Specification-Driven Planning for Safe Autonomy

Parv Kapoor

October 2025

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.

Keywords: Specification-driven planning, Signal Temporal Logic, Embedding Temporal Logic, Constrained Decoding

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.

Contents

List of Figures

List of Tables

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 [30]. 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 [29, 74]. Specifically, temporal logics such as *linear temporal logic (LTL)* [77] and *signal temporal logic (STL)* [69] 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 [28]. Recent work has explored integrating these logics into learning-based frameworks [37, 72, 87], 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 [72, 87, 96] 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 [8]. 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 [80], 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 [58]. 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 [28] 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 [18]. 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 [82].

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 and text that are processed by pretrained modern vision and text transformers such as ViT [26] and CLIP [20]. 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 planning with image goals as is often the case for ImageNav tasks [107]. Additionally, this also allows 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, retraining or fine-tuning these large pre-trained models to directly embed temporal logic specification is challenging [52]. 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 [5, 11]. 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.

Solution: 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 [1,14,94] 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 [14, 94]. We extend the constrained decoding paradigm to enforce STL specifications on candidate action trajectories and propose *specification aligned decoding (SpecDec)*. 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. SpecDec reduces the risk of infeasible outputs while preserving the original action distribution of the model.

1.2 Thesis

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 List of Contributions

The expected 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.

Scope	Description	Est. Time
Must-Have	A technique to decompose STL specifications	Completed
	An efficient deep learning library for STL evaluation	Completed
	Constrained decoding for invariant STL specifications	Completed
	Embedding based temporal logic over image features	4 months
May-Have	Constrained decoding for conditional STL specifications	1 month
	Embedding based temporal logic over textual encodings	1 month
	Evaluating ETL for real world applications	1 month

Table 1.1: Scope and current progress of this work. Green rows denote completed tasks; red rows denote ongoing or planned ones.

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 Proposed Timeline

Table 1.1 summarizes the planned and completed milestones, where *Must-Have* are the essential components that must be delivered in this work and *May-Have* are the components that I would potentially deliver. So far, I have completed three *Must-Have* contributions:

1. **STL Decomposition**, an approach for decomposing complex specifications.
2. **STLCG++**, an efficient deep learning library for STL evaluation.
3. **Specification Constrained Decoding**, for enforcing invariant STL specifications in RFMs.

Next, I will focus on **Embedding Temporal Logic (ETL)** over image features (4 months), extending specifications to perception-based inputs. This would address the generality challenge, as highlighted earlier.

We also propose three stretch goals beyond the primary objectives of this work. The three *May-Have* milestones that extend beyond the core milestones are:

1. **Conditional constrained decoding** extending constrained decoding to enforce conditional STL specifications for context-aware enforcement.
2. **ETL over textual embeddings** generalizing ETL for natural language task specifications.
3. **Real-world ETL evaluation** deploying ETL for runtime monitoring and behavior synthesis for robotic use cases.

1.5 Outline

The remainder of the proposal 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 proposed solutions that addresses the respective challenges. Chapter 3 investigates a decomposition technique for translating complex long horizon STL specifications into a set of reachbility and invariance constraints. Chapter 4 investigates a technique for improving STL robustness evaluation using computational graphs. Chapter 5 discusses a technique for runtime enforcement of temporal logic requirements in modern robotic foundation models. Chapter 6 discusses proposed design and application of a novel embedding space logic called ETL for high-level specification and monitoring.

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)* [77], *metric temporal logic (MTL)* [56], and *signal temporal logic (STL)* [69] provide a precise way to encode objectives that are expressed in a natural language. These logics have been used for trajectory planning [57, 62, 87], reinforcement learning [2, 3, 6, 104], runtime monitoring [12, 103], and adaptive control [13, 50, 64, 79].

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 [15, 36]. Specialized logics such as Timed Quality Temporal Logic (TQTL) [25] and Spatio-Temporal Quality Logic (STQL) [10, 40] have been proposed for perception systems that permit reasoning about properties over bounding boxes used in object detection. Recently, Spatiotemporal Perception Logic (STPL) [41] 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 [69]. A signal $\mathbf{s}$ is a function $\mathbf{s} : \mathbb{T} \to \mathbb{R}^n$ that maps a time domain $T \subseteq \mathbb{R}_{\geq 0}$ to a real valued vector. Then, an STL formula is defined as:

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

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 $\mathscr{U}$ 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* ($F_{[a,b]}\ \phi := \top\ \mathscr{U}_{[a,b]}\ \phi$) 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:*

$$\begin{aligned}
s_t \models \mu &\iff \mu(s(t)) > 0 \\
s_t \models \neg\varphi &\iff \neg(s_t \models \varphi) \\
s_t \models \varphi_1 \wedge \varphi_2 &\iff (s_t \models \varphi_1) \wedge (s_t \models \varphi_2) \\
s_t \models \mathrm{F}_{[a,b]}(\varphi) &\iff \exists t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \varphi \\
s_t \models \mathrm{G}_{[a,b]}(\varphi) &\iff \forall t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \varphi
\end{aligned}$$

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:*

$$\begin{aligned}
\rho(s_t, \mu_c) &= \mu(x_t) - c \\
\rho(s_t, \neg\varphi) &= -\rho(s_t, \varphi) \\
\rho(s_t, \varphi_1 \wedge \varphi_2) &= \min(\rho(s_t, \varphi_1), \rho(s_t, \varphi_2)) \\
\rho(s_t, \mathrm{F}_{[a,b]}(\varphi)) &= \max_{t' \in [t+a, t+b]} \rho(s'_t, \varphi) \\
\rho(s_t, \mathrm{G}_{[a,b]}(\varphi)) &= \min_{t' \in [t+a, t+b]} \rho(s'_t, \varphi)
\end{aligned}$$

For example, suppose that we are given (1) $\phi \equiv \mathrm{G}_{[0,3]}(\texttt{distToR3}(t) \geq 3.0)$, 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`. Evaluting 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 [2, 3, 64, 81, 83]. One of the first papers in this direction involved translating STL specifications into constraints within a Mixed Integer Linear Program (MILP) [81]. 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 [87] and reducing binary variables via logarithmic encoding [58]. 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 involve using reinforcement learning [2, 45], search-based techniques [4] and control barrier functions [64] to generate

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

Name	AutoDiff	Vectorize	GPU
`STLCG` [61]	PyTorch	✓	✓
Argus [9]	✗	✗	✗
stlpy [59]	✗	✗	✗
PyTeLo [17]	✗	✗	✗
pySTL [91]	✗	✗	✗
RTAMT [98]	✗	✗	✗

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 [60, 99]. In [99], the authors restrict themselves to an STL fragment that does not allow nesting of temporal operators. In [60], 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 [24], Drake [88]) is often required.

RTAMT [98] 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) [7], safe reinforcement learning [33], and temporal logic-based shielding approaches [5]. 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 [102] 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 [84] 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 [95] 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.

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 subtasks. It has been shown that incremental subtask planning can be done more efficiently compared to planning for a composite task [16, 22, 75]. 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 [3]. 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

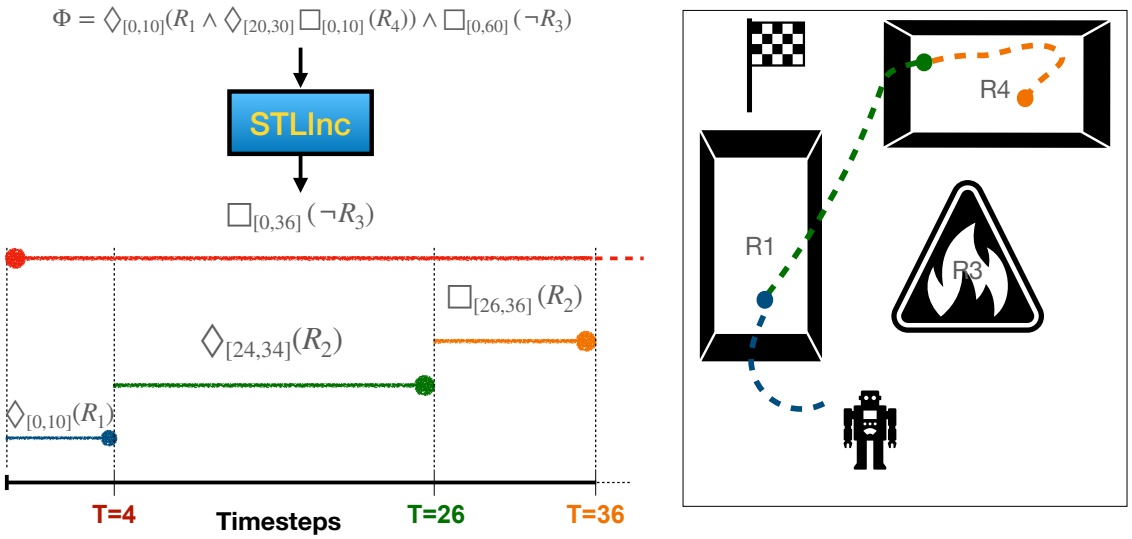

Figure 3.1: **Left**: An STL specification ϕ with multiple nested temporal operators and a possible decomposition into subtasks. **Right**: A sample trajectory that satisfies ϕ in a planar environment.

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 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:

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, $\mathscr{X}^{\exists}$ and $\mathscr{X}^{\forall}$, 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, $\mathscr{X}^{\exists}$ and $\mathscr{X}^{\forall}$, and generates a sequence of *atomic tasks*, $\sigma = \langle at_0, at_1 ... at_k \rangle$, 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 $\mathscr{X}^{\exists}$ and $\mathscr{X}^{\forall}$, thus satisfying the original goal of ϕ.

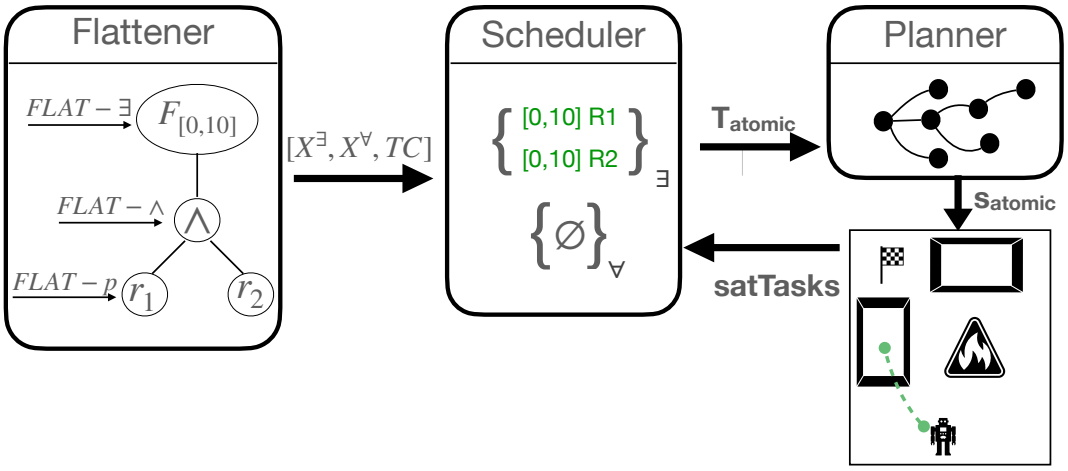

Figure 3.2: Overview of the STLINC approach

	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.

3.2 Evaluation

3.2.1 Experimental Setup

Specifications.

We investigated multiple motion planning STL specifications from [19, 74]. 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 [58]. Please refer to [58] 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 [58]. 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 [34], SNOPT [31], 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 tool[1] while using stlpy and Drake [89] 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 [13] (which we call *standard MICP*) and [58] (*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 paper are:

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

3.2.2 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 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

[1] https://github.com/parvkpr/MCTSTL

Spec	N	D	Solve time (s)					Robustness				
			LinEnv			NonLinEnv		LinEnv			NonLinEnv	
			[13]	[58]	STLINC	[58]	STLINC	[13]	[58]	STLINC	[58]	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 ([13]) and reduced MICP ([58]).

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.

3.3 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 is equipped with *quantitative* semantics, or *robustness formulas*, which measure how well a given robot trajectory satisfies a requirement. 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 [32, 76], deep learning [68, 71], and control synthesis [65, 97]. Recently, `STLCG` [61] 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 [23, 53, 63, 66, 67, 92, 93, 106].

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) [42] 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 [90], 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

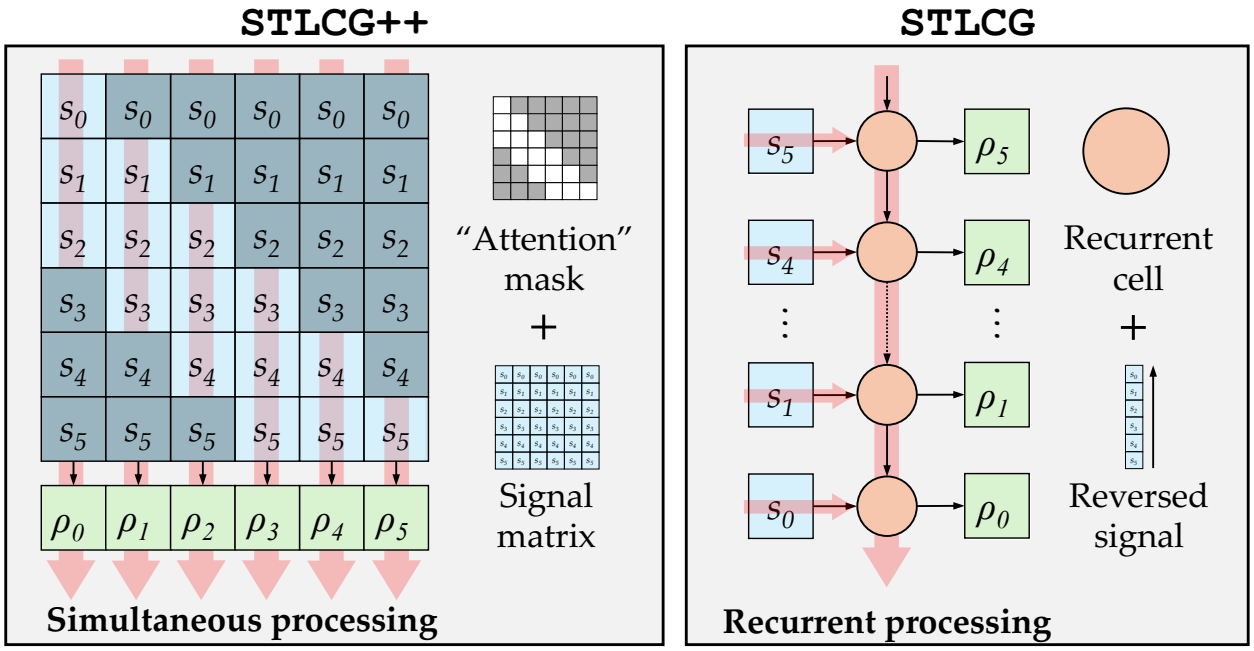

Figure 4.1: We propose STLCG++ a masking approach to evaluating and backpropagating through signal temporal logic (STL) robustness formulas. The masking approach offers stronger computational, theoretical, and practical benefits compared to STLCG, a recurrent approach.

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 Recurrent v/s Masking based Approach

STLCG's primary performance bottleneck stemmed from it's 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

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	$\mathbf{F}_I(\varphi_9 \wedge \psi_9) \wedge \ldots \wedge \mathbf{F}_I(\varphi_0 \wedge \psi_0)$

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 [90], we introduce a mask M to select relevant parts of a signal that can be later processed simultaneously rather than sequentially.

4.2 Evaluation

We analyze the computational properties of the approaches `STLCG++` (masking-based) and `STLCG` (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 `STLCG++` compute robustness traces faster than `STLCG` as measured by median computation time?

RQ2: How does `STLCG++`'s computation time scale with sequence length compared to `STLCG`?

We perform experiments on CPU and GPU. Since `STLCG++` (masking) involves large matrix computations, we anticipate `STLCG++` to scale favorably on a GPU. As `STLCG` (recurrent) utilizes a recurrent structure, it scales with sequence length. Tab. 4.1 describes six different STL formulas with varying complexities [38] 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. 4.2 and Tab. 4.2 and make the following observations.

CPU backend. We observe that `STLCG++` generally achieves lower computation times than `STLCG`. The exception is in ϕ_3, where `STLCG++` is slower than `STLCG` 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 `STLCG++` 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 `STLCG` the computation time increase is expected since the time complexity is $\mathcal{O}(T)$. For `STLCG++`, 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. `STLCG++` outperforms `STLCG` for all formulas. Additionally, `STLCG++` exhibits essentially constant computation time except for ϕ_3, which has $\mathcal{O}(T^3)$ space complexity. From

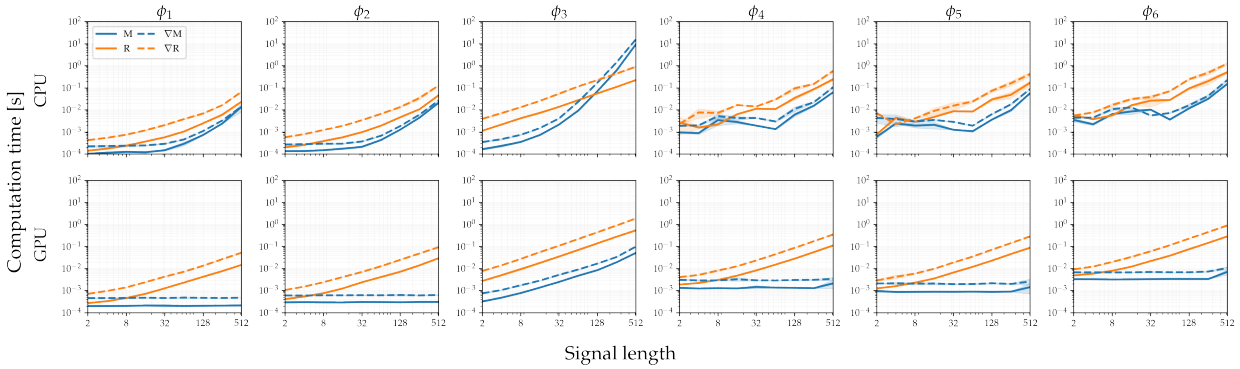

Figure 4.2: Comparison of computation time using PyTorch for masking (M, blue) and recurrent (R, orange) approach on CPU (top row) and GPU (bottom row) as signal length increases, across six STL formulas. Solid lines denote robustness computation; dashed lines denote gradient evaluation.

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**	**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%

Tab. 4.2, we see that **`STLCG++` on GPU provides around 95% and 85% improvement on JAX and PyTorch, respectively**. For `STLCG`, moving from CPU to GPU gives virtually the same scaling, with no significant improvement/reduction in computation times.

4.3 Summary

This chapter highlights `STLCG++`, a masking-based approach for computing STL robustness using automatic differentiation libraries. `STLCG++` mimics the operations that underpin transformer architectures, and outperforms the proposed masking approach over `STLCG`, which uses a recurrent approach. `STLCG++` offers significant computational advantages over `STLCG`, 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

Constrained Decoding for Foundation Models

Recent advances in developing large transformer-based models for robotics have enabled general-purpose policies that map multi-modal inputs such as RGB images, natural language instructions, and proprioceptive inputs to action sequences [44]. Shortest Path Oracle Clone (SPOC) [27], PoliFormer [101], Flare [43] and OpenVLA [54] exhibit impressive generalization in navigation and manipulation tasks and serve as versatile robot controllers for real-world deployment contexts. However, these models are primarily data-driven and lack any explicit notion of safety. Although these models may implicitly exhibit safety-related behaviors depending on the patterns in their training data, there is no formal guarantee that models will consistently behave safely in all situations. This serves as a limiting factor for deploying these foundation models in the physical world where rule compliance and regulatory safety rule adherence are crucial.

Formal specifications have long been used to specify safety requirements for robotic deployments [29, 73]. Temporal logics (TL) can capture safety constraints on robot behavior, such as "remain within the permitted region zones and avoid dangerous obstacles". Although TL has seen success in classical robotic planning for safety constraint satisfaction, its use for enforcing safety for large transformer-based robot policies remains limited. Additionally, retraining or fine-tuning these large pre-trained models to directly embed temporal logic specification is challenging [52]. 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 safety constraints through training alone. Hence, there is a pressing need for methods that can enforce safety specifications efficiently at inference time without disrupting the model's pre-trained behavior.

In the field of natural language processing, syntactic constraints have been successfully enforced by applying constrained decoding at inference time [1, 14, 94]. 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 [14, 94]. Inspired by this line of work, we extend the paradigm of constrained decoding to enforce safety constraints over action trajectories in dynamical systems and propose *safety specification aligned decoding (`SpecDec`)* for

transformer based policies that ensures generated action sequences provably satisfy Signal Temporal Logic (STL) [70] 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. `SpecDec` reduces risk of infeasible outputs while preserving the original action distribution of the model. To enforce such 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. To efficiently evaluate STL specifications at inference time, we use `STLCG++` [51].

5.1 Specification-Guided Constrained Decoding

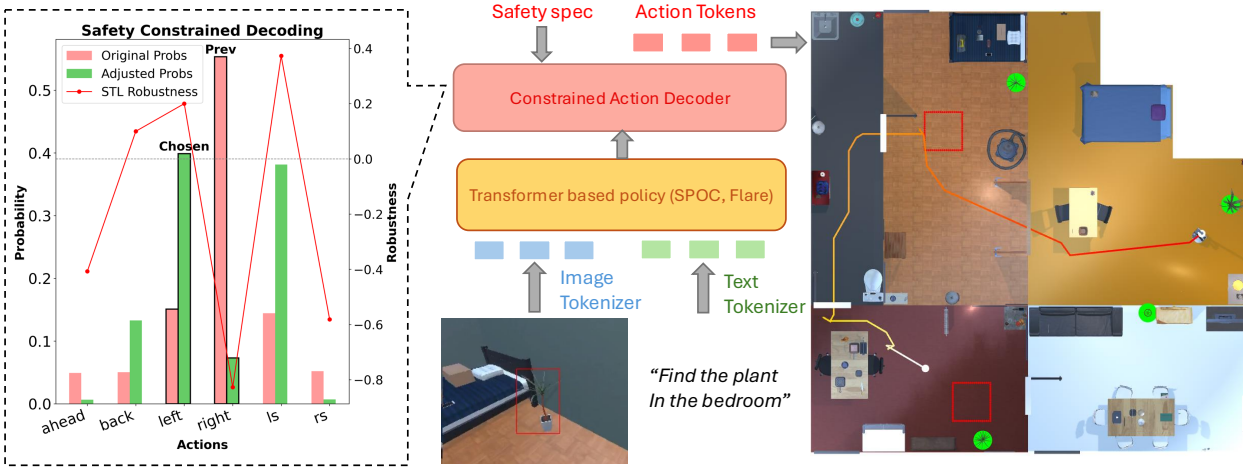

Figure 5.1: Overview of our specification aligned decoding framework. Given multimodal inputs (e.g., RGB images and natural language instructions), a pretrained transformer-based robot policy (e.g., SPOC) generates candidate actions. These actions are filtered or reweighted by the constrained decoding technique based on STL safety constraints. In the figure, green markers denote target object locations, while red zones represent regions to avoid.

Our proposed constrained decoding framework (`SpecDec`) for autoregressive transformer-based robot policies operates at the final output layer. Operating entirely at inference time, `SpecDec` 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 `SpecDec` a plug-and-play solution for ensuring specification-compliant robot behavior across diverse transformer-based policies.

5.2 Evaluation

5.2.1 Implementational Details

We comprehensively evaluate our constrained decoding framework on procedurally generated AI2-THOR [55] indoor scenes with diverse objects and layouts using three state-of-the-art (SOTA) generalist robot policies: Shortest Path Oracle Clone (SPOC) [27], PoliFormer [101] and Flare [43]. All three are large transformer-based embodied agents 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 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 `SpecDec` across different learning regimes.

In this work, we address safety specifications for robotics and, therefore, select those most relevant to real-world deployment. In particular, we focus on an important class of safety specifications called *invariants*, which are specifications that must be enforced at every reachable state of the system (e.g., "always avoid an unsafe region"). We enforce *geofencing* and *obstacle avoidance* by encoding them as invariant specifications in STL. Specifically, we generate random regions in the configuration space that the robot must either avoid (obstacle zones) or remain within (safe zones), and apply these constraints in real time during execution. The specifications used are: $\varphi_{\text{geofence}} = \mathbf{G}\left(\bigvee_{i=1}^{N}\left(x_i^{\text{L}} \leq x \leq x_i^{\text{U}} \wedge z_i^{\text{L}} \leq z \leq z_i^{\text{U}}\right)\right)$, $\varphi_{\text{avoid}} = \mathbf{G}\left(\bigwedge_{i=1}^{N}\neg\left(x_i^{\text{L}} \leq x \leq x_i^{\text{U}} \wedge z_i^{\text{L}} \leq z \leq z_i^{\text{U}}\right)\right)$. The size of the regions for φ_{avoid} is 1 m^2. For $\varphi_{geofence}$, we randomly pick a subset of rooms in each house and use each chosen room's full bounds. These safety specifications are common for robot learning applications [39, 48, 100, 105]. We encode our test STL specifications using STLCG++ that can evaluate multiple state signals in parallel [51]. This ensures minimal inference overhead at runtime (10^{-5} s per timestep) , which is crucial for policy deployment. For our dynamics model,

	ϕ_{avoid}: STL / SR (% ↑)			$\phi_{geofence}$: STL / SR (% ↑)		
Decoding	**SPOC**	**Flare**	**PoliFormer**	**SPOC**	**Flare**	**PoliFormer**
Unconstrained	72.0 / 82.5	75.5 / 82.0	77.0 / 82.5	78.0 / 81.5	68.0 / 81.0	73.0 / 81.5
Filtering	100.0 / 72.0	100.0 / 78.5	100.0 / 75.5	100.0 / 72.0	100.0 / 66.5	100.0 / 67.5
HCD	100.0 / 72.5	100.0 / 81.0	100.0 / 78.5	100.0 / 76.5	100.0 / 67.5	100.0 / 72.5
RCD	93.0 / 76.0	83.0 / 82.5	87.5 / 83.5	95.5 / 80.0	80.0 / 71.5	85.5 / 77.5

Table 5.1: Comparison by decoding technique across models (SPOC, FLARE, PoliFormer) for specifications ϕ_{avoid} and $\phi_{geofence}$. Each cell reports STL satisfaction / success rate (%). Higher is better (↑).

we assume a unicycle model, an approximate first-order dynamics abstraction widely used in the robotics literature for analysis and control [21]. This representation captures the essential kinematics of motion in the plane and is widely used because it is applicable for diverse robotic platforms.

5.2.2 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 a default action (turning left or right in place) upon predicted violation of the safety specification, similar to the Simplex architecture [86]. 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. 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. The three main research questions we investigated in this paper:

1. **RQ1**: Do HCD and RCD provide higher STL satisfaction than the unconstrained baselines?
2. **RQ2**: Do HCD and RCD preserve task success rates comparable to the unconstrained baselines?
3. **RQ3**: Does RCD achieve better task success than HCD while maintaining high STL satisfaction?

5.2.3 Results

Our results are highlighted in Table 5.1. We also visualize sample trajectories in Figure 5.2 for one scene and task. Unless stated otherwise, all numbers are averaged over 200 evaluation episodes.

RQ1 – STL satisfaction. Both HCD and RCD consistently improve STL satisfaction relative to the unconstrained baselines across all models. For ϕ_{avoid}, unconstrained controllers achieve 72-77% satisfaction, while HCD raises this to 100% and RCD achieves 83–93%. For $\phi_{geofence}$, the gap is even larger: unconstrained models reach only 68–78%, whereas HCD attains perfect

(a) Unconstrained v/s HCD

(b) Unconstrained v/s RCD

Figure 5.2: Qualitative comparison of decoded trajectories for a sample scene. Each plot shows a top down view of an overlay of trajectories starting from the white dot under the instruction "find an alarm clock". The unconstrained model passes through two forbidden regions (red squares) on the way to the target object located on the table. In contrast, HCD (left) and RCD (right) modify the trajectories to respect STL safety specifications while still reaching the goal.

compliance (100 %) in all cases and RCD achieves 80–95%. We observe that the Simplex-style filtering baseline achieves similar STL-satisfaction rate as HCD, 97 % for ϕ_{avoid} and 100 % $\phi_{geofence}$. This parity is expected as both methods block any action predicted to violate the specification.

RQ2 – Task completion. Simplex-style filtering attains high satisfaction but sacrifices task success because the agent takes predefined safe actions. HCD shows similar behavior: although safety is maximized, success rates are consistently 5–10% lower than the 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. In contrast, RCD preserves success rates much closer to the unconstrained level. For ϕ_{avoid}, RCD achieves 82–85% success compared to 82–83% for the unconstrained controllers for Flare and PoliFormer. For $\phi_{geofence}$, it maintains 77–80% compared to 81–82% unconstrained for SPOC and PoliFormer. However, we note that RCD does not fully recover success in every case: on SPOC with ϕ_{avoid} and Flare with $\phi_{geofence}$, task success remains several points below the unconstrained baseline. Nevertheless, RCD enforces safety while avoiding the large performance penalty observed with filtering.

RQ3 – RCD vs. HCD. 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 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. This highlights the feasibility of combining learning-based models with formal safety constraints.

5.3 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 6

ETL: Extending STL Beyond State-Based Constraints (Proposed Work)

6.1 Motivation

Modern artificial intelligence (AI) technologies, such as foundation models (FMs), are rapidly merging as a key component in autonomous systems, being used to perform critical functions such as perception and planning. Techniques for achieving high assurance, such as verification and runtime monitoring, rely on the availability of *formal specifications* that capture the desired properties of a system. However, formally specifying the behavior of an AI-enabled system remains an open challenge [85].

Formal specifications, especially those written in a temporal logic, are expressed in terms of propositions about parts of the system state that can be observed or estimated through sensors (e.g., the velocity of a robot). For autonomous systems, behavioral properties often refer to interactions with physical objects (e.g., "the robot should avoid colliding with a table"); to observe these objects, the system relies on a perception model that operates over a high-dimensional input image. A formal specification for such a system would require propositions that relate physical objects to the input image. Here lies the fundamental obstacle to specification: devising a precise, mathematical encoding of a physical object (e.g., a table) over the high-dimensional input space (e.g., pixels) is likely to be difficult, if not impossible.

6.2 Proposed Solution

I propose a new approach for formally specifying the behavioral properties of an AI-based system. The key idea is to introduce *embeddings*—mathematical representations of real-world objects—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). For example, consider a rescue robot that is tasked with satisfying the following requirement: "Locate a potential victim while avoiding

areas with fire." Such a property would be difficult to specify using an existing specification language, due to its dependence on perception; even with access to accurate localization, the exact locations of these real-world objects are often unknown and dynamic. Instead, in an embedding-based approach, this task may be expressed as "reach the state of the world in which the system observes an object that closely resembles a potential victim, while avoiding those states where the observation resembles an area with fire." Such a specification could then be used, for example, as part of a run-time monitor or a planner to ensure that the system conforms to the desired property.

As a realization of this approach, we propose *Embedding Temporal Logic (ETL)*, a temporal logic for specifying the behaviors of AI-based systems. Compared to state-based temporal specifications (such as LTL), which are evaluated over a sequence of states, an ETL specification is evaluated over a *sequence of embeddings*, where each embedding is created from an observation that the system makes at a particular point in its execution. As counterparts to propositions in LTL, atomic constructs in ETL are *embedding predicates*, which impose a constraint over the distance between a pair of embeddings; e.g., $dist(z_o, z_t) \leq \varepsilon$ such that $dist \in \mathscr{M}_{\mathscr{Z}}$, where z_t and z_o are the target and observed embeddings in $\mathscr{Z}$, respectively, and $\varepsilon \in \mathbb{M}$ is a given *distance threshold* in metric space $\mathscr{Z}$ and $\mathscr{M}_{\mathscr{Z}}$ is the set of all metrics over $\mathscr{Z}$. A target embedding in a predicate is specified by providing images or text that correspond to a real-world concept (e.g., images or a textual description of fire). Standard temporal operators (e.g., $\mathbf{G}, \mathbf{F}$) are used to construct temporal properties out of atomic predicates.

Embeddings as a specification mechanism have the potential to **significantly broaden the range of properties that can be specified using a formal specification language**, facilitate the development of new assurance methods, and enable existing methods to be applied to AI-enabled systems. Additionally, leveraging embeddings for specification allows us to use the power of large, pretrained FMs [30,78]—which excel at encoding high-level features and concepts into embedding spaces. Moreover, integrating world models [35] that evolve within these embedding spaces further enriches our ability to reason about and verify the dynamic behaviors of such systems.

6.3 Evaluation

As highlighted in Chapter 1, first I will define the semantics of ETL over image embeddings and evaluate it for *planning* tasks in robotic systems that use an FM for scene understanding and behavioral prediction. In particular, I propose a planning method that generates actions towards the goal of satisfying a given ETL specification for diverse robotic tasks.

Then, I will focus on the stretch goal of defining ETL over textual embeddings. These specifications will allow designers to write task specifications in natural language. Then, I will work on another stretch goal of ETL defined over textual embeddings for open-ended planning tasks such as object navigation. Finally I will investigate ETL for runtime monitoring in safety critical contexts. These contributions are all potential extensions of our ETL defined on image features and considered a stretch goal beyond the primary objectives of this work.

Chapter 7

Proposed Contributions

My thesis is expected to make a number of contributions to extending existing temporal logic frameworks to ensure safety-aware deep learning based planning, including:

1. `STLINC` [49], a tool for decomposing deeply nested STL specifications for efficient trajectory planning.

2. `STLCG++` [50], a tool for efficiently computing and backpropogating through STL robustness using modern autodifferentiation frameworks such as JAX and PyTorch.

3. `SpecDec` [46] , 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`, [47] A new logic specification framework for specifying desired behavior at a higher level of abstraction such as image features.

Bibliography

[1] AI, G. Guidance: A language model control framework. https://github.com/guidance-ai/guidance, 2023. Accessed: April 27, 2025. 1.1, 5

[2] 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

[3] 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

[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), IEEE, pp. 1320–1326. 2.3

[5] ALSHIEKH, M., BLOEM, R., EHLERS, R., KÖNIGHOFER, B., NIEKUM, S., AND TOPCU, U. Safe reinforcement learning via shielding, 2017. 1.1, 2.5

[6] 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

[7] AMES, A. D., COOGAN, S., EGERSTEDT, M., NOTOMISTA, G., SREENATH, K., AND TABUADA, P. Control barrier functions: Theory and applications, 2019. 2.5

[8] 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

[9] BALAKRISHNAN, A. Signal Temporal Logic C++ Toolbox. *Available at https://github.com/anand-bala/signal-temporal-logic* (2021). 2.1

[10] 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

[11] BARTOCCI, E., DESHMUKH, J., DONZÉ, A., FAINEKOS, G., MALER, O., NICKOVIC, D., AND SANKARANARAYANAN, S. Specification-based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. *Lectures on Runtime Verification*

(2018). 1.1

[12] 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. 2.1

[13] 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, 3.2.1, **??**, **??**, 3.2

[14] 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, 5

[15] 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

[16] BOTEA, 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

[17] 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

[18] 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

[19] 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). 3.2.1

[20] 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

[21] 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. 5.2.1

[22] 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

[23] 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

[24] 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

[25] 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

[26] DOSOVITSKIY, A., BEYER, L., KOLESNIKOV, A., WEISSENBORN, D., ZHAI, X., UNTERTHINER, T., DEHGHANI, 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

[27] 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. 5, 5.2.1

[28] 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

[29] FARRELL, M., LUCKCUCK, M., AND FISHER, M. *Robotics and Integrated Formal Methods: Necessity Meets Opportunity*. Springer International Publishing, 2018, p. 161–171. 1, 5

[30] 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, 6.2

[31] 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. 3.2.1

[32] 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

[33] 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

[34] GUROBI OPTIMIZATION, INC. Gurobi optimizer reference manual, 2012. 3.2.1

[35] HA, D., AND SCHMIDHUBER, J. World models. *CoRR abs/1803.10122* (2018). 6.2

[36] 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

[37] 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

[38] 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). 4.2

[39] 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). 5.2.1

[40] HEKMATNEJAD, M. *Formalizing safety, perception, and mission requirements for testing and planning in autonomous vehicles*. PhD thesis, Arizona State University, 2021. 2.1

[41] 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

[42] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation* (1997). 4

[43] 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. 5, 5.2.1

[44] 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). 5

[45] 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

[46] 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

[47] KAPOOR, P., HAMMER, A., KAPOOR, A., LEUNG, K., AND KANG, E. Pretrained embeddings as a behavior specification mechanism, 2025. 4

[48] 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. 5.2.1

[49] 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. 1

[50] KAPOOR, P., MIZUTA, K., KANG, E., AND LEUNG, K. Stlcg++: A masking approach for differentiable signal temporal logic specification. *arXiv preprint arXiv:2501.04194* (2025).

2.1, 2

[51] KAPOOR, P., MIZUTA, K., KANG, E., AND LEUNG, K. Stlcg++: A masking approach for differentiable signal temporal logic specification. *IEEE Robotics and Automation Letters 10*, 9 (Sept. 2025), 9240–9247. 5, 5.2.1

[52] KAPOOR, P., VEMPRALA, S., AND KAPOOR, A. Logically constrained robotics transformers for enhanced perception-action planning. *arXiv preprint arXiv:2408.05336* (2024). 1.1, 5

[53] 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

[54] KIM, M. J., PERTSCH, K., KARAMCHETI, S., XIAO, T., BALAKRISHNA, A., NAIR, S., RAFAILOV, R., FOSTER, E., LAM, G., SANKETI, P., VUONG, Q., KOLLAR, T., BURCHFIEL, B., TEDRAKE, R., SADIGH, D., LEVINE, S., LIANG, P., AND FINN, C. Openvla: An open-source vision-language-action model, 2024. 5

[55] 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. 5.2.1

[56] KOYMANS, R. Specifying real-time properties with metric temporal logic. *Real-time systems 2*, 4 (1990), 255–299. 2.1

[57] 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

[58] 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, 3.2.1, 3.2.1, 3.2.1, **??**, **??**, **??**, **??**, 3.2

[59] KURTZ, V., AND LIN, H. Mixed-Integer Programming for Signal Temporal Logic with Fewer Binary Variables. *IEEE Control Systems Letters 6* (2022), 2635–2640. 2.1

[60] 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

[61] LEUNG, K., ARÉCHIGA, N., AND PAVONE, M. Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. *Int. Journal of Robotics Research* (2022). 2.1, 4

[62] 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

[63] LEUNG, K., AND PAVONE, M. Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications. In *American Control Conference* (2022). 4

[64] 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

[65] 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

[66] 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

[67] 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

[68] 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

[69] MALER, O., AND NICKOVIC, D. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT* (2004). 1, 2.1, 2.2

[70] MALER, O., AND NICKOVIC, D. Monitoring Temporal Properties of Continuous Signals. *Lecture Notes in Computer Science*, 3253 (2004), 152–166. 5

[71] MENG, Y., AND FAN, C. Diverse Controllable Diffusion Policy With Signal Temporal Logic. *IEEE Robotics and Automation Letters 9*, 10 (2024), 8354–8361. 4

[72] 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

[73] MENGHI, C., TSIGKANOS, C., PELLICCIONE, P., GHEZZI, C., AND BERGER, T. Specification patterns for robotic missions. *CoRR abs/1901.02077* (2019). 5

[74] 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, 3.2.1

[75] NAIR, S., AND FINN, C. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *ArXiv abs/1909.05829* (2019). 3

[76] 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

[77] 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

[78] RADFORD, A., KIM, J. W., HALLACY, C., RAMESH, A., GOH, G., AGARWAL, S., SASTRY, G., ASKELL, A., MISHKIN, P., CLARK, J., KRUEGER, G., AND SUTSKEVER, I. Learning transferable visual models from natural language supervision, 2021. 6.2

[79] RAMAN, V., DONZE, A., MAASOUMY, M., MURRAY, R. M., SANGIOVANNI-VINCENTELLI, A., AND SESHIA, S. Model predictive control with signal temporal logic specifications. In *Conference on Decision and Control* (2014). 2.1

[80] 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 *Proc. IEEE Conf. on Decision and Control* (2014). 1.1

[81] 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. 2.3

[82] SADIGH, D. *Safe and Interactive Autonomy: Control, Learning, and Verification*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2017. 1.1

[83] 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

[84] 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

[85] 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. 6.1

[86] SHA, L. Using simplicity to control complexity. *IEEE Software 18*, 4 (2001), 20–28. 5.2.2

[87] 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

[88] TEDRAKE, R., ET AL. Drake: Model-based design and verification for robotics. *Available at* `https://drake.mit.edu` (2019). 2.4

[89] TEDRAKE, R., AND THE DRAKE DEVELOPMENT TEAM. Drake: Model-based design and verification for robotics, 2019. 3.2.1

[90] 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, 4.1

[91] VAZQUEZ-CHANLATTE, M. pySTLToolbox. *Available at* `https://github.com/mvcisback/py-signal-temporal-logic` (2022). 2.1

[92] 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

[93] 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

[94] WILLARD, B. T., AND LOUF, R. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702* (2023). 1.1, 5

[95] 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

[96] 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

[97] 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

[98] 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

[99] 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

[100] 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. 5.2.1

[101] 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. 5, 5.2.1

[102] 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

[103] 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

[104] 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

[105] 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). 5.2.1

[106] 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

[107] ZHU, Y., MOTTAGHI, R., KOLVE, E., LIM, J. J., GUPTA, A. K., FEI-FEI, L., AND FARHADI, A. Target-driven visual navigation in indoor scenes using deep reinforcement

learning. In *Proc. IEEE Conf. on Robotics and Automation* (2016). 1.1