstration. Each calibration demonstration contains at least one ground-truth positive timestep.

Definition 23 (Conformal Calibration Threshold). *For a calibration set of size n_{cal} and a target mis-recall level $\alpha \in (0, 1)$, a conformal calibration threshold ϵ_{CP} is a value in $\mathbb{R}_{\geq 0}$ defined as follows:*

$$\epsilon_{\text{CP}} = \text{score}_{(k)}, \quad k = \lceil (1 - \alpha)(n_{\text{cal}} + 1) \rceil,$$

where $\text{score}_{(k)}$ denotes the k -th smallest value among $\{\text{score}_i\}_{i=1}^{n_{\text{cal}}}$.

Theorem 5 (Conformal Recall Guarantee). *Assume the calibration demonstrations and a future test demonstration are exchangeable. Then*

$$\mathbb{P}\left(\forall t : \delta_{\omega}^{GT}(s_t^{(i)}) = 1 \Rightarrow d(z_t, z_{\omega}) \leq \epsilon_{\text{CP}}\right) \geq 1 - \alpha.$$

Equivalently, with probability at least $1 - \alpha$ over a newly drawn test demonstration, the embedding predicate achieves perfect per-demonstration recall, i.e., it detects every ground-truth positive timestep.

Proof sketch. By exchangeability, the augmented set of scores

$$\text{score}_1, \dots, \text{score}_{n_{\text{cal}}}, \text{score}_{\text{test}}$$

is exchangeable, where

$$\text{score}_{\text{test}} = \max_{\substack{t \in \zeta_{\text{test}} \\ \delta_{\omega}^{GT}(s_t) = 1}} d(z_t, z_{\omega}).$$

The standard split conformal order-statistic argument implies

$$\mathbb{P}(\text{score}_{\text{test}} \leq \text{score}_{(k)}) \geq \frac{k}{n_{\text{cal}} + 1} \geq 1 - \alpha.$$

By construction, the event $\text{score}_{\text{test}} \leq \epsilon_{\text{CP}}$ is exactly the event that every positive timestep in the test demonstration satisfies $d(z_t, z_{\omega}) \leq \epsilon_{\text{CP}}$, which proves the recall guarantee. $\square$

At deployment, the predicate is evaluated solely by checking whether

$$d(z_t, z_{\omega}) \leq \epsilon_{\text{CP}},$$

The guarantee holds for any embedding function and any underlying data distribution, provided the calibration and test demonstrations remain exchangeable.

5.6 Evaluation

We evaluate the semantic correctness of ETL monitors by comparing embedding predicate outputs against ground-truth propositions.¹ All experiments were run on a compute cluster using two NVIDIA GeForce RTX 5090 GPUs, each with 34.2 GB of memory.

¹Artifacts for reproducibility are provided at <https://github.com/ETLMonitoringAuthors/ETLMonitoring>

5.6.1 Simple Navigation Dubins Car

First, we evaluate ETL specification based runtime monitoring in a two dimensional navigation task where privileged information about the state space is available, similar to the one proposed in [1]. This fully observable setup allows direct comparison between embedding-based predicates and ground-truth specifications.

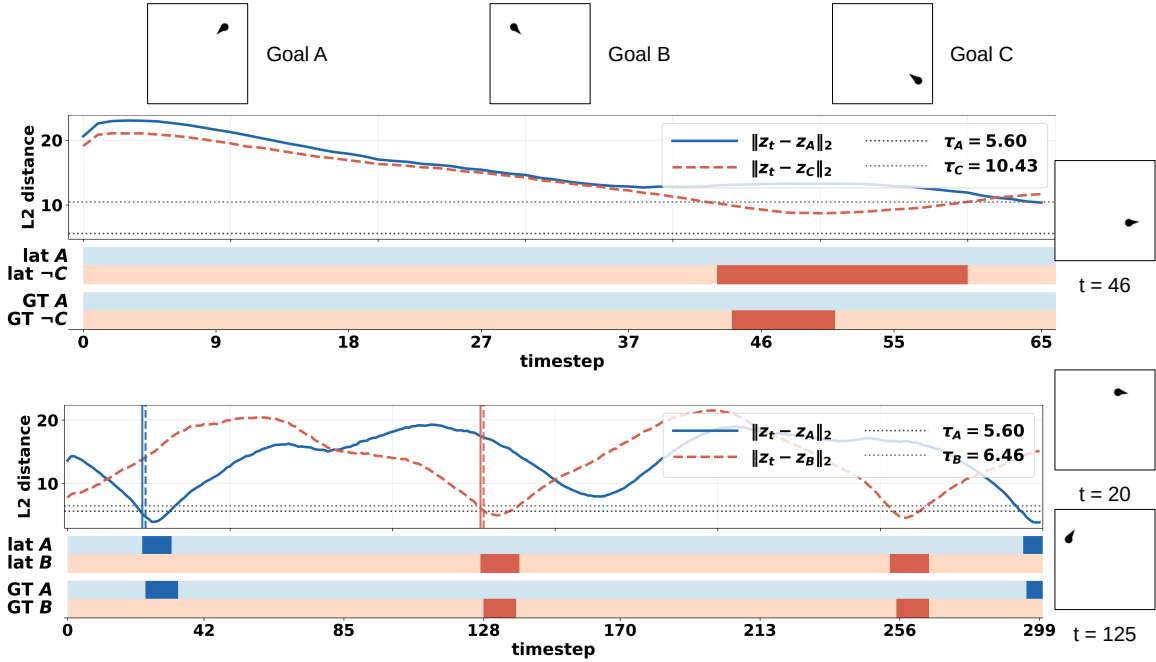
Setup The navigation task is defined for a robot that respects discrete-time Dubins car dynamics in a controlled environment. We generate $N = 100$ trajectories using a feedback controller with obstacle avoidance. For each task, we construct pairs of equivalent specifications in the ground-truth state space, ω_i , and in the embedding space, ϕ_i . We utilize the encoder from the world model used in [1], which is based on Dreamer [48]. The encoder produces a task-relevant representation space in which distances reflect semantic similarity between observations. We evaluate four specification patterns: *Reach* ($\mathbf{FA}$), *Avoid* ($\mathbf{G}\neg C$), *Reach-Avoid* ($\mathbf{FA} \wedge \mathbf{G}\neg C$), and *Sequential* ($\mathbf{F}(A \wedge \mathbf{FB})$). A , B , and C denote reaching a goal in the top-right, the top-left, and the bottom-right corners of the environment respectively. For visualization, see the goals in Figure 5.2a. ETL monitoring is evaluated by comparing the satisfaction of each embedding-based specification ϕ with the satisfaction of corresponding ground-truth specifications ω on the respective latent and state traces for each timestep of each trace. We report precision, recall, and F1 to assess how closely ETL predicates match ground-truth events. For temporal specifications, we measure trajectory-level satisfaction agreement with the ground-truth monitor; for sequential specifications, we also report ordering accuracy of detected subgoals. Additional details can be found in Appendix A.1.1.

Results We now discuss our findings; a tabularized version of results can be found in Appendix A.1.2.

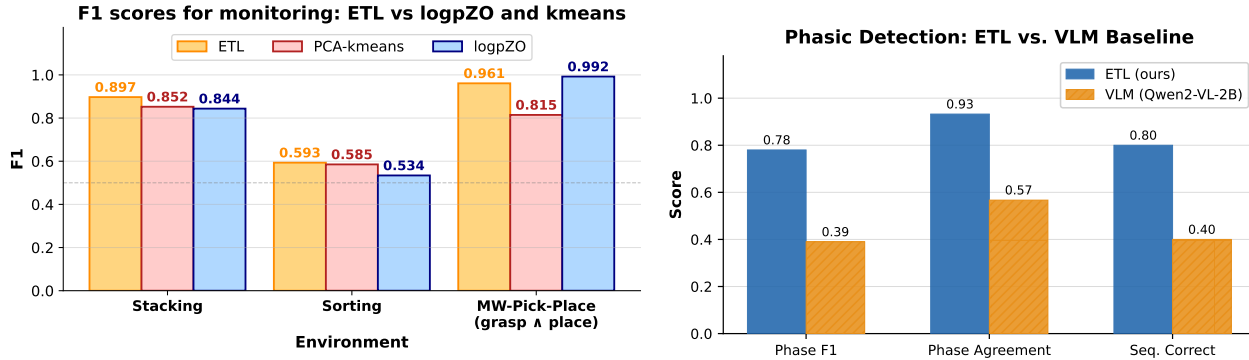
Can ETL monitors achieve semantic correctness as defined in Definition 15 with respect to ground-truth specifications in a controlled environment? ETL shows strong semantic correctness for atomic predicates, visualized in Figure 5.2a. Across reach, avoid, and reach-avoid specifications, the F1-optimal monitor achieves F1 scores of 0.80–0.85 with agreement above 96%, indicating that embedding predicates closely match the corresponding state-based events. This implies that embedding-space predicates can serve as faithful proxies for state-based propositions. Additionally, for the sequential specification $A \rightarrow B$, the monitor achieves 100% precision, recall, and ordering agreement at the episode level. Thus, ETL is not limited to detecting isolated semantic events; *ETL can also correctly track their ordering over time.*

Does threshold calibration produce the intended precision–recall tradeoff for safety-oriented monitoring? Relative to the F1-optimal threshold, ϵ_{CP} increases recall for reach (from 0.83 to 0.93) while incurring only a modest drop in precision (from 0.87 to 0.79), with essentially no change in agreement. This shows that conformal calibration provides a practical safety-oriented operating point: it makes the monitor more conservative against missed detections without substantially changing overall semantic alignment.

Overall, the Dubins results show that ETL monitors align closely with state-based specification monitors, support monitoring of temporally composed behaviors, and exhibit a clear precision–recall tradeoff under threshold calibration.



(a) Boolean predicate traces on the Dubins car. Goals A , B , and C are visualized along top of the image; the graph plots the L2 distance between embeddings and along with the F1-optimal thresholds. The bars along the bottom show when the latent and ground-truth predicates are satisfied (darker bars); two observation states are displayed on the left of the graph.



(b) Predicate-monitoring F1 scores across domains.

(c) Comparison of ETL and Qwen2-VL-2B on phasic DROID episodes.

Figure 5.2: Qualitative and quantitative ETL results.

5.6.2 Simulated Manipulation Tasks

To assess whether ETL remains effective beyond the controlled navigation domain, we evaluate it for simulated contact-rich manipulation tasks with richer visual scenes, complex object interactions, and additional distractor objects that make perceptual monitoring harder. We consider two environments with complementary task structure: D3IL [64] and MetaWorld [100]. For D3IL we consider two complex tasks: SORTING requires a Franka robot to push two blocks into their corresponding color-matching target boxes, while STACKING requires the robot to arrange colored blocks in a target region. For MetaWorld, we evaluate ETL on robotic-arm pick-and-place tasks with sequential grasp and place subgoals. Unlike single-goal manipulation tasks, these benchmarks expose phase-like progress structures, making them a natural testbed for evaluating whether ETL predicates can monitor task-relevant semantic milestones in rich manipulation settings.

Setup We compare our ETL-based monitoring approach against recent embedding-based monitoring baselines: PCA-kmeans [87] and logpZO [129]. PCA-kmeans clusters principal components of successful observation embeddings and scores a new observation by its distance to the nearest cluster center, while logpZO fits a flow-matching density model over observation embeddings and uses the inferred latent norm as an uncertainty score. These baselines are the most direct comparison for ETL because they operate on observation embeddings and detect distributional deviations. We evaluate monitoring performance by comparing predicted satisfaction or violation labels against ground-truth labels derived from simulator rewards and state variables. Additional details regarding our experimental setup are provided in Appendix A.2.1.

Results: Can ETL outperform embedding-based monitoring baselines on simulated manipulation tasks? Figure 5.2b shows that ETL matches or exceeds the observation embedding baselines on average across the three simulated manipulation environments. Using ϵ_{F1} , ETL achieves the highest average F1 score of 0.817, compared with 0.790 for logpZO and 0.751 for PCA-kmeans. ETL improves over the strongest baseline on STACKING (0.897 vs. 0.852) and slightly outperforms PCA-kmeans on SORTING (0.593 vs. 0.585). SORTING is the hardest setting for all methods, suggesting that its OOD shifts induce more ambiguous changes in the embedding space. On MetaWorld pick-place-wall, logpZO reaches near-saturated performance (0.992 F1), indicating that the predicate boundary aligns closely with the support of successful embeddings. ETL remains competitive at 0.961 F1, while PCA-kmeans lags behind at 0.815.

5.6.3 Real World Evaluation via DROID

Finally, we evaluate ETL on real-world robot data to assess its effectiveness for monitoring in-the-wild temporally extended manipulation behaviors. To this end, we utilize the manipulation dataset DROID [74], which contains large-scale, diverse, and compositional manipulation demonstrations in more realistic settings.

Setup We analyze manipulation episodes with compositional, multi-phase structure. We instantiate ETL specifications over phase-based predicates, where each phase corresponds to a contiguous

interaction window in the demonstration. This setup allows us to evaluate how ETL monitoring performs over real-world manipulation episodes in comparison to a Vision Language Model (VLM)-based baseline monitor. For evaluation, we compute F1 score, agreement, and sequential ordering accuracy over phase-based predicates. For our baseline, we use a state-of-the-art VLM: Qwen2-VL [124]. For each phase, the VLM is prompted with the ground-truth desired behavior and phase frame to determine if a task was completed; the response is distilled to a Boolean value, which is then used to compute frame-level F1, agreement, and sequential ordering accuracy for comparison with the ETL monitor.

Results: Can ETL accurately monitor perceptual specifications on real-world manipulation data in comparison to a VLM-based baseline for monitoring phasic manipulation behaviors?

ETL achieves a mean F1 score of 0.813 and mean agreement of 0.940, whereas the Qwen2-VL baseline reaches a mean F1 score of 0.390 and mean agreement of 0.567. Though ETL and Qwen2-VL perform competitively on one of the five tasks, these results demonstrate that ETL generalizes to diverse, unstructured manipulation scenes, and is markedly more reliable overall, especially on phases that are visually similar but semantically distinct. Additionally, ETL correctly identifies that sequential phasic ordering in 4/5 episodes (Figure 5.2c), indicating it can monitor compositional task structure beyond isolated phase detection. The single failure occurs in an episode with low inter-phase separability, suggesting that monitoring performance degrades when distinct phases occupy overlapping regions of the latent space. Tabularized results can be found in Appendix A.3.1.

Overall, these results show that ETL extends beyond simulation to real-world manipulation data. ETL successfully monitors temporal and sequential task structure across demonstrations. Moreover, compared to a VLM-based baseline, ETL provides substantially more reliable monitoring of phasic behaviors without requiring explicit language supervision.

5.7 Summary

We have developed ETL as a runtime monitoring framework for perception-based autonomous systems. By defining temporal logic predicates directly over the embedding space of pretrained encoders, we eliminate the need for hand-crafted feature extractors and brittle state-to-proposition mappings. Our threshold-mining procedure couples an F1-optimal calibration bound with a distribution-free conformal recall guarantee, providing actionable safety margins without manual annotation.

Empirical results on three environments spanning 2D navigation, robot manipulation, and contact-rich pick-and-place confirm that ETL monitors are semantically correct ($F1 \geq 0.83$ on distinctive specs), task-specific (cross-task FPR = 0), and capable of tracking complex sequential specifications with 100% ordering accuracy. These results lay a rigorous empirical foundation for deploying ETL monitors on physical robot systems and suggest natural extensions to multi-task deployments, temporal abstraction over subtask boundaries, and adaptive threshold recalibration online. The current approach has two main limitations: latent predicates are not yet fully interpretable in human-understandable terms, and monitoring performance depends on whether task-relevant semantic concepts are well separated in the encoder’s representation. These limitations

point to promising directions for future work, including more transparent predicate explanations, encoder selection and adaptation for improved semantic separation [136], temporal abstraction over subtask boundaries [125], and adaptive online thresholding [10].

Chapter 6

Constrained Decoding for Foundation Models

Recent advances in developing large transformer-based models for embodied navigation have enabled general-purpose policies that map multi-modal inputs such as RGB images, natural language instructions, and proprioceptive inputs to action sequences [60]. For example, Shortest Path Oracle Clone (SPOC) [36], PoliFormer [137], Flare [59], and LEO [61] exhibit impressive zero-shot generalization for embodied navigation tasks such as open vocabulary object-goal navigation (“find a mug”) and attribute-conditioned navigation (“locate the chair closest to the refrigerator in the kitchen”). These models have also shown promising transfer for robot navigation in real-world settings [61, 137]. However, these policies are primarily data-driven and lack any explicit notion of safety. Although policies may implicitly exhibit safety-related behaviors depending on patterns in their training data, there is no formal guarantee that they will consistently behave safely in all situations. This limitation hinders the deployment of such policies in the physical world where adherence to explicit safety rules and regulatory constraints is essential.

Formal specifications have long been used to specify contextual safety requirements for robotic deployments [39, 97]. Specifically, TLs provide a structured language for specifying rich safety constraints on robot behavior, ranging from simple invariants (e.g., remaining within permitted operational zones or avoiding dangerous obstacles) to contextual and history-dependent specifications. For example, a robot may be required to avoid certain regions until visiting another region (temporal ordering), permanently avoid a region after entering a restricted area (conditional safety) or never return to a region once it has exited. Such specifications naturally capture real-world operational rules that depend not only on the robot’s current state, but also on what situations the robot has previously encountered.

Although TL has seen success in classical robotic planning and reinforcement learning for safety constraint satisfaction, its use for enforcing safety for large transformer-based robot policies remains limited [93]. Additionally, retraining or fine-tuning these large pre-trained policies to directly embed such non-Markovian TL safety specifications is challenging [72]. First, retraining policies is a costly endeavor in terms of computational resources and data requirements. Moreover, due to the stochastic nature of these models, it is difficult to guarantee strict satisfaction of safety constraints through training alone. Hence, there is a pressing need for methods that can reliably

enforce safety specifications at inference time without policy retraining.

In the field of natural language processing, syntactic constraints have been successfully enforced by applying *constrained decoding* at inference time [2, 19, 127]. These approaches typically mask out tokens that violate a syntactic constraint defined over token sequences. For example, regular expressions (regex) represent a widely used form of syntactic constraint, requiring that generated token sequences conform to predefined structural patterns [19, 127]. Inspired by this line of work, we extend the paradigm of constrained decoding to enforce contextual safety constraints over action trajectories in dynamical systems. We propose *safety specification aligned decoding* (*SafeDec*) for transformer based policies that ensures generated action sequences provably satisfy STL specifications under assumed dynamics. Our key insight is that decoding-time interventions can be used not just to filter unsafe actions, but to *condition the generation process itself* on specification satisfaction. This conditioning is critical because it steers the model toward generating safety specification satisfying actions rather than relying on post hoc rejection. To enforce safety specifications, we leverage the formal semantics of STL to evaluate candidate actions at runtime and mask those that lead to future violations. Our method is agnostic to the underlying foundation model, requiring only two properties: (1) access to the decoding-layer logits during inference, and (2) access to an approximate dynamics model to predict future states. We focus on STL *safety* [5] specifications without liveness operators, since such specifications are sufficient to model a wide range of safety constraints encountered in robot learning [53, 69, 135, 141].

6.1 Background: Constrained Decoding in Transformers

A large variety of autoregressive transformer-based models generate final outputs by producing a probability distribution over the model vocabulary at each timestep. Then, through the process of *decoding*, tokens are selected to maximize the overall likelihood of an output sequence. In standard decoding, this maximization can be performed by either greedily selecting the most probable token at each step or by using a beam search to maintain multiple high-likelihood candidates. However, this often leads to degenerate output sequences that are repetitive [58]. A common approach is to use sampling strategies like top k [38], and nucleus sampling [58] that introduce stochasticity to encourage more diverse outputs. Constrained decoding [57] modifies this probabilistic selection by pruning invalid tokens to ensure that the generated sequences satisfy predefined constraints. These constraints are often syntactic, such as regular expressions, JSON formatting, or programming language grammars [126]. There is also recent work on enforcing *semantic* constraints that ensure coherence of the output or alignment with specific knowledge bases [106]. Formally, constrained decoding can be seen as maximizing the probability of the output sequence subject to a constraint $\mathcal{C}$: $\arg \max_{y \in \mathcal{Y}_{\mathcal{C}}} P(y | x)$ where $\mathcal{Y}_{\mathcal{C}}$ is the set of sequences satisfying $\mathcal{C}$.

6.2 Specification-Guided Constrained Decoding

In this section, we introduce a novel problem formulation for *SafeDec* in autoregressive transformer-based policies. First, we highlight the challenge in specification checking for these policies in con-

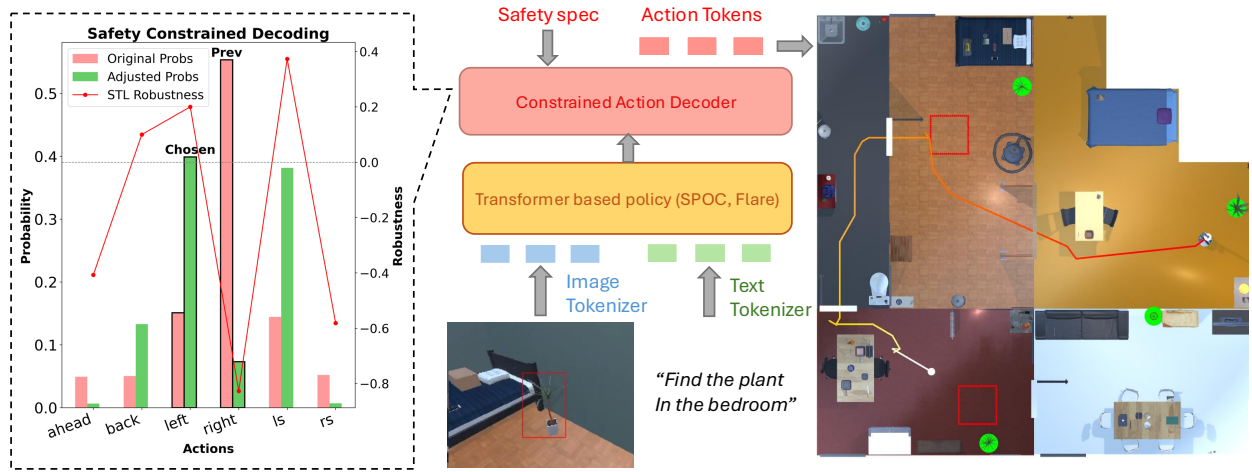


Figure 6.1: **Overview of our specification aligned decoding framework.** Given multimodal observations (RGB images and language goals), a pretrained transformer-based navigation policy generates candidate actions. Our constrained decoder then filters or reweights these actions using robustness scores from a user-defined STL safety specification. **Left:** Original action probabilities (red) from the pretrained policy are modified by SafeDec using STL robustness: actions predicted to violate the safety specification receive reduced weights, while safer actions are boosted. The chosen safe action and adjusted probabilities (green) are highlighted. **Right:** An example navigation episode where the robot starts in the kitchen (white dot) and moves toward the bedroom to locate the target object (green marker middle-right), while avoiding user-defined hazardous regions (red zones) throughout the trajectory.

trast to traditional syntactical constraint checking adopted by LLMs, and our solution to remedy it. Then, we propose two novel inference-time techniques for specification aligned decoding: *Hard Constrained Decoding (HCD)* and *Robustness Constrained Decoding (RCD)*.

6.2.1 Problem Statement

As highlighted in the background section, existing techniques in constrained decoding for language models enforce *syntactic constraints* defined over tokens, such as conforming to a context-free grammar or matching a regular expression. In these setups, constraint checking can be performed in the model’s token space.

In contrast, transformer-based policies operate in a physical environment and constraints (captured via temporal logic) are defined over state variables. Since a large class of end-to-end policies solely propose action sequences, specification checking can only be performed as actions are executed and the environment is simulated forward. In this case, constraint checking cannot be done solely in the token space and requires environmental feedback or a dynamics stepping function. Hence, we leverage an approximate first order dynamics function to compute specification satisfaction of different action sequences proposed by these policies.

Consider a discrete dynamical system with states $x_t \in \mathbb{R}^n$ and actions $a_t \in \mathbb{A}$ at time step t . The system’s dynamics are defined by $x_{t+1} = f(x_t, a_t)$ where $f : \mathbb{R}^n \times \mathbb{A} \rightarrow \mathbb{R}^n$ maps the current state ($x_t \in \mathbb{R}^n$) and a discrete action ($a_t \in \mathbb{A}$) to the next state $x_{t+1} \in \mathbb{R}^n$. This system is controlled by a policy that selects an action a_t at each time step based on observations and task context such as user-provided natural-language instructions or goal waypoints.

In this work, we focus on robot policies that generate actions based on multi-modal inputs, including sensor observations (e.g., RGB, depth, LiDAR) and natural language instructions. Let $\mathcal{J}_t$ represent the aggregated input at timestep t . Given the history of encoded inputs up to time t , a transformer-based policy parameterized by θ predicts embeddings for the next $T-t$ actions:

$$\{\hat{e}_{a_{t+k}}\}_{k=1}^{T-t} = \text{Transformer}_{\theta}(\{e_{\mathcal{J}_{\tau}}\}_{\tau=0}^t).$$

Each predicted action embedding is decoded into an action $\hat{a}_{t+k} \in \mathbb{A}$, resulting in a predicted action sequence $\{\hat{a}_{t+1}, \dots, \hat{a}_T\}$, where T denotes the planning horizon.

Now, consider that the system is required to satisfy requirements encoded using an STL formula φ defined over the state variables of the system. Formally, the goal is to ensure that the resulting trajectory satisfies the specification φ :

$$\{(x_0, \hat{a}_0), \dots, (x_T, \hat{a}_T)\} \models \varphi$$

Most of the existing techniques for specification enforcement perform post-hoc manipulation of proposed actions (after sampling from the policy) by filtering or rejecting action sequences that violate the specification φ . Although manipulation after sampling can ensure specification satisfaction, it can favor safe but low-likelihood actions and unintentionally suppress feasible safe behaviors that the model would have selected if safety constraints had been considered pre-sampling. This undermines the inductive biases learned during pretraining and lead to degenerate and brittle behaviors. A similar problem was highlighted when ensuring compliance with logical constraints

for large language models (LLM) in [104]. Additionally, these models decode actions sequentially, where each action a_t is conditioned on previously generated tokens $a_{<t}$. Post hoc manipulation can disrupt this causal chain and lead to a mismatch between the model’s internal hidden state and the executed sequence. Hence, we propose the following problem statement:

How can we enforce temporal logic constraints during action generation in robot foundation models such that the output sequence (1) satisfies an STL specification φ , and (2) remains faithful to the model’s autoregressive distribution $\pi(a_{1:T} \mid \mathcal{I}_{1:T})$?

Let $\pi(a_{0:T})$ be the *unconstrained* action-sequence distribution produced by the transformer-based policy’s decoder (e.g. the softmax over logits generated by the Transformer). We define the ideal constrained distribution over action sequences as:

$$Q_{\pi, \varphi}(a_{0:T}) = \frac{\pi(a_{0:T}) \cdot \mathbf{1}[(x_{1:T}, a_{0:T}) \models \varphi]}{\sum_{a'_{0:T}} \pi(a'_{0:T}) \cdot \mathbf{1}[(x'_{1:T}, a'_{0:T}) \models \varphi]} \quad (6.1)$$

where $x_{1:T}$ denotes the state trajectory induced by the system dynamics under actions $a_{0:T}$ and $\mathbf{1}[\cdot]$ is the indicator function that returns 1 iff the trajectory-action pair satisfies the specification. Equation 6.1 is the exact Bayesian conditioning of π on the event that the generated rollout satisfies φ . Hence, sampling from $Q_{\pi, \varphi}$ would give sequences that (i) inherit the original model’s preferences encoded in π and (ii) *guarantee* specification satisfaction.

In this work, we propose a technique to overcome the drawbacks of post-hoc safety enforcement methods (such as filtering) by leveraging constrained decoding techniques. Specifically, we propose `SafeDec`: A constrained decoding strategy that integrates STL specifications into the policy’s action selection process, ensuring satisfaction while remaining as close as possible to the base model distribution.

6.2.2 Hard Constrained Decoding

As highlighted in the background section, in the final layer, predictions are detokenized and a projection layer converts the embeddings into logits over the vocabulary space. These logits are further converted into a probability distribution using a softmax operation. In prior work, for structured output generation in LLMs, some invalid tokens are masked based on syntactical constraints or other criteria [104, 126]. This is done by setting their logit value as $-\infty$ before the softmax operation is applied. For HCD, we use a similar approach as constrained decoding literature [126] and mask out predicted action tokens that violate our given STL specification φ during sequential generation. Formally, to enforce the STL specification φ during sequential generation, we adjust the logits at each timestep $t+k$ as follows:

Let $\mathbf{z}_{t+k}$ denote the logits at timestep $t+k$. For each action choice i at timestep $t+k$, we define:

$$z_{t+k}^{(i)} = \begin{cases} -\infty, & \text{if } \hat{\tau}_{0:t+k}^{(i)} = (x_0, x_1, \dots, \hat{x}_{t+k}^{(i)}) \text{ violates } \varphi \\ z_{t+k}^{(i)}, & \text{otherwise} \end{cases}$$

where $\hat{x}_{t+k}^{(i)} = f(\hat{x}_{t+k-1}, \hat{a}_{t+k}^{(i)})$. Here, t is the current decision step, k is an index for the look-ahead step $t+k$ within a planning horizon of length T ($k \in [1..T]$), $\hat{a}_{t+k}^{(i)}$ is the action mapping to

Table 6.1: STL specifications used in our evaluation. Each region R_i denotes a spatial predicate over (x, z) .

Spec name	STL formula	Description
ϕ_{avoid}	$\mathbf{G}(\bigwedge_{i=1}^N \neg R_i)$	Avoidance: the robot must always stay outside all unsafe regions.
ϕ_{ordered}	$\mathbf{G}(\neg R_2) \vee (\neg R_2 \mathbf{U} R_1)$	Weak-until ordering: the robot must avoid region R_2 until reaching region R_1 , or avoid R_2 forever.
$\phi_{\text{no_return}}$	$\mathbf{G}(R_0 \Rightarrow \mathbf{G}(\neg R_0 \Rightarrow \mathbf{G}\neg R_0))$	Conditional no-return: if the robot is in region R_0 , then once it leaves R_0 , it may never re-enter.
ϕ_{geofence}	$\mathbf{G}(\bigvee_{i=1}^N R_i)$	Geofencing: the robot must always remain within at least one allowed region.

the token i and $\hat{x}_{t+k}^{(i)}$ is the next state value upon taking this action. This next state is elicited using a simple dynamics model (f) as highlighted in the previous section. Adjusting logits in this fashion ensures that any invalid token with respect to the safety specification will have zero probability of being selected after applying the softmax function.

6.2.3 Robustness Constrained Decoding

HCD ensures compliance but can lead to compromising task success, which can be undesirable. A similar tradeoff was observed by [86] when probability space-steering preserved model fluency while reducing toxic continuations compared with hard-filtering strategies that inflated perplexity and eroded diversity. Hence, we propose an alternative approach, called RCD, where we leverage the quantitative semantics of STL specifications (robustness). Unlike HCD, which applies hard masking to completely remove unsafe actions, RCD softly guides the model toward safer actions by incorporating robustness scores that reflect the degree of satisfaction of ϕ . This is similar to the approach proposed in [86] where the next-token distributions were re-weighted based on the utility scores provided by another language model. Our utility scores are quantified by the robustness function($\rho(\langle x_0, x_1, \dots, x_t \rangle, \phi)$) that returns a real-valued score indicating how well a predicted state satisfies the specification. Positive robustness values denote specification satisfaction, while negative values capture the degree of violation.

First, we compute a robustness score for each candidate action: $r_{t+k}^{(i)} = \rho(\langle x_0, x_1, \dots, x_{t+k-1}, \hat{x}_{t+k}^{(i)} \rangle, \phi)$ where $\rho(\cdot, \phi)$ is the STL robustness metric, and $\hat{x}_{t+k}^{(i)}$ is the predicted next state under action $\hat{a}_{t+k}^{(i)}$. This robustness score $r_{t+k}^{(i)}$ quantifies how well each candidate action satisfies the specification ϕ . These scores are then converted into weights using exponential scaling: $w_{t+k,i} = \exp(\alpha \cdot r_{t+k}^{(i)})$ where α is a temperature parameter that adjusts the sharpness of the bias. We use these weights to shift the original logits: $\tilde{z}_{t+k,i} = z_{t+k,i} + \beta \cdot w_{t+k,i}$ where β is a hyperparameter that modulates the trade-off between specification adherence and the original task objective. Finally, we obtain the action distribution by applying softmax over the adjusted logits: $p_{t+k} = \text{softmax}(\tilde{\mathbf{z}}_{t+k})$

This approach allows for graded preferences that improve flexibility and robustness to dynamics approximation errors. By ensuring that all actions preserve a non-zero probability, the policy remains capable of recovering from errors arising from an imperfect dynamics model. Concretely, if the predicted successor $\hat{x}_{t+1}$ is off by ϵ , an action that appeared marginally unsafe can be safe

in the true system, and vice versa. Retaining a weighted down probability for this action gives the sampler a fall-back option whereas HCD would completely rule this action out due to 0 probability. Since we are shifting the probability mass for unsatisfying actions, it is possible that they are still chosen and lead to a violation. However, this is a tradeoff that we allow to achieve a given task objective. We note that this still ensures higher STL satisfaction than unconstrained actions.

Our proposed constrained decoding framework (*SafeDec*) for autoregressive transformer-based robot policies operates at the final output layer. Operating entirely at inference time, *SafeDec* intervenes in the action selection process without modifying the model weights or requiring retraining. The base policy first generates action logits autoregressively from multimodal inputs such as images, language instructions, and proprioceptive states. A specification monitor then evaluates the task-specific safety requirements, expressed as STL formulas, over the predicted future trajectories. These future trajectories are generated using an assumed higher-order dynamics model. Finally, the constrained decoding layer selects actions that satisfy the STL specification through one of two proposed inference-time techniques: Hard Constrained Decoding (HCD), which enforces strict satisfaction by filtering out actions that would violate STL specifications, and Robustness Constrained Decoding (RCD), which softly adjusts the action distribution using robustness scores to balance safety and performance. This modular and lightweight design makes *SafeDec* a plug-and-play solution for ensuring specification-compliant robot behavior across diverse transformer-based policies.

6.3 Evaluation

6.3.1 Implementational Details

We comprehensively evaluate our constrained decoding framework on procedurally generated AI2-THOR [76] indoor scenes with diverse objects and layouts. AI2-THOR is a photorealistic 3D simulation environment built on the Unity engine and is a widely adopted benchmark and training environment for embodied navigation research. We use three state-of-the-art (SOTA) generalist robot navigation policies: Shortest Path Oracle Clone (SPOC) [36], PoliFormer [137] and Flare [59]. All three are large transformer-based policies trained on extensive language-conditioned robot trajectory datasets. These models achieve strong zero-shot generalization for a vast variety of navigation tasks that span open vocabulary object-goal navigation (“find a mug”), room-to-room traversal (“visit all rooms”), waypoint-based navigation (“move three meters forward and stop near the red rug”), and attribute-conditioned variants (“locate the chair closest to the refrigerator in the kitchen”). These models also demonstrate reliable zero-shot transfer to real-world environments, achieving robust task satisfaction.

In addition, these models capture three different training paradigms for generalist robot navigation policies. SPOC is trained purely with imitation learning from shortest-path rollouts. PoliFormer employs a hybrid approach that combines reinforcement learning and imitation learning, enabling it to learn long-horizon structure while retaining expert priors. Flare adopts a large-scale pretraining plus fine-tuning on embodied navigation data in line with recent foundation model training paradigms. This diversity in training paradigms allows us to evaluate the applicability of

SafeDec across different learning regimes.

In this work, we address safety specifications for robotics and, therefore, select those most relevant to real-world deployment. Our specifications are highlighted in Table 6.1. These specifications are inspired by common safe navigation specifications as highlighted in [97] and have varying levels of complexity. These specifications capture state-based invariants (ϕ_{avoid} , $\phi_{geofence}$), history-dependent conditionals (ϕ_{no_return}) and temporal ordering constraints ($\phi_{ordered}$). We en-

Table 6.2: Comparison by decoding technique across models (SPOC, Flare, PoliFormer) for specifications ϕ_{avoid} , $\phi_{ordered}$, and ϕ_{no_return} . Each cell reports STL satisfaction / success rate (%). Higher is better ($\uparrow$).

Decoding	ϕ_{avoid} : STL St / SR (% $\uparrow$)			$\phi_{ordered}$: STL St / SR (% $\uparrow$)			ϕ_{no_return} : STL St / SR (% $\uparrow$)		
	SPOC	Flare	PoliFormer	SPOC	Flare	PoliFormer	SPOC	Flare	PoliFormer
Unconstrained	72.0 / 82.5	75.5 / 82.0	77.0 / 82.5	76.0 / 80.0	72.0 / 80.5	71.0 / 81.5	84.0 / 79.5	82.0 / 81.0	82.0 / 79.0
Filtering	100.0 / 72.0	100.0 / 78.5	100.0 / 75.5	100.0 / 69.0	100.0 / 70.0	100.0 / 67.5	100.0 / 68.5	100.0 / 74.0	100.0 / 66.5
HCD	100.0 / 72.5	100.0 / 81.0	100.0 / 78.5	100.0 / 70.5	100.0 / 73.0	100.0 / 72.5	100.0 / 71.0	100.0 / 76.5	100.0 / 69.5
RCD	93.0 / 76.0	83.0 / 82.5	87.5 / 83.5	91.0 / 75.0	89.0 / 79.5	95.0 / 78.0	96.0 / 77.5	96.0 / 80.0	94.0 / 74.0

code our test STL specifications using an efficient computational graph-based STL library called STLGG++ that can evaluate multiple state signals in parallel [71]. This ensures minimal inference overhead at runtime (10^{-5} s per timestep), which is crucial for policy deployment. For our specifications, we generate random regions in the configuration space at runtime and use their coordinates as spatial predicates. For our dynamics model, we assume a unicycle model, an approximate first-order dynamics abstraction widely used in the robotics literature for analysis and control [28]. This representation captures the essential kinematics of motion in the plane and is widely used because it is applicable for diverse robotic platforms.

6.3.2 Experimental setup

We compare our proposed techniques with (1) an unconstrained base model and (2) a base model with a filtering mechanism. The filtering mechanism picks default actions that cannot immediately violate the specification (eg. turning in place) upon predicted violation of the safety specification, similar to the Simplex architecture [115]. Simplex architecture is a classic scheme in which a high-performance advanced controller is continuously monitored by a provably safe but less capable backup controller. Simplex based techniques have been used extensively for safety-critical robotics and are a widely accepted standard for runtime-safety comparisons. Additionally, our base policies represent actions as a discrete set of low-level action primitives (e.g., move forward/backward, lateral strafing, rotation, end, and a few other discrete navigation actions exposed by each policy). At each timestep, the policy outputs a categorical distribution over this shared action set. We evaluate performance using two main metrics: STL Satisfaction Rate (**STL St**), defined as the proportion of trajectories that satisfy the specified STL formula, and Task Success Rate (**SR**), which measures standard task success.

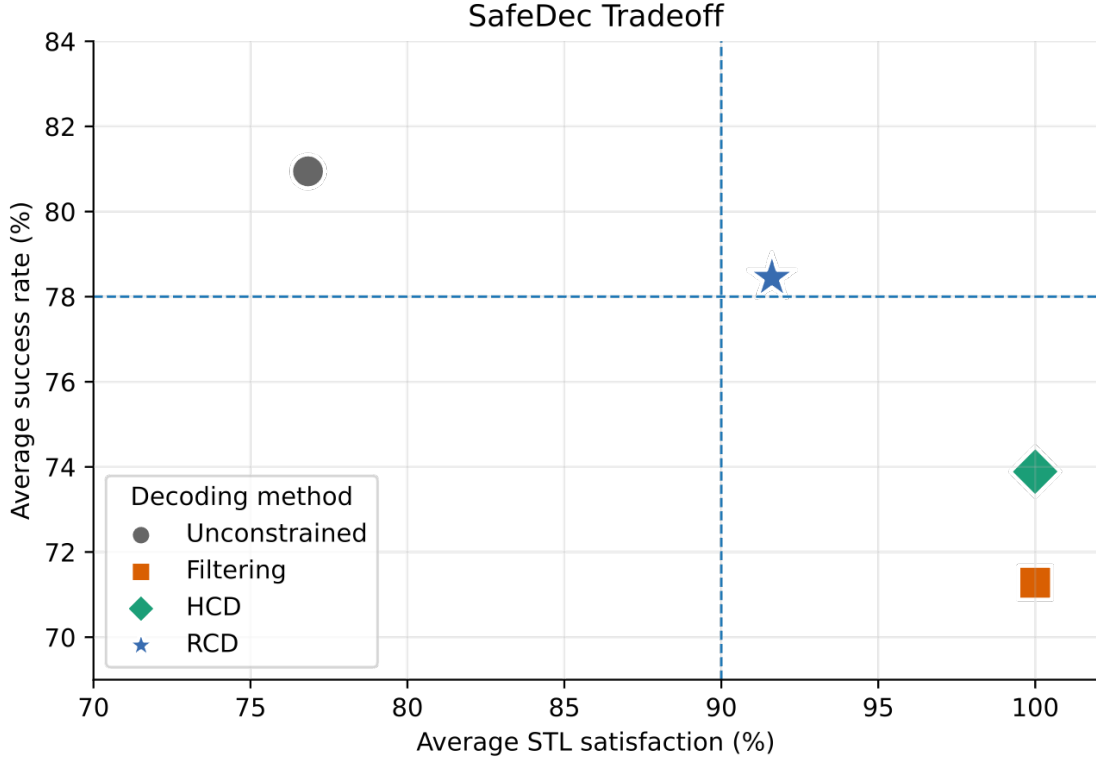


Figure 6.2: **SafeDec trade-off between STL satisfaction and task success.** Average STL satisfaction versus task success rate aggregated across all specifications and three policies. RCD achieves the strongest trade-off, substantially improving STL satisfaction while remaining close to the unconstrained policy’s task performance. HCD and filtering achieve near-perfect STL satisfaction, but incur a larger reduction in task success.

6.3.3 Results

Our results are highlighted in Table 6.2. For brevity, we provide the results for ϕ_{geofence} in A.5 since it is syntactically similar to ϕ_{avoid} . Additionally, we summarize the safety–performance trade-off across all specifications and policies in Figure 6.2. We also visualize sample trajectories in Figure 6.3 for one scene and task. Unless stated otherwise, all numbers are averaged over 200 evaluation episodes.

Do HCD and RCD provide higher STL satisfaction than the unconstrained baselines?

Both HCD and RCD consistently improve STL satisfaction relative to the unconstrained baselines across all models and specifications. All policies with RCD show significant gains, with improvements ranging from 8-24 percentage points. HCD achieves perfect STL satisfaction, with improvements ranging from 16-29 percentage points. We observe that the Simplex-style filtering baseline achieves similar STL satisfaction rate as HCD. This parity is expected, as both methods block any action predicted to violate the specification.



(a) Unconstrained vs. HCD



(b) Unconstrained vs. RCD

Figure 6.3: Top-down visualization of trajectories for a sample scene under the instruction “*find the sofa*”. The unconstrained model passes through two forbidden regions (red squares) on its way to the sofa (center-left). In contrast, HCD (left) and RCD (right) modify the trajectories to respect STL safety specifications while still reaching the goal. More visualizations are available in Appendix A.4.

Which safety enforcement technique achieves task success rates comparable to the unconstrained baselines? As highlighted in Table 6.2, Simplex-style filtering attains high STL satisfaction but sacrifices task success because the agent takes predefined safe actions. HCD shows similar behavior: although safety is maximized, success rates are consistently lower than the unconstrained baseline across models and specifications. However, as HCD factors in base model logits, it is able to achieve higher task satisfaction compared to Simplex-style filtering with improvements ranging from 0.5-3 percentage points. In contrast, RCD preserves success rates much closer to the unconstrained level with success-rate drops of 0.5–6.5 percentage points (smaller drop is better). However, we note that RCD does not fully recover success for certain models and specifications, task success remains several points below the unconstrained baseline. Nevertheless, RCD enforces safety while avoiding the large performance penalty observed with filtering.

Does RCD achieve better task success than HCD while maintaining high STL satisfaction? While both HCD and RCD improve safety over the unconstrained baseline, they differ in how they balance constraint satisfaction with task success. HCD enforces strict STL satisfaction that results in frequent conservatism and lower successful task completion rates. In contrast, RCD’s soft penalization leads to higher task success while still maintaining reasonable STL satisfaction. These results show that RCD achieves a better trade-off between safety and goal-directed behavior, especially in settings where occasional low-risk actions can lead to higher long-term rewards.

Overall, Our proposed techniques effectively enforce safety STL specifications during policy

Table 6.3: Comparison with SafeVLA on the SafetyChores benchmark. Lower cost indicates fewer safety violations, while higher success denotes better task completion.

Method	Cost ↓	SR (%) ↑
ISA (SafeVLA)	0.205	86.5
ISA + HCD	0.015	82.0
ISA + RCD	0.060	86.0
FLARE	0.192	82.0
FLARE + HCD	0.115	84.0
FLARE + RCD	0.155	79.0

execution. HCD ensures full compliance, but occasionally sacrifices task success due to strict truncation. RCD strikes a balance, offering high satisfaction rates and robust performance.

6.3.4 Comparison with safe reinforcement learning baseline

We additionally compare SafeDec with SafeVLA [138], a safe reinforcement learning method that proposes an integrated safety approach (ISA) using explicit safety costs. To enable a fair comparison, we focus on the fragile-object and dangerous-object specifications in the SafetyChores benchmark [138], which closely align with our setting, and construct corresponding STL specifications for evaluation. We report the composite safety cost defined in SafeVLA, which directly reflects the degree of safety violations across both object classes, along with task success.

Our results are highlighted in Table 6.3. Our findings indicate that both HCD and RCD substantially reduce safety violations relative to their respective vanilla baselines while largely preserving task success. In particular, integrating HCD with ISA (SafeVLA) achieves an order-of-magnitude reduction in safety cost ($0.205 \rightarrow 0.015$), demonstrating effective enforcement of fragile and dangerous object specifications with minimal impact on task performance. We also find that RCD+ISA preserves task success close to vanilla ISA while reducing safety violations. Our results show that SafeDec, despite being a training-free method, preserves task satisfaction close to the safeRL baseline (ISA) for our specifications. Additionally, combining SafeDec with the ISA policy leads to further reduction in safety costs, highlighting its complementary benefits to data-driven safe learning.

6.3.5 Ablation studies

Does inaccurate dynamics modeling substantially reduce STL satisfaction? In this work, we assume a simple unicycle dynamics model due to its generalization capability for diverse robotic platforms. Although this represents a high-level abstraction of true dynamics, such modeling simplifications are standard in the formally assured robot safety literature [28]. However, both RCD and HCD depend on this assumption and inaccurate modeling can impact STL satisfaction. To evaluate the impact of inaccurate dynamics modeling, we conducted an ablation in which we inject gaussian perturbations into the dynamics (0.01 m per step translational noise i.e. 5% of nominal forward step, 1 ° per step rotational noise i.e. 3.3% of yaw step) for both HCD and RCD. Our results are visualized in Figure 6.4a. Across all three base models, the drop in STL satisfaction rates

from baseline to noisy dynamics is relatively small. We observe that *SafeDec* shows graceful degradation under significant per-step disturbances.

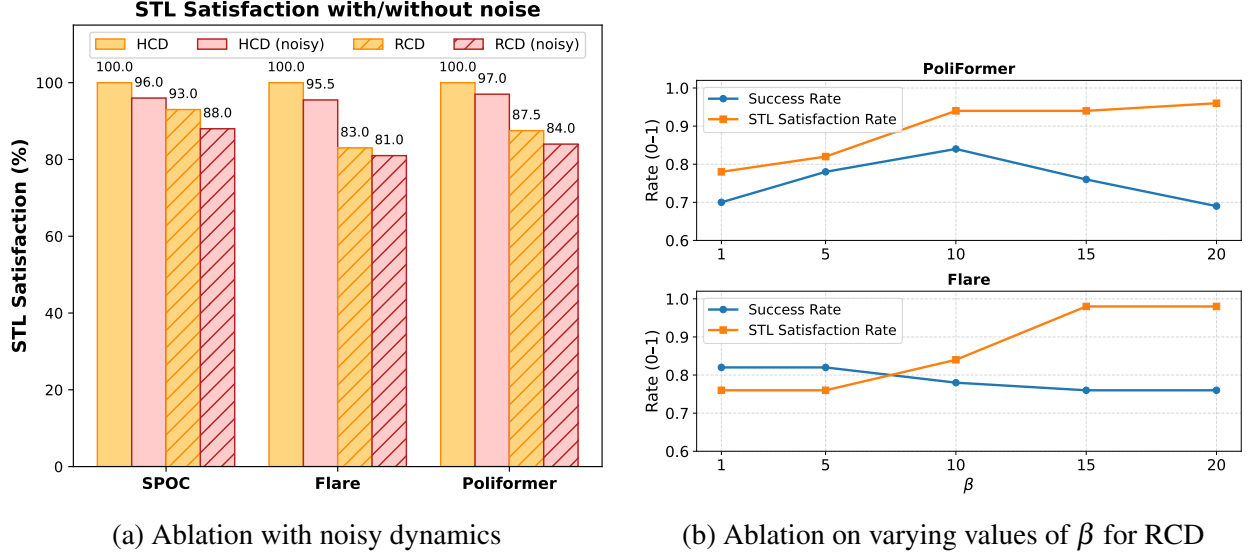


Figure 6.4: **Ablation studies.** (a) STL satisfaction (%) for HCD and RCD under baseline vs noisy dynamics across base models. (b) Effect of β on success rate and safety satisfaction.

How do varying values of β affect success rate and STL satisfaction in RCD? As highlighted in section 6.2.3, RCD uses a hyperparameter β that modulates the trade-off between specification adherence and the original task objective. The value of β affects the relative weighting between robustness and base logits. To investigate the impact of β on the success rate and STL satisfaction, we performed an ablation with varying values for β for Flare and PoliFormer. As visualized in Figure 6.4b, We observe that as β increases for PoliFormer, both STL satisfaction and success rate improve in tandem until $\beta = 10$, suggesting that moderate regularization can actually aid policy execution. Beyond this, STL satisfaction continues to improve but at the cost of lower success rates. For Flare, larger β values improve STL satisfaction but reduce success rates. These results highlight that the influence of β is model-dependent but in general demonstrate that *SafeDec* provides a tunable mechanism to balance safety and performance objectives.

6.4 Summary

In this chapter, we introduce a constrained decoding framework for enforcing safety specifications for large transformer based robot policies. Our approach enables runtime adaptation to novel safety specifications without retraining. Through experiments across multiple simulated environments, we demonstrated that our method significantly improves STL satisfaction while maintaining high task success rates.

Chapter 7

Future Work

This thesis developed methods for making temporal logic more compatible with modern AI-driven planning systems by improving efficiency, expanding specification beyond explicit state variables, and enabling inference-time enforcement for pretrained policies. At the same time, the work opens several directions for future research. These directions are not merely extensions of the current methods, but part of a broader agenda toward making formal specifications a practical interface for general-purpose autonomous systems.

From Hand-Written to Learned Specifications A central assumption throughout this thesis is that the specification is provided by a human designer. While this is appropriate in many safety-critical settings, it also limits scalability and places a significant burden on users. An important direction for future work is to infer specifications automatically from demonstrations, corrections, preferences, or natural language instructions. For example, one could imagine learning STL or ETL formulas that capture recurring task structure from successful trajectories, or extracting safety invariants from examples of undesirable behavior similar to the work proposed in [139]. We do a more restricted form of formula learning by mining STL timing parameters from data using STLPG++, however we still don't learn the formula structure itself from scratch. In the long term, specification-driven autonomy will likely require a bidirectional interface in which specifications are not only enforced, but also proposed, refined, and repaired from data.

Planning and Control Directly from Rich Perceptual Specifications Although this thesis develops ETL as a formalism for specification and monitoring, a more ambitious direction is to use perceptual specifications directly inside closed-loop planning and control. This would require planners that can optimize behavior with respect to embedding-based objectives while accounting for the geometry, uncertainty, and possible non-stationarity of learned representation spaces. Such an approach could enable robots to plan from goals expressed through images, videos, or textual prompts rather than manually engineered state predicates. Additionally, planning directly in this embedding space involves the encoder to be trained with objectives such as latent temporal smoothness, similar to the work proposed in [125]. With these extensions, the practical usability of ETL in perceptual and unstructured environments would be significantly broadened.

Uncertainty-Aware Semantics and Enforcement The methods in this thesis largely assume that specification evaluation and action filtering are performed using point estimates of state, dynamics, or representation. Real deployments, however, involve uncertainty in sensing, forecasting, and action outcomes. An important research direction is to develop uncertainty-aware variants of STL and ETL, together with corresponding planning and decoding algorithms. This may include probabilistic predicate satisfaction, robustness measures over distributions, or risk-sensitive decoding rules that account for model uncertainty when determining whether an action should be allowed. Extending formal specification methods to uncertain settings is essential for translating these ideas from controlled benchmarks to real-world deployment.

Overall, I view the methods in this thesis as steps toward a broader vision in which formal specifications serve as an executable interface between human intent and modern learning-enabled autonomy. Advancing that vision will require not only faster algorithms and richer formalisms, but also tighter connections between data, prediction, interaction, and control.

Chapter 8

Conclusion

In this thesis, I argued that formal specifications should remain central to the design of safe autonomous systems, even as planning and control increasingly shift toward data-driven and foundation-model-based approaches. Modern AI systems have greatly expanded what robots can do, but they have also made it harder to state, enforce, and verify desired behavior in a precise and reliable way. My central claim has been that temporal logic remains a powerful framework for this purpose, but only if it is extended to address the practical demands of modern autonomy: computational efficiency, compatibility with perception-driven systems, and applicability to pretrained generative policies. This thesis developed a set of methods toward that goal as follows:

1. `STLINC` [70], a tool for decomposing deeply nested STL specifications for efficient trajectory planning.
2. `STLCG++` [71], a tool for efficiently computing and backpropogating through STL robustness using modern autodifferentiation frameworks such as JAX and PyTorch.
3. `SafeDec` [67], a lightweight decoding layer that can sit on top of existing robot foundation models and enforce user-designed STL specifications in a provably safe way.
4. `ETL`, [68] A new logic specification framework for specifying desired behavior at a higher level of abstraction such as image features.

The first contribution of this thesis addressed efficiency in planning from temporally rich specifications. Complex STL formulas are expressive, but they often hide sequential task structure behind nested temporal operators, making planning unnecessarily difficult. To address this, I presented a decomposition-based view of STL planning that translates long-horizon nested specifications into incrementally satisfiable reachability and invariance constraints. This decomposition makes the sequential structure of the task explicit and enables planners to reason over smaller, more manageable subproblems. In doing so, it reduces the burden imposed by deeply nested temporal requirements and provides a more scalable path toward long-horizon specification-guided planning.

The second contribution addressed the computational bottleneck in STL robustness evaluation. Existing differentiable STL evaluation tools rely heavily on recursive or recurrent computation,

which limits parallelism and makes them increasingly expensive for long trajectories and large batches. In response, I developed STL_{CG++}, a masking-based approach to STL robustness computation that is better aligned with modern automatic differentiation frameworks. By reformulating temporal aggregation in a way that supports parallel execution across time, STL_{CG++} improves the practicality of using STL not only for monitoring, but also inside optimization loops, gradient-based planning, diffusion-based generation, and parameter mining. More broadly, this contribution shows that temporal logic evaluation itself can be redesigned to better match the computational structure of contemporary machine learning systems.

The third contribution of this thesis addressed a more fundamental limitation of classical temporal logic: its dependence on explicit state variables. In many modern robotic systems, especially those built around learned perception pipelines, the variables needed to define traditional symbolic predicates are unavailable, unreliable, or expensive to engineer. To overcome this limitation, I introduced Embedding Temporal Logic (ETL), a specification framework in which predicates are defined over learned representations rather than hand-specified state variables. This shifts formal specification from low-dimensional symbolic state spaces to semantically meaningful embedding spaces induced by pretrained models. With ETL, I showed that it is possible to monitor temporal properties directly over perceptual representations, making temporal logic more compatible with perception-heavy autonomous systems. In this sense, ETL expands the scope of formal specification from systems with explicit state access to systems that reason through learned latent structure.

The fourth contribution addressed applicability to foundation models. Large pretrained policies offer strong generalization and semantic priors, but they do not natively provide mechanisms for enforcing strict behavioral constraints at test time. Retraining them for every new safety requirement is expensive and often impractical. To address this, I proposed specification-guided constrained decoding methods that enforce STL safety requirements during inference. Rather than treating safety as a post hoc filtering step alone, these methods intervene directly in the action selection process by masking or reweighting unsafe candidate actions using specification robustness. This makes it possible to adapt a pretrained policy to new safety rules at deployment time without changing the model parameters. I view this as an important step toward making formal methods usable in the era of foundation models: instead of competing with pretrained generative models, formal specifications can shape their behavior at inference time.

This thesis suggests that the future of safe autonomy will depend on specification mechanisms that are not only formally grounded, but also computationally compatible with modern AI pipelines. It is no longer sufficient for a specification language to be expressive in theory; it must also be usable with batched tensor computation, high-dimensional perception, approximate world models, and large pretrained policies. My work takes several steps in that direction, but it also points to a larger research agenda: developing specification-driven autonomy stacks in which formal requirements are present throughout the lifecycle of a system, from task design and planning to runtime monitoring, adaptation, and intervention.

In conclusion, this thesis advances the view that formal specifications are not an artifact of an earlier robotics paradigm, but an increasingly necessary component of trustworthy AI-driven planning. By improving the efficiency of STL-based planning and evaluation, extending specification design to learned embeddings, and enabling inference-time enforcement for foundation models, I

have aimed to make formal specification a more practical tool for modern autonomous systems. I hope this thesis helps move the field toward autonomous agents that are not only capable and general, but also structured and safe.

Appendix A

A.1 Additional Evaluation Details for Dubins Car

A.1.1 Details for Dubins Car Setup

The navigation task is defined for a robot that respects discrete-time Dubins car dynamics. A state is defined as $s = [p^x, p^y, \theta]$ where $s_{t+1} = s_t + \Delta t[v \cos(\theta_t), v \sin(\theta_t), a_t]$ with continuous angular velocity $a_t \in A = [-a_{\max}, a_{\max}]$. We fix $a_{\max} = 1.25$ rad/s and $v = 1$ m/s, with discretization at $\Delta t = 0.05s$. We generate $N = 100$ trajectories using a feedback controller with obstacle avoidance. For each task, we construct pairs of equivalent specifications in the ground-truth state space and the embedding space. The ground-truth specifications, ω_i , are defined over the state space of the car (e.g., the position p^x, p^y , angular velocity y , etc.). In parallel, the ETL specifications, ϕ_i , are defined over the embedded images observed during simulation; target embeddings are obtained from the simulated observations corresponding to selected goal states. We utilize the encoder from the world model used in [1], which is based on Dreamer [48] with a Recurrent State Space Model (RSSM). The encoder produces a task-relevant representation space in which distances reflect semantic similarity between observations.

We evaluate four specification patterns: *Reach* (**FA**), *Avoid* (**G** $\neg$ **C**), *Reach-Avoid* (**FA** $\wedge$ **G** $\neg$ **C**), and *Sequential* (**F**(**A** $\wedge$ **F****B**)). *A*, *B*, and *C* denote reaching the top-right goal $|p_t - (0.8, 0.8)| < 0.25$, reaching the top-left goal $|p_t - (-0.8, 0.8)| < 0.25$, and entering the obstacle proximity zone $|p_t - (0.8, -0.8)| < 0.5$, respectively. This setup enables direct comparison between embedding-based satisfaction and ground-truth logical semantics. Thresholds are calibrated on a 40/60 calibration/test split using both F1-optimal and conformal procedures. In all experiments, we use $\alpha = 0.10$ for computing ϵ_{cp} .

A.1.2 Results for Dubins Car

The results for the experiments with the Dubins car can be found in Table A.1, including F1, Precision, Recall, and Agreement scores for both F1-Optimal and Conformal Prediction thresholds.

Table A.1: ETL predicate evaluation on the Dubins car.

Spec	Scope	ϵ^* (F1-optimal)				ϵ_{CP} ($\alpha=0.10$)		
		F1	Prec.	Rec.	Agree	Prec.	Rec.	Agree
Reach A	frames	0.85	0.87	0.83	98.6%	0.79	0.93	98.5%
Avoid C	frames	0.80	0.69	0.94	96.5%	0.70	0.92	96.5%
RA $A \wedge \neg C$	frames	0.85	0.87	0.83	98.6%	0.79	0.93	98.5%
Seq. $A \rightarrow B$	episodes	1.00	1.00	1.00	100%	1.00	1.00	100%

Table A.2: ETL predicate evaluation on MetaWorld (test split). Frame-level metrics are reported for all predicates.

Task	F1	Prec.	Rec.	Agree	CP Rec.	CP Prec.
Grasp (A)	0.985	0.971	0.999	97.3%	0.883	0.976
Place (B)	0.948	0.901	1.000	91.4%	0.916	0.926
Sequential	100% ordering agreement (24/24)					

A.2 Additional Details for Manipulation Tasks

We provide qualitative visualizations of ETL predicate behavior on manipulation tasks. These plots illustrate how embedding distances evolve over time and how predicate satisfaction aligns with ground-truth signals.

A.2.1 Tabularized Results

We present tabularized results from the experiments from Section 5.6.2 in Table A.2 and Table A.3.

A.2.2 Sequential Predicate Trace

Figure A.1 illustrates dual-predicate traces for the pick-place-wall task. The plots show distances to both subgoal embeddings (z_A for grasp and z_B for place), along with corresponding predicate activations.

The embedding distances to z_A and z_B decrease at distinct timesteps corresponding to grasp and placement events. This temporal separation is preserved in the embedding space, allowing ETL to correctly identify the ordering of subgoals across all trajectories.

A.3 Additional DROID Evaluation Details

A.3.1 Tabularized Results

We present tabularized results from the experiments from Section 5.6.3 in Table A.4.

Table A.3: F1 score for failure / predicate detection across three environments. All methods use the F1-optimal threshold; ETL (CP) additionally reports the class-conditional split-CP threshold. Per-column best in **bold**.

Method	STACKING	SORTING	MW-PICK-PLACE	Avg
logpZO	0.844	0.534	0.992	0.790
PCA-kmeans	0.852	0.585	0.815	0.751
ETL (best F1)	0.897	0.593	0.961	0.817
ETL (CP)	0.887	0.556	0.923	0.789

Table A.4: Comparison of ETL and Qwen2-VL-2B on phasic DROID episodes.

Task	Ph.	ETL			Qwen2-VL-2B		
		F1	Agr.	Seq.	F1	Agr.	Seq.
Obj. → paper	2	0.968	0.977	✓	0.493	0.383	✓
Tap → spoon	2	0.839	0.937	✗	0.292	0.700	✓
4-way chain	4	0.531	0.862	✓	0.585	0.811	✗
White → blue cloth	3	0.884	0.959	✓	0.335	0.258	✗
Bottles → stove	3	0.844	0.966	✓	0.244	0.685	✗
Mean	–	0.813	0.940	4/5	0.390	0.567	2/5

A.3.2 Ground Truth Construction

Unlike simulation environments, DROID does not provide explicit task-phase annotations. We derive ground-truth predicates directly from proprioceptive signals:

$$\begin{aligned}\pi_{\text{hold}}(t) &: \text{gripper}(t) > 0.5 \\ \pi_{\text{release}}(t) &: \text{gripper}(t) < 0.15 \wedge \exists t' < t, \pi_{\text{hold}}(t')\end{aligned}$$

The sequential specification is defined as:

$$\exists t_1 < t_2 : \pi_{\text{hold}}(t_1) \wedge \pi_{\text{release}}(t_2),$$

corresponding to a pick-and-place interaction. For tasks where proprioception is not sufficient to identify phase of tasks, we manually annotate the videos and generate the ground truth predicates.

A.3.3 Encoder and Representation

We use the **SVD VAE** from Ctrl-World [45], a video diffusion model trained directly on DROID data. Each frame is encoded into a $4 \times 24 \times 40$ latent tensor, flattened to a 3,840-dimensional vector. Distances are computed using cosine similarity. This encoder is domain-matched to DROID and provides stronger geometric separation than general-purpose encoders such as DINOv2.

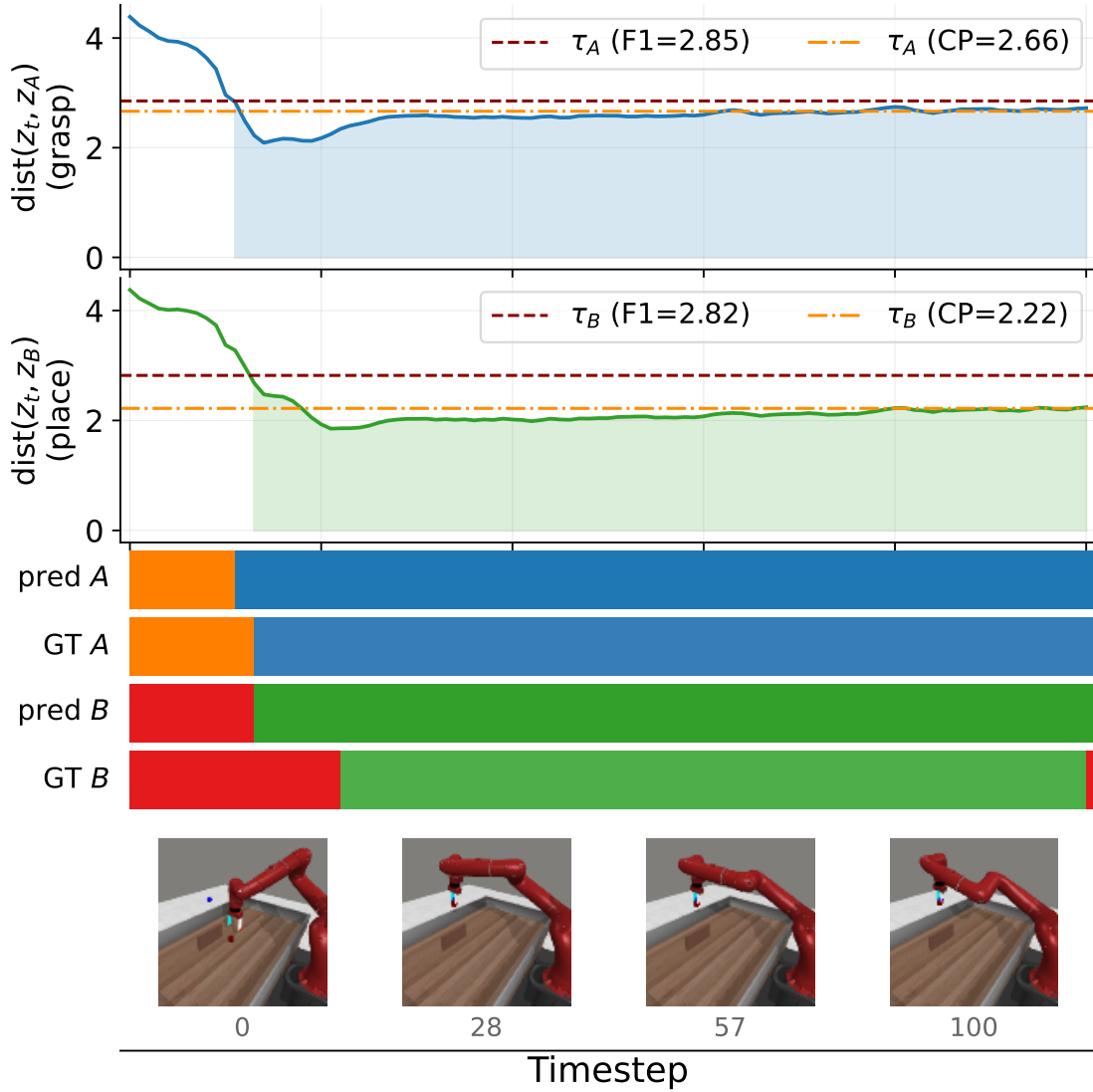


Figure A.1: Dual-predicate Boolean timelines for `mw-pick-place-wall`. Top panels show distances to grasp (z_A) and place (z_B) embeddings with thresholds ϵ^* and ϵ_{CP} . Lower panels show ETL and ground-truth predicate activations.

A.3.4 Sequential Evaluation Details

We identify 25 episodes containing valid grasp-then-release transitions, using 10 for calibration and 15 for evaluation.

The release predicate exhibits low precision due to visual ambiguity: the approach phase with an open gripper is visually similar to the post-release phase. Despite this, sequential ordering is correctly identified in all episodes.

Table A.5: Sequential predicate evaluation on DROID (SVD VAE, wrist camera).

Predicate	F1	Precision	Recall	Agreement	Seq. Agreement
π_{hold}	0.666	0.721	0.750	0.701	1.000
π_{release}	0.258	0.162	1.000	0.163	

A.3.5 Discussion of Failure Modes

ETL performance depends on the separability of task-relevant concepts in the embedding space. For manipulation tasks, success states (e.g., button press, object placement) are highly distinctive, leading to sharp transitions in embedding distance and high predicate accuracy.

Failure modes arise when embeddings corresponding to different semantic states overlap in the latent space, which can result in delayed or premature predicate activation near decision boundaries. However, such cases are rare in these tasks, and conformal thresholds help mitigate missed detections by prioritizing recall.

Overall, these qualitative results reinforce that embedding distances provide a reliable and interpretable signal for detecting semantic events and temporal structure in manipulation tasks.

We evaluate whether *Embedding Temporal Logic* (ETL) monitoring can serve as a competitive, interpretable alternative to state of the art failure predictors for generative robot policies. Our central hypothesis is that explicitly encoding the *sequential milestone structure* of a task into a small number of latent spec anchors, and then asking whether each anchor was ever approached during execution, yields a sharper failure signal than methods that either model the full observation distribution or score each timestep independently against a single goal embedding.

A.4 SafeDec Visualizations

To complement our quantitative results, we provide numerous trajectory plots from evaluations across a diverse set of procedurally generated indoor environments. Figure A.2 illustrates representative top-down visualizations of trajectories induced by SafeDec. Each plot shows the agent’s starting point and the resulting path under constrained decoding, with red circles marking target objects, green boxes denoting forbidden regions, and orange paths depicting the safe trajectories generated by our method. Together with our quantitative analysis, these qualitative results illustrate the performance of SafeDec across environments and tasks.

A.5 Results for ϕ_{geofence}



Figure A.2: Top-down views across indoor environments with SafeDec induced trajectories Red circles mark target objects, green boxes are avoid regions, and orange paths show safe trajectories under constrained decoding.

Table A.6: Performance comparison for the geofence specification $\phi_{geofence}$ across decoding techniques and base models. Each cell reports STL satisfaction / success rate (%). Higher is better ($\uparrow$).

Decoding	$\phi_{geofence}$: STL St / SR (% $\uparrow$)		
	SPOC	Flare	PoliFormer
Unconstrained	78.0 / 81.5	68.0 / 81.0	73.0 / 81.5
Filtering	100.0 / 72.0	100.0 / 66.5	100.0 / 67.5
HCD	100.0 / 76.5	100.0 / 67.5	100.0 / 72.5
RCD	95.5 / 80.0	80.0 / 71.5	85.5 / 77.5

Bibliography

- [1] AGRAWAL, S., SEO, J., NAKAMURA, K., TIAN, R., AND BAJCSY, A. Anysafe: Adapting latent safety filters at runtime via safety constraint parameterization in the latent space, 2025. 5.6.1, 5.6.1, A.1.1
- [2] AI, G. Guidance: A language model control framework. <https://github.com/guidance-ai/guidance>, 2023. Accessed: April 27, 2025. 1.1, 6
- [3] AKSARAY, D., JONES, A., KONG, Z., SCHWAGER, M., AND BELTA, C. Q-learning for robust satisfaction of signal temporal logic specifications, 2016. 2.1, 2.3, 2.6
- [4] ALOOR, J. J., PATRIKAR, J., KAPOOR, P., OH, J., AND SCHERER, S. Follow the rules: Online signal temporal logic tree search for guided imitation learning in stochastic domains. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), pp. 1320–1326. 2.1, 2.3, 3
- [5] ALPERN, B., AND SCHNEIDER, F. B. Recognizing safety and liveness. *Distributed computing* 2, 3 (1987), 117–126. 6
- [6] ALSHIEKH, M., BLOEM, R., EHLERS, R., KÖNIGHOFFER, B., NIEKUM, S., AND TOPCU, U. Safe reinforcement learning via shielding, 2017. 1.1, 2.5
- [7] ALUR, R., BASTANI, O., JOTHIMURUGAN, K., PEREZ, M., SOMENZI, F., AND TRIVEDI, A. Policy synthesis and reinforcement learning for discounted ltl. In *International Conference on Computer Aided Verification* (2023), Springer, pp. 415–435. 2.1
- [8] AMES, A. D., COOGAN, S., EGERSTEDT, M., NOTOMISTA, G., SREENATH, K., AND TABUADA, P. Control barrier functions: Theory and applications, 2019. 2.5, 2.6
- [9] ANGELOPOULOS, A. N., AND BATES, S. Conformal prediction: A gentle introduction. *Found. Trends Mach. Learn.* 16, 4 (Mar. 2023), 494–591. 5.5.2
- [10] ARECES, F., MOHRI, C., HASHIMOTO, T., AND DUCHI, J. Online conformal prediction via online optimization. In *Forty-second International Conference on Machine Learning* (2025). 5.7
- [11] BACCHUS, F., AND KABANZA, F. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22, 1–2 (Jan. 1998), 5–27. 1.1
- [12] BALAKRISHNAN, A. Signal Temporal Logic C++ Toolbox. Available at <https://github.com/anand-bala/signal-temporal-logic> (2021). 2.1
- [13] BALAKRISHNAN, A., DESHMUKH, J., HOXHA, B., YAMAGUCHI, T., AND FAINEKOS,

- G. Percemon: online monitoring for perception systems. In *Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings 21* (2021), Springer, pp. 297–308. 2.1
- [14] BALAKRISHNAN, A., AND DESHMUKH, J. V. Structured reward shaping using signal temporal logic specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), IEEE, pp. 3481–3486. 2.6
- [15] BANIODEH, M., GOEL, K., ETtinger, S., FUERTES, C., SEFF, A., SHEN, T., GULINO, C., YANG, C., JERFEL, G., CHOE, D., WANG, R., KALLEM, V., CASAS, S., AL-RFOU, R., SAPP, B., AND ANGUELOV, D. Scaling laws of motion forecasting and planning – a technical report, 2025. 5.1
- [16] BARTOCCI, E., DESHMUKH, J., DONZÉ, A., FAINEKOS, G., MALER, O., NIČKOVIĆ, D., AND SANKARANARAYANAN, S. Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In *Lectures on Runtime Verification: Introductory and Advanced Topics*, E. Bartocci and Y. Falcone, Eds., Lecture Notes in Computer Science. Springer International Publishing, Cham, 2018, pp. 135–175. 1.1, 2.1, 5.2.2
- [17] BARTOCCI, E., MANJUNATH, N., MARIANI, L., MATEIS, C., AND NIČKOVIĆ, D. CPS-Debug: Automatic failure explanation in CPS models. *Int. Journal on Software Tools for Technology Transfer* (2021). 4.5.3
- [18] BELTA, C., AND SADRADDINI, S. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems* 2, 1 (2019), 115–140. (document), 2.1, 3.3.1, 3.2
- [19] BEURER-KELLNER, L., FISCHER, M., AND VECHEV, M. Prompting is programming: A query language for large language models. *Proc. ACM Program. Lang.* 7, PLDI (June 2023). 1.1, 6
- [20] BORTOLUSSI, L., AND NENZI, L. Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools* (2014), pp. 66–73. 2.1
- [21] BOTEÀ, A., AND CIRÉ, A. A. Incremental heuristic search for planning with temporally extended goals and uncontrollable events. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 2009), IJCAI’09, Morgan Kaufmann Publishers Inc., p. 1647–1652. 3, 5.1
- [22] BRADBURY, J., FROSTIG, R., HAWKINS, P., JOHNSON, M. J., LEARY, C., MACLAURIN, D., NECULA, G., PASZKE, A., VANDERPLAS, J., WANDERMAN-MILNE, S., AND ZHANG, Q. JAX: composable transformations of Python+NumPy programs. Available at <http://github.com/google/jax> (2018). 4.5
- [23] CARDONA, G. A., LEAHY, K., MANN, M., AND VASILE, C.-I. A Flexible and Efficient Temporal LogicTool for Python: PyTeLo. Available at <https://arxiv.org/abs/2310.08714> (2023). 2.1

- [24] CARLONE, L., KIM, A., BARFOOT, T., CREMERS, D., AND DELLAERT, F., Eds. *SLAM Handbook. From Localization and Mapping to Spatial Intelligence*. Cambridge University Press, 2025. 1.1
- [25] CHEN, M., TAM, Q., LIVINGSTON, S. C., AND PAVONE, M. Signal Temporal Logic meets Hamilton-Jacobi Reachability: Connections and Applications. In *Workshop on Algorithmic Foundations of Robotics* (2018). 2.6
- [26] CHEN, Y., GANDHI, R., ZHANG, Y., AND FAN, C. NL2TL: transforming natural languages to temporal logics using large language models. *CoRR abs/2305.07766* (2023). 3.3.1
- [27] CHERTI, M., BEAUMONT, R., WIGHTMAN, R., WORTSMAN, M., ILHARCO, G., GORDON, C., SCHUHMANN, C., SCHMIDT, L., AND JITSEV, J. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 2818–2829. 1.1
- [28] COHEN, M. H., MOLNAR, T. G., AND AMES, A. D. Safety-critical control for autonomous systems: Control barrier functions via reduced-order models. *Annual Reviews in Control* 57 (2024), 100947. 6.3.1, 6.3.5
- [29] COLOMBO, C., AND PACE, G. J. What is runtime verification. In *Runtime Verification: A Hands-On Approach in Java*. Springer, 2022, pp. 9–15. 5.1
- [30] CZECHOWSKI, K., ODRZYGÓŹDŹ, T., ZBYSIŃSKI, M., ZAWALSKI, M., OLEJNIK, K., WU, Y., KUCIŃSKI, Ł., AND MIŁOŚ, P. Subgoal search for complex reasoning tasks. *Advances in Neural Information Processing Systems* 34 (2021), 624–638. 3
- [31] DECASTRO, J., LEUNG, K., ARÉCHIGA, N., AND PAVONE, M. Interpretable Policies from Formally-Specified Temporal Properties. In *Proc. IEEE Int. Conf. on Intelligent Transportation Systems* (2020). 4
- [32] DIAMOND, S., AND BOYD, S. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5. 2.4
- [33] DOKHANCHI, A., AMOR, H. B., DESHMUKH, J. V., AND FAINEKOS, G. Evaluating perception systems for autonomous vehicles using quality temporal logic. In *Runtime Verification: 18th International Conference, RV 2018, Limassol, Cyprus, November 10–13, 2018, Proceedings* 18 (2018), Springer, pp. 409–416. 2.1
- [34] DONNARUMMA, F., MAISTO, D., AND PEZZULO, G. Problem solving as probabilistic inference with subgoal: Explaining human successes and pitfalls in the tower of hanoi. *PLoS Computational Biology* 12, 4 (2016). 3.1
- [35] DOSOVITSKIY, A., BEYER, L., KOLESNIKOV, A., WEISSENBORN, D., ZHAI, X., UNTERTHINER, T., DEGHANI, M., MINDERER, M., HEIGOLD, G., GELLY, S., USZKOREIT, J., AND HOULSBY, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 1.1
- [36] EHSANI, K., GUPTA, T., HENDRIX, R., SALVADOR, J., WEIHS, L., ZENG, K.-H., SINGH, K. P., KIM, Y., HAN, W., HERRASTI, A., ET AL. Spoc: Imitating shortest paths

- in simulation enables effective navigation and manipulation in the real world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2024), pp. 16238–16250. 6, 6.3.1
- [37] FAINEKOS, G. E., AND PAPPAS, G. J. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291. 1, 1.1, 5.3.1, 15
 - [38] FAN, A., LEWIS, M., AND DAUPHIN, Y. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833* (2018). 6.1
 - [39] FARRELL, M., LUCKCUCK, M., AND FISHER, M. *Robotics and Integrated Formal Methods: Necessity Meets Opportunity*. Springer International Publishing, 2018, p. 161–171. 1, 6
 - [40] FIROOZI, R., TUCKER, J., TIAN, S., MAJUMDAR, A., SUN, J., LIU, W., ZHU, Y., SONG, S., KAPOOR, A., HAUSMAN, K., ICHTER, B., DRIESS, D., WU, J., LU, C., AND SCHWAGER, M. Foundation models in robotics: Applications, challenges, and the future, 2023. 1
 - [41] GILL, P. E., MURRAY, W., AND SAUNDERS, M. A. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.* 47, 1 (jan 2005), 99–131. 3.3.1
 - [42] GILPIN, Y., KURTZ, V., AND LIN, H. A Smooth Robustness Measure of Signal Temporal Logic for Symbolic Control. *IEEE Control Systems Letters* 5, 1 (2020), 241–246. 4
 - [43] GU, S., YANG, L., DU, Y., CHEN, G., WALTER, F., WANG, J., AND KNOLL, A. A review of safe reinforcement learning: Methods, theory and applications, 2024. 2.5
 - [44] GU, Y., HUNT, W., ARCHIBALD, B., XU, M., SEVEGNANI, M., AND SOORATI, M. D. Successful swarms: operator situational awareness with modelling and verification at run-time. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (2023), IEEE, pp. 541–548. 5.1
 - [45] GUO, Y., SHI, L. X., CHEN, J., AND FINN, C. Ctrl-world: A controllable generative world model for robot manipulation, 2026. A.3.3
 - [46] GUROBI OPTIMIZATION, INC. Gurobi optimizer reference manual, 2012. 3.3.1
 - [47] HAFNER, D., LILLICRAP, T., BA, J., AND NOROUZI, M. Dream to control: Learning behaviors by latent imagination, 2020. 5.1
 - [48] HAFNER, D., PASUKONIS, J., BA, J., AND LILLICRAP, T. Mastering diverse control tasks through world models. *Nature* 640, 8059 (Apr 2025), 647–653. 5.6.1, A.1.1
 - [49] HAGHIGHI, I., JONES, A., KONG, Z., BARTOCCI, E., GROS, R., AND BELTA, C. Spatel: a novel spatial-temporal logic and its applications to networked systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control* (2015), pp. 189–198. 2.1
 - [50] HANSEN, N., SU, H., AND WANG, X. Td-mpc2: Scalable, robust world models for continuous control, 2024. 5.1

- [51] HASHEMI, N., HOXHA, B., PROKHOROV, D., FAINEKOS, G., AND DESHMUKH, J. V. Scaling learning-based policy optimization for temporal logic tasks by controller network dropout. *ACM Transactions on Cyber-Physical Systems* 8, 4 (2025), 1–28. 1
- [52] HE, J., BARTOCCI, E., NIČKOVIĆ, D., IAKOVIC, H., AND GROSU, R. DeepSTL – From English Requirements to Signal Temporal Logic. In *IEEE/ACM Int. Conf. on Software Engineering* (2022). 4.3.1, 4.5.3
- [53] HE, T., ZHANG, C., XIAO, W., HE, G., LIU, C., AND SHI, G. Agile but safe: Learning collision-free high-speed legged locomotion. In *Robotics: Science and Systems (RSS)* (2024). 6
- [54] HEKMATNEJAD, M. *Formalizing safety, perception, and mission requirements for testing and planning in autonomous vehicles*. PhD thesis, Arizona State University, 2021. 2.1
- [55] HEKMATNEJAD, M., HOXHA, B., DESHMUKH, J. V., YANG, Y., AND FAINEKOS, G. Formalizing and evaluating requirements of perception systems for automated vehicles using spatio-temporal perception logic. *The International Journal of Robotics Research* 43, 2 (2024), 203–238. 2.1, 5.1
- [56] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation* (1997). 4
- [57] HOKAMP, C., AND LIU, Q. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vancouver, Canada, July 2017), R. Barzilay and M.-Y. Kan, Eds., Association for Computational Linguistics, pp. 1535–1546. 6.1
- [58] HOLTZMAN, A., BUYS, J., DU, L., FORBES, M., AND CHOI, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019). 6.1
- [59] HU, J., HENDRIX, R., FARHADI, A., KEMBHAVI, A., MARTÍN-MARTÍN, R., STONE, P., ZENG, K.-H., AND EHSANI, K. Flare: Achieving masterful and adaptive robot policies with large-scale reinforcement learning fine-tuning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)* (2025), IEEE, pp. 3617–3624. 6, 6.3.1
- [60] HU, Y., XIE, Q., JAIN, V., FRANCIS, J., PATRIKAR, J., KEETHA, N. V., KIM, S., XIE, Y., ZHANG, T., ZHAO, S., ET AL. Toward general-purpose robots via foundation models: A survey and meta-analysis. *CoRR* (2023). 6
- [61] HUANG, J., YONG, S., MA, X., LINGHU, X., LI, P., WANG, Y., LI, Q., ZHU, S.-C., JIA, B., AND HUANG, S. An embodied generalist agent in 3d world. In *Proceedings of the International Conference on Machine Learning (ICML)* (2024). 6
- [62] INTELLIGENCE, P., AMIN, A., ANICETO, R., BALAKRISHNA, A., BLACK, K., CONLEY, K., CONNORS, G., DARPINIAN, J., DHABALIA, K., DICARLO, J., DRIESS, D., EQUI, M., ESMAIL, A., FANG, Y., FINN, C., GLOSSOP, C., GODDEN, T., GORYACHEV, I., GROOM, L., HANCOCK, H., HAUSMAN, K., HUSSEIN, G., ICHTER, B., JAKUBCZAK, S., JEN, R., JONES, T., KATZ, B., KE, L., KUCHI, C., LAMB, M., LEBLANC, D., LEVINE, S., LI-BELL, A., LU, Y., MANO, V., MOTHUKURI, M., NAIR, S., PERTSCH,

- K., REN, A. Z., SHARMA, C., SHI, L. X., SMITH, L., SPRINGENBERG, J. T., STACHOWICZ, K., STOECKLE, W., SWERDLOW, A., TANNER, J., TORNE, M., VUONG, Q., WALLING, A., WANG, H., WILLIAMS, B., YOO, S., YU, L., ZHILINSKY, U., AND ZHOU, Z. $\pi_{0.6}^*$: a vla that learns from experience, 2025. 5.1
- [63] JANNER, M., DU, Y., TENENBAUM, J., AND LEVINE, S. Planning with Diffusion for Flexible Behavior Synthesis. In *Int. Conf. on Machine Learning* (2022). 4.5.2
- [64] JIA, X., BLESSING, D., JIANG, X., REUSS, M., DONAT, A., LIOUTIKOV, R., AND NEUMANN, G. Towards diverse behaviors: A benchmark for imitation learning with human demonstrations. In *The Twelfth International Conference on Learning Representations* (2024). 5.6.2
- [65] KANG, E., GANLATH, A., MISHRA, S., BAIDUC, F., AND AMMAR, N. Contract-Driven Runtime Adaptation. In *NASA Formal Methods Symposium* (2024). 4.5.3
- [66] KAPOOR, P., BALAKRISHNAN, A., AND DESHMUKH, J. V. Model-based reinforcement learning from signal temporal logic specifications. *arXiv preprint arXiv:2011.04950* (2020). 2.3
- [67] KAPOOR, P., GANLATH, A., CLIFFORD, M., LIU, C., SCHERER, S., AND KANG, E. Constrained decoding for robotics foundation models. *arXiv preprint arXiv:2509.01728* (2025). 3
- [68] KAPOOR, P., HAMMER, A., KAPOOR, A., LEUNG, K., AND KANG, E. Pretrained embeddings as a behavior specification mechanism, 2025. 4
- [69] KAPOOR, P., HIGGINS, I., KEETHA, N., PATRIKAR, J., MOON, B., YE, Z., HE, Y., CISNEROS, I., HU, Y., LIU, C., KANG, E., AND SCHERER, S. Demonstrating visafe: Vision-enabled safety for high-speed detect and avoid. 6
- [70] KAPOOR, P., KANG, E., AND MEIRA-GÓES, R. Safe planning through incremental decomposition of signal temporal logic specifications. In *NASA Formal Methods Symposium* (2024), Springer, pp. 377–396. 4.5.3, 1
- [71] KAPOOR, P., MIZUTA, K., KANG, E., AND LEUNG, K. Stlsg++: A masking approach for differentiable signal temporal logic specification. *IEEE Robotics and Automation Letters* 10, 9 (Sept. 2025), 9240–9247. 2.1, 6.3.1, 2
- [72] KAPOOR, P., VEMPRALA, S., AND KAPOOR, A. Logically constrained robotics transformers for enhanced perception-action planning. *arXiv preprint arXiv:2408.05336* (2024). 1.1, 6
- [73] KARAGULLE, R., ARÉCHIGA, N., BEST, A., DECASTRO, J., AND OZAY, N. A Safe Preference Learning Approach for Personalization With Applications to Autonomous Vehicles. *IEEE Robotics and Automation Letters* 9, 5 (2024), 4226–4233. 4
- [74] KHAZATSKY, A., PERTSCH, K., NAIR, S., BALAKRISHNA, A., DASARI, S., KARAMCHETI, S., NASIRIANY, S., SRIRAMA, M. K., CHEN, L. Y., ELLIS, K., FAGAN, P. D., HEJNA, J., ITKINA, M., LEPERT, M., MA, Y. J., MILLER, P. T., WU, J., BELKHALE, S., DASS, S., HA, H., JAIN, A., LEE, A., LEE, Y., MEMMEL, M., PARK, S., RADOSAVOVIC,

- I., WANG, K., ZHAN, A., BLACK, K., CHI, C., HATCH, K. B., LIN, S., LU, J., MERCAT, J., REHMAN, A., SANKETI, P. R., SHARMA, A., SIMPSON, C., VUONG, Q., WALKE, H. R., WULFE, B., XIAO, T., YANG, J. H., YAVARY, A., ZHAO, T. Z., AGIA, C., BAIJAL, R., CASTRO, M. G., CHEN, D., CHEN, Q., CHUNG, T., DRAKE, J., FOSTER, E. P., GAO, J., GUIZILINI, V., HERRERA, D. A., HEO, M., HSU, K., HU, J., IRSHAD, M. Z., JACKSON, D., LE, C., LI, Y., LIN, K., LIN, R., MA, Z., MADDUKURI, A., MIRCHANDANI, S., MORTON, D., NGUYEN, T., O’NEILL, A., SCALISE, R., SEALE, D., SON, V., TIAN, S., TRAN, E., WANG, A. E., WU, Y., XIE, A., YANG, J., YIN, P., ZHANG, Y., BASTANI, O., BERSETH, G., BOHG, J., GOLDBERG, K., GUPTA, A., GUPTA, A., JAYARAMAN, D., LIM, J. J., MALIK, J., MARTÍN-MARTÍN, R., RAMAMOORTHY, S., SADIGH, D., SONG, S., WU, J., YIP, M. C., ZHU, Y., KOLLAR, T., LEVINE, S., AND FINN, C. Droid: A large-scale in-the-wild robot manipulation dataset. 5.6.3
- [75] KIM, M., PERTSCH, K., KARAMCHETI, S., XIAO, T., BALAKRISHNA, A., NAIR, S., RAFAILOV, R., FOSTER, E., LAM, G., SANKETI, P., VUONG, Q., KOLLAR, T., BURCHFIELD, B., TEDRAKE, R., SADIGH, D., LEVINE, S., LIANG, P., AND FINN, C. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246* (2024). 5.1
- [76] KOLVE, E., MOTTAGHI, R., HAN, W., VANDERBILT, E., WEIHS, L., HERRASTI, A., DEITKE, M., EHSANI, K., GORDON, D., ZHU, Y., KEMBHAVI, A., GUPTA, A., AND FARHADI, A. Ai2-thor: An interactive 3d environment for visual ai, 2022. 6.3.1
- [77] KOYMANS, R. Specifying real-time properties with metric temporal logic. *Real-time systems* 2, 4 (1990), 255–299. 2.1
- [78] KRESS-GAZIT, H., FAINEKOS, G. E., AND PAPPAS, G. J. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics* 25, 6 (2009), 1370–1381. 2.1
- [79] KURTZ, V., AND LIN, H. Mixed-integer programming for signal temporal logic with fewer binary variables. *IEEE Control Systems Letters* (2022). (document), 1.1, 2.3, 2.1, 3.1, 3.3.1, 3.3.1, 3.3.1, 3.2
- [80] LEAHY, K., MANN, M., AND VASILE, C.-I. Rewrite-based decomposition of signal temporal logic specifications. In *NASA Formal Methods* (Cham, 2023), K. Y. Rozier and S. Chaudhuri, Eds., Springer Nature Switzerland, pp. 224–240. 2.3, 3.2.1, 3.2.1
- [81] LEI, J., G’SSELL, M., RINALDO, A., TIBSHIRANI, R. J., AND WASSERMAN, L. Distribution-free predictive inference for regression, 2017. 5.5.2
- [82] LEUNG, K., ARÉCHIGA, N., AND PAVONE, M. Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. *The International Journal of Robotics Research* (2023), 02783649221082115. 2.1, 2.1, 3.1, 4, 4.1, 4.1.5, 4.2.2
- [83] LEUNG, K., AND PAVONE, M. Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications. In *American Control Conference* (2022). 4
- [84] LIN, Z., AND BARAS, J. S. Planning and runtime monitoring of robotic manipulator using metric interval temporal logic. In *2019 IEEE International Systems Conference (SysCon)*

- (2019), pp. 1–8. 5.1
- [85] LINDEMANN, L., AND DIMAROGONAS, D. V. Control barrier functions for signal temporal logic tasks. *IEEE Control Systems Letters* 3 (2019), 96–101. 2.1, 2.3
 - [86] LIU, A., SAP, M., LU, X., SWAYAMDIPTA, S., BHAGAVATULA, C., SMITH, N. A., AND CHOI, Y. Dexperts: Decoding-time controlled text generation with experts and anti-experts, 2021. 6.2.3
 - [87] LIU, H., ZHANG, Y., BETALA, V., ZHANG, E., LIU, J., DING, C., AND ZHU, Y. Multi-task interactive robot fleet learning with visual world models, 2024. 5.6.2
 - [88] LIU, W., MEHDIPOUR, N., AND BELTA, C. Recurrent Neural Network Controllers for Signal Temporal Logic Specifications Subject to Safety Constraints. *IEEE Control Systems Letters* 6 (2021), 91 – 96. 4
 - [89] LIU, W., NISHIOKA, M., AND BELTA, C. Safe Model-based Control from Signal Temporal Logic Specifications Using Recurrent Neural Networks. In *Proc. IEEE Conf. on Robotics and Automation* (2023). 4
 - [90] LIU, W., XIAO, W., AND BELTA, C. Learning Robust and Correct Controllers from Signal Temporal Logic Specifications Using BarrierNet. In *Proc. IEEE Conf. on Decision and Control* (2023). 4
 - [91] MA, M., GAO, J., FENG, L., AND STANKOVIC, J. STLnet: Signal Temporal Logic Enforced Multivariate Recurrent Neural Networks. In *Conf. on Neural Information Processing Systems* (2020). 4
 - [92] MALER, O., AND NICKOVIC, D. Monitoring Temporal Properties of Continuous Signals. *Lecture Notes in Computer Science*, 3253 (2004), 152–166. 1, 2.1, 2.2, 4.1, 5.1, 5.2.2, 5.3.1
 - [93] MANGANARIS, A., GIAMMARINO, V., QURESHI, A. H., AND JAGANNATHAN, S. Formal methods in robot policy learning and verification: A survey on current techniques and future directions. *Transactions on Machine Learning Research* (2026). 6
 - [94] MEHDIPOUR, N., VASILE, C.-I., AND BELTA, C. Arithmetic-Geometric Mean Robustness for Control from Signal Temporal Logic Specifications. In *American Control Conference* (2019). 4.1.2
 - [95] MENG, Y., AND FAN, C. Diverse Controllable Diffusion Policy With Signal Temporal Logic. *IEEE Robotics and Automation Letters* 9, 10 (2024), 8354–8361. 4
 - [96] MENG, Y., AND FAN, C. Telograf: Temporal logic planning via graph-encoded flow matching. In *Forty-second International Conference on Machine Learning* (2025). 1, 1.1
 - [97] MENGHI, C., TSIGKANOS, C., PELLICCIONE, P., GHEZZI, C., AND BERGER, T. Specification patterns for robotic missions. *IEEE Transactions on Software Engineering* 47, 10 (2021), 2208–2224. 1, 3.2, 3.3.1, 6, 6.3.1
 - [98] MIZUTA, K., AND LEUNG, K. CoBL-Diffusion: Diffusion-Based Conditional Robot Planning in Dynamic Environments Using Control Barrier and Lyapunov Functions. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems* (2024). 4.5.2

- [99] NAIR, S., AND FINN, C. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *ArXiv abs/1909.05829* (2019). 3
- [100] NICKLAS HANSEN, HAO SU, X. W. Learning massively multitask world models for continuous control, 2026. 5.6.2
- [101] NVIDIA, :, ALI, A., BAI, J., BALA, M., BALAJI, Y., BLAKEMAN, A., CAI, T., CAO, J., CAO, T., CHA, E., CHAO, Y.-W., CHATTOPADHYAY, P., CHEN, M., CHEN, Y., CHEN, Y., CHENG, S., CUI, Y., DIAMOND, J., DING, Y., FAN, J., FAN, L., FENG, L., FERRONI, F., FIDLER, S., FU, X., GAO, R., GE, Y., GU, J., GUPTA, A., GURURANI, S., HANAFI, I. E., HASSANI, A., HAO, Z., HUFFMAN, J., JANG, J., JANNATY, P., KAUTZ, J., LAM, G., LI, X., LI, Z., LIAO, M., LIN, C.-H., LIN, T.-Y., LIN, Y.-C., LING, H., LIU, M.-Y., LIU, X., LU, Y., LUO, A., MA, Q., MAO, H., MO, K., NAH, S., NARANG, Y., PANASKAR, A., PAVAO, L., PHAM, T., RAMEZANALI, M., REDA, F., REED, S., REN, X., SHAO, H., SHEN, Y., SHI, S., SONG, S., STEFANIAK, B., SUN, S., TANG, S., TASMEEN, S., TCHAPMI, L., TSENG, W.-C., VARGHESE, J., WANG, A. Z., WANG, H., WANG, H., WANG, H., WANG, T.-C., WEI, F., XU, J., YANG, D., YANG, X., YE, H., YE, S., ZENG, X., ZHANG, J., ZHANG, Q., ZHENG, K., ZHU, A., AND ZHU, Y. World simulation with video foundation models for physical ai, 2026. 5.1
- [102] OQUAB, M., DARCEY, T., MOUTAKANNI, T., VO, H., SZAFRANIEC, M., KHALIDOV, V., FERNANDEZ, P., HAZIZA, D., MASSA, F., EL-NOUBY, A., ASSRAN, M., BALLAS, N., GALUBA, W., HOWES, R., HUANG, P.-Y., LI, S.-W., MISRA, I., RABBAT, M., SHARMA, V., SYNNAEVE, G., XU, H., JEGOU, H., MAIRAL, J., LABATUT, P., JOULIN, A., AND BOJANOWSKI, P. Dinov2: Learning robust visual features without supervision, 2024. 5.1, 5.2.1
- [103] PANT, Y. V., ABBAS, H., AND MANGHARAM, R. Smooth Operator: Control using the Smooth Robustness of Temporal Logic. In *IEEE Conf. on Control Technology and Applications* (2017). 4
- [104] PARK, K., WANG, J., BERG-KIRKPATRICK, T., POLIKARPOVA, N., AND D'ANTONI, L. Grammar-aligned decoding. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2024), NIPS '24, Curran Associates Inc. 6.2.1, 6.2.2
- [105] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic differentiation in PyTorch. In *Conf. on Neural Information Processing Systems - Autodiff Workshop* (2017). 4.5
- [106] PEYRARD, M., JOSIFOSKI, M., AND WEST, R. The era of semantic decoding, 2024. 6.1
- [107] PNUELI, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (1977), pp. 46–57. 1, 2.1
- [108] RADFORD, A., KIM, J. W., HALLACY, C., RAMESH, A., GOH, G., AGARWAL, S., SAS-TRY, G., ASKELL, A., MISHKIN, P., CLARK, J., ET AL. Learning transferable visual models from natural language supervision. In *International conference on machine learning* (2021), PMLR, pp. 8748–8763. 5.1, 5.2.1

- [109] RAMAN, V., DONZÉ, A., MAASOUMY, M., MURRAY, R. M., SANGIOVANNI-VINCENTELLI, A., AND SESHIA, S. A. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control* (2014), pp. 81–87. 1.1, 2.1, 2.3, 3.1
- [110] SADIGH, D. *Safe and Interactive Autonomy: Control, Learning, and Verification*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2017. 1.1
- [111] SADRADDINI, S., AND BELTA, C. Formal synthesis of control strategies for positive monotone systems. *IEEE Transactions on Automatic Control* 64, 2 (2019), 480–495. 2.3, 3.1
- [112] SCHÖN, O., AND LINDEMANN, L. Spatiotemporal robustness of temporal logic tasks using multi-objective reasoning. *arXiv preprint arXiv:2603.29868* (2026). 5.1
- [113] SERMANET, P., MAJUMDAR, A., IRPAN, A., KALASHNIKOV, D., AND SINDHWANI, V. Generating robot constitutions & benchmarks for semantic safety. *arXiv preprint arXiv:2503.08663* (2025). 2.5
- [114] SESHIA, S. A., DESAI, A., DREOSSI, T., FREMONT, D. J., GHOSH, S., KIM, E., SHIVAKUMAR, S., VAZQUEZ-CHANLATTE, M., AND YUE, X. Formal specification for deep neural networks. In *Automated Technology for Verification and Analysis* (2018), pp. 20–34. 5.1
- [115] SHA, L. Using simplicity to control complexity. *IEEE Software* 18, 4 (2001), 20–28. 6.3.2
- [116] SONG, J., MENG, C., AND ERMON, S. Denoising Diffusion Implicit Models. In *Int. Conf. on Learning Representations* (2021). 4.5.2
- [117] SUN, D., CHEN, J., MITRA, S., AND FAN, C. Multi-agent motion planning from signal temporal logic specifications. *IEEE Robotics and Automation Letters PP* (2022), 1–1. 1, 1.1, 2.1, 2.3
- [118] TEDRAKE, R., AND THE DRAKE DEVELOPMENT TEAM. Drake: Model-based design and verification for robotics, 2019. 2.4, 3.3.1
- [119] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is All You Need. In *Conf. on Neural Information Processing Systems* (2017). 4, 4.2
- [120] VAZQUEZ-CHANLATTE, M. pySTLToolbox. Available at <https://github.com/mvcisback/py-signal-temporal-logic> (2022). 2.1
- [121] VEER, S., LEUNG, K., COSNER, R., CHEN, Y., AND PAVONE, M. Receding Horizon Planning with Rule Hierarchies for Autonomous Vehicles. In *Proc. IEEE Conf. on Robotics and Automation* (2023). 4
- [122] VEER, S., SHARMA, A., AND PAVONE, M. Multi-Predictor Fusion: Combining Learning-based and Rule-based Trajectory Predictors. In *Conf. on Robot Learning* (2023). 4
- [123] VOVK, V., GAMMERMAN, A., AND SHAFER, G. *Algorithmic Learning in a Random World*. Springer-Verlag, Berlin, Heidelberg, 2005. 5.2.3
- [124] WANG, P., BAI, S., TAN, S., WANG, S., FAN, Z., BAI, J., CHEN, K., LIU, X., WANG,

- J., GE, W., FAN, Y., DANG, K., DU, M., REN, X., MEN, R., LIU, D., ZHOU, C., ZHOU, J., AND LIN, J. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution, 2024. 5.6.3
- [125] WANG, Y., BOUNOU, O., ZHOU, G., BALESTRIERO, R., RUDNER, T. G. J., LECUN, Y., AND REN, M. Temporal straightening for latent planning, 2026. 5.7, 7
- [126] WELLECK, S., BERTSCH, A., FINLAYSON, M., SCHOELKOPF, H., XIE, A., NEUBIG, G., KULIKOV, I., AND HARCHAOUI, Z. From decoding to meta-generation: Inference-time algorithms for large language models, 2024. 6.1, 6.2.2
- [127] WILLARD, B. T., AND LOUF, R. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702* (2023). 1.1, 6
- [128] WU, Y., XIONG, Z., HU, Y., IYENGAR, S. S., JIANG, N., BERA, A., TAN, L., AND JAGANNATHAN, S. Selp: Generating safe and efficient task plans for robot agents with large language models, 2025. 2.5
- [129] XU, C., NGUYEN, T. K., DIXON, E., RODRIGUEZ, C., MILLER, P., LEE, R., SHAH, P., AMBRUS, R., NISHIMURA, H., AND ITKINA, M. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies, 2025. 5.6.2
- [130] XU, S., LUO, X., HUANG, Y., LENG, L., LIU, R., AND LIU, C. Nl2hltl2plan: Scaling up natural language understanding for multi-robots through hierarchical temporal logic task specifications. *IEEE Robotics and Automation Letters* (2025). 1.1
- [131] YAGHOUBI, S., AND FAINEKOS, G. Worst-case satisfaction of STL specifications using feedforward neural network controllers: A lagrange multipliers approach. *ACM Transactions on Embedded Computing Systems* 18, 5s. 4
- [132] YAMAGUCHI, T., HOXHA, B., AND NICKOVIC, D. RTAMT: Runtime Robustness Monitors with Application to CPS and Robotics. *Int. Journal on Software Tools for Technology Transfer* 26 (2023), 79–99. 2.1, 2.4
- [133] YE, S., GE, Y., ZHENG, K., GAO, S., YU, S., KURIAN, G., INDUPURU, S., TAN, Y. L., ZHU, C., XIANG, J., MALIK, A., LEE, K., LIANG, W., RANAWAKA, N., GU, J., XU, Y., WANG, G., HU, F., NARAYAN, A., BJORCK, J., WANG, J., KIM, G., NIU, D., ZHENG, R., XIE, Y., WU, J., WANG, Q., JULIAN, R., XU, D., DU, Y., CHEBOTAR, Y., REED, S., KAUTZ, J., ZHU, Y., FAN, L. J., AND JANG, J. World action models are zero-shot policies, 2026. 5.1
- [134] YU, X., WANG, C., YUAN, D., LI, S., AND YIN, X. Model predictive control for signal temporal logic specifications with time interval decomposition, 2022. 2.3
- [135] YUN, K. S., CHEN, R., DUNAWAY, C., DOLAN, J. M., AND LIU, C. Safe control of quadruped in varying dynamics via safety index adaptation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)* (2025), IEEE, pp. 7771–7777. 6
- [136] ZARLENGA, M. E., BARBIERO, P., CIRAVEGNA, G., MARRA, G., GIANNINI, F., DILIGENTI, M., SHAMS, Z., PRECIOSO, F., MELACCI, S., WELLER, A., LIO, P., AND JAMNIK, M. Concept embedding models: Beyond the accuracy-explainability trade-off, 2022.

- [137] ZENG, K.-H., ZHANG, Z., EHSANI, K., HENDRIX, R., SALVADOR, J., HERRASTI, A., GIRSHICK, R., KEMBHAVI, A., AND WEIHS, L. Poliformer: Scaling on-policy rl with transformers results in masterful navigators. In *Conference on Robot Learning* (2025), PMLR, pp. 408–432. 6, 6.3.1
- [138] ZHANG, B., ZHANG, Y., JI, J., LEI, Y., DAI, J., CHEN, Y., AND YANG, Y. Safevla: Towards safety alignment of vision-language-action model via safe reinforcement learning, 2025. 2.5, 6.3.4
- [139] ZHANG, C., KAPOOR, P., DARDIK, I., CUI, L., MEIRA-GOES, R., GARLAN, D., AND KANG, E. Constrained ltl specification learning from examples. *arXiv preprint arXiv:2412.02905* (2024). 2.1, 7
- [140] ZHANG, C., KAPOOR, P., MEIRA-GÓES, R., GARLAN, D., KANG, E., GANLATH, A., MISHRA, S., AND AMMAR, N. Tolerance of reinforcement learning controllers against deviations in cyber physical systems. In *International Symposium on Formal Methods* (2024), Springer, pp. 267–285. 2.1
- [141] ZHAO, W., SUN, Y., LI, F., CHEN, R., LIU, R., WEI, T., AND LIU, C. GUARD: A safe reinforcement learning benchmark. *Transactions on Machine Learning Research* (2024). 6
- [142] ZHONG, Z., REMPE, D., XU, D., CHEN, Y., VEER, S., CHE, T., RAY, B., AND PAVONE, M. Guided Conditional Diffusion for Controllable Traffic Simulation. In *Proc. IEEE Conf. on Robotics and Automation* (2023). 4
- [143] ZHOU, G., PAN, H., LECUN, Y., AND PINTO, L. Dino-wm: World models on pre-trained visual features enable zero-shot planning, 2025. 5.1