Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

SafeTap: Trustworthy Neurosymbolic Language to Quadrupedal Locomotion via Shield Synthesis Modulo Bitvectors

Andoni Rodríguez[1,2] , **César Sánchez**[1]
[1]IMDEA Software Institute, Spain
[2]Universidad Politécnica de Madrid, Spain

Abstract

Large language models (LLMs) are increasingly used to control embodied agents by mapping natural-language (NL) commands to high-level actions. While this paradigm enables flexible human–robot interaction, it also introduces significant safety risks, as LLM-generated (or translated) commands are not guaranteed to respect physical, environmental, or mission-critical constraints. In this paper, we present an application of *reactive synthesis modulo theories* to the real-time guardrailing of an LLM-controlled quadruped robot, using the first-order theory of bitvectors as a symbolic abstraction of the robot's action space and environment state.

Our system translates NL commands into discrete bitvector-encoded actions, which are then filtered by a formally synthesized guardrail (a.k.a., a *shield*) that enforces safety-critical properties expressed in a temporal logic. The shield operates online and corrects unsafe commands while preserving the intent of the LLM whenever possible. We instantiate our framework in a realistic locomotion setting, inspired by recent work on language-driven robot control, and demonstrate that the robot maintains safety under adversarial and dynamic environmental conditions.

1 Introduction

LLMs (Vaswani et al. 2017) have recently emerged as a powerful interface between humans and robotic systems, enabling users to specify complex tasks and actions using NL. In this paradigm, a human issues a high-level command such as "move slowly" or "turn left and avoid obstacles," which is then translated by an LLM into an actionable control representation (Yao et al. 2022). This approach has been shown to significantly increase the flexibility and usability of robotic platforms, particularly in unstructured environments (Lewis et al. 2020).

However, LLMs lack intrinsic guarantees of correctness or safety (Driess et al. 2023). They are trained to optimize linguistic plausibility rather than physical feasibility or adherence to formal constraints. As a result, LLM-generated translations (or commands) may lead to undesired behaviors, such as excessive speed on slippery terrain, collisions with obstacles, or unstable postures. This issue is particularly acute in safety-critical domains such as legged locomotion, where small deviations can result in falls, hardware damage, or harm to humans (Lee et al. 2020).

Runtime enforcement as a safety paradigm. A general way of dealing with this kind of unsafe behavior is *runtime enforcement* (a.k.a. *shielding*): a corrective mechanism that monitors the actions proposed by an untrusted component (here, an LLM) and overrides them online whenever they would violate a safety requirement. Conceptually, this paradigm has deep roots in classical *supervisory control* of discrete-event systems (Cassandras and Lafortune 2008), where a supervisor disables actions that would lead to forbidden states, and it has recently re-emerged in modern AI under different names, ranging from filter-based safety layers in reinforcement learning to recent agentic safety wrappers around LLMs (e.g., *ShieldAgent* (Chen, Kang, and Li 2025)). Within this broader family, *shield synthesis* (Bloem et al. 2015; Alshiekh et al. 2017; Corsi et al. 2025; Jansen et al. 2020; Könighofer et al. 2025; Heck et al. 2026) is the formal instantiation that we adopt: the shield is automatically constructed from a temporal-logic specification and comes with correctness guarantees against *any* sequence of proposed actions and environment behaviors.

Our approach. We use shield synthesis to guardrail LLM-controlled or LLM-assisted robots, enforcing formal safety specifications expressed in Linear Temporal Logic, LTL (Pnueli 1977). We focus on locomotion as a demanding application domain, inspired by recent work on language-conditioned robot control, *SayTap* (Tang et al. 2023), which uses a real quadruped (see Fig. 1), and related LLM-driven robot planning systems like *SayCan* (Ahn and others 2022). We deliberately build on top of SayTap among many recent LLM-based robotics frameworks because (i) it provides a structured, interpretable action space (foot-contact patterns plus discrete locomotion parameters), and (ii) this action space is directly compatible with bitvector encoding and formal synthesis. Other very recent LLM-based control approaches (e.g., end-to-end vision-language-action policies) are largely complementary to our contribution and could in principle be plugged into the same shielding pipeline whenever they expose a discrete, interpretable action interface. As in the original paper, we encode high-level actions such as gait selection, speed, turning, and posture using fixed-width bitvectors. In addition, environmental observations such as terrain type, slope, and obstacle proximity are encoded in

the same structured manner.

Since bitvectors have finite domains, realizability can be reduced to classical finite-state LTL synthesis; thus, shield construction is decidable. However, using classic Boolean LTL to express large bitvectors is neither scalable nor interpretable, so we express safety requirements using LTL **Modulo** the First-Order Theory of Bitvectors (LTL_{BV}), which we define in this paper for the first time. This allows us to state properties like:

"The robot must never gallop on slippery terrain,"
"High speed must eventually be followed by stabilization,"
"If the battery is low, jumping is disabled."

Thus, we use shield synthesis modulo theories (Rodríguez and Sánchez 2024a; Rodríguez et al. 2025a) to compute our shields offline before execution.

The overarching question we set out to answer is:

Can formal shielding guarantee safety in language-driven robot control without sacrificing flexibility or efficiency?

In our architecture, at each timestep, an LLM proposes an action encoded as a bitvector. Then, a synthesized shield monitors the current state of the robot and the environment and modifies the proposed action if and only if necessary to ensure compliance with the specification. The resulting combined behavior of LLM and shield is therefore correct-by-construction with respect to the specification, regardless of the commands issued by the LLM or the evolution of the environment.

Beyond NL to bitvector translation, our framework also supports explanation and autonomous planning. A *translation model* (LLM1) maps NL into structured bitvector actions; an *explanation model* (LLM2) explains shield interventions; and optionally, a *planner model* (LLM-P) generates high-level instructions autonomously. Crucially, none of these components can bypass the shield: formal guarantees are enforced independently of linguistic variability or hallucinations (Kaplan et al. 2020).

In summary, this paper makes the following contributions:

- We introduce a structured encoding of quadruped locomotion actions **and** environmental observations using fixed-width bitvectors, enabling temporal specifications that combine word-level arithmetic, comparisons, and bitwise constraints.
- We instantiate reactive synthesis modulo theories with the theory of bitvectors and formalize runtime shielding for LLM-generated actions using LTL_{BV} specifications. We note that, due to the bounded nature of the domain of bitvectors, shield construction is decidable.
- We design an architecture that combines synthesized shields with LLM-based translation, explanation, and optional autonomous planning, while preserving correctness regardless of NL input.
- We implement the framework in a simulated locomotion environment (see Fig. 2) and demonstrate that the synthesized shield eliminates unsafe behaviors proposed by LLM-driven control with negligible runtime overhead.

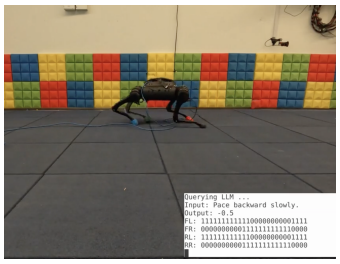

Figure 1: Robot from the *SayTap* paper, together with an input example of the human. The source code is private.

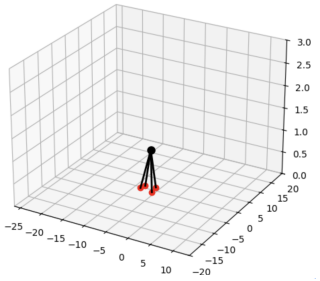

Figure 2: Simulated environment. The cube in which the robot operates is moving all the time to simulate real displacement.

By integrating learning-based control with formal methods, this work demonstrates a practical path toward deploying LLM-driven robotic systems with strong safety guarantees. More broadly, it illustrates how reactive (shield) synthesis modulo theories can serve as a principled foundation for building trustworthy, human-interpretable, and correct-by-construction robotic controllers in the age of LLMs.

2 Preliminaries

2.1 LTL, Synthesis, and $\text{LTL}_{\mathcal{T}}$

LTL (Pnueli 1977; Manna and Pnueli 1995; Pnueli and Rosner 1989) is a modal logic whose syntax consists of formulas with atoms $a \in 2^{AP}$ (where AP is a set of atomic propositions), Boolean operators like $\wedge$ and $\neg$, and temporal operators such as $\bigcirc$ and $\mathcal{U}$.

Semantics of LTL associate traces $\pi \in \Sigma^\omega$ with formulas as follows:

- $\pi \models \top$ always holds;
- $\pi \models a$ iff $a \in \pi(0)$ for $a \in AP$;
- $\pi \models \varphi_1 \vee \varphi_2$ iff $\pi \models \varphi_1$ or $\pi \models \varphi_2$;
- $\pi \models \bigcirc\varphi$ iff $\pi^1 \models \varphi$ (where π^1 is the suffix of π starting at position 1);
- $\pi \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff for some $i \geq 0$, $\pi^i \models \varphi_2$, and for all $0 \leq j < i$, $\pi^j \models \varphi_1$.

From these, common operators like $\Box$ ("always") and $\Diamond$ ("eventually") are derived. Intuitively, $\Box\varphi$ means that the property φ holds at *every* time step of the trace, while $\Diamond\varphi$ means that φ holds at *some* (current or future) time step. For example, in our locomotion setting, $\Box(\texttt{terrain} =$

$\mathsf{slippery} \rightarrow \mathsf{speed} \leq 3$) requires that the speed is bounded whenever the terrain is slippery, at every time step; whereas $\Diamond(\mathsf{gait} = \mathsf{walk})$ requires the robot to switch to the walk gait *at some point* in the future. For safety formulas (very typical in shielding) φ, which are such that for every failing trace $\pi \not\models \varphi$, there exists a finite prefix u of π where all extensions π' of u also falsify φ (i.e., $\pi' \not\models \varphi$).

Reactive synthesis for LTL (Piterman, Pnueli, and Sa'ar 2006; Finkbeiner 2016; Jacobs et al. 2017) involves constructing a system from a specification φ, where variables are partitioned into environment-controlled (uncontrollable) and system-controlled (controllable) sets. Realizability determines whether a system exists that satisfies φ against all environment behaviors, modeled as a two-player game. The system wins if it can enforce φ on the resulting trace; otherwise, the specification is unrealizable, indicating the environment has a strategy to violate φ.

LTL$_\mathcal{T}$ (Geatti, Gianola, and Gigante 2022; Rodríguez and Sánchez 2023) extends traditional LTL by replacing atomic propositions with literals from a first-order theory T (Bradley and Manna 2007) which consists of functions, constants, variables, and a domain (e.g., Presburger or real arithmetic). The semantics of LTL$_\mathcal{T}$ associate infinite traces $\pi \in \Sigma_\mathcal{T}^\omega$ with formulas, where $\Sigma_\mathcal{T}$ maps variables to values in the domain of T. A trace π satisfies a literal l at position i if the valuation $\pi(i)$ evaluates l to true under T.

LTL$_\mathcal{T}$ realizability extends LTL realizability to games in arenas with potentially infinitely many successors if variable ranges are infinite. This is why the problem is usually called *infinite-state* synthesis (Katis et al. 2018; Samuel, D'Souza, and Komondoor 2021; Samuel, D'Souza, and Komondoor 2023; Heim and Dimitrova 2025b; Heim and Dimitrova 2025a; Azzopardi et al. 2025). LTL$_\mathcal{T}$ semantics associate traces $\pi \in \Sigma_\mathcal{T}^\omega$ with formulas, where $\pi \models l$ holds if $\pi(0) \models_\mathcal{T} l$, i.e., the valuation $\pi(0)$ makes literal l true under $\mathcal{T}$.

This infinite-state/modulo-theories synthesis framework is particularly relevant to build safety shields that can enforce complex properties under complex environments, which has been demonstrated so far in use cases involving reinforcement learning agents (Kim et al. 2025; Rodríguez et al. 2025a; Corsi et al. 2024). Beyond classical realizability, more permissive notions such as *maximum realizability* for LTL modulo theories (Rodríguez and Sánchez 2026) have been recently studied as a way of recovering useful shields when the full specification is not realizable.

2.2 First-Order Theory of Bitvectors

We instantiate LTL$_\mathcal{T}$ with the first-order theory of fixed-width bitvectors, denoted BV$_k$. Bitvectors offer a finite, word-level representation of discrete actions and environment states, together with arithmetic and bitwise operations that are natural for digital control encodings.

Syntax and Semantics. For a fixed width $k \in \mathbb{N}$, a bitvector variable $x \in \text{BV}_k$ ranges over the finite domain $\{0,1\}^k$. Formulas are built from variables, constants, and the following operators and predicates:

- Bitwise Boolean operators: and, or, xor, not. These are interpreted componentwise over the k bits.
- Arithmetic operators (modulo 2^k): $+$, $-$ (and optionally $\times$), with wraparound semantics.
- Comparison predicates: equality $=$, disequality $\neq$, and unsigned comparisons $<, \leq, >, \geq$. Note that signed comparisons can be defined via the most significant bit.
- Structural operators: extraction $\mathsf{extract}_{i:j}(x)$, concatenation $\mathsf{concat}(x, y)$, and bitwise shifts.
- Derived word-level operations: tests on the most significant bit (MSB), masking via conjunction with constant words, and overflow detection (e.g., via carry or MSB change).

All arithmetic is interpreted modulo 2^k. Thus, for $k = 4$, $15 + 1 = 0$. Unsigned comparisons follow the natural order on $\{0, \ldots, 2^k - 1\}$, while signed comparisons interpret the MSB as a sign bit (two's complement). Masking operations such as x and m allow selecting specific fields within a word, which is particularly useful when encoding structured control signals.

A valuation ν assigns to each variable a word in $\{0,1\}^k$. Satisfaction of a literal ℓ under ν, written $\nu \models_{\text{BV}_k} \ell$, is defined by the standard interpretation of the above operators over fixed-width words. We refer the reader to (Barrett, Dill, and Levitt 1998; Kroening and Strichman 2016; Subramanian et al. 2016) for further understanding and classic results of bitvectors.

Decidability and Complexity. The first-order theory of fixed-width bitvectors is decidable. Since each sort BV$_k$ has a finite domain of size 2^k, any closed formula can in principle be evaluated by finite enumeration. In particular, quantifier-free BV$_k$ formulas are decidable and form a standard background theory of SMT solvers, like the ones within SMT-lib (Barrett, Fontaine, and Tinelli 2016)). In addition, Fragments with quantifier prefixes such as $\exists^*\forall^*\exists^*$ remain decidable due to the finite domain, although naive quantifier elimination may incur exponential blowup in k.

Decision procedures typically combine word-level reasoning with bit-blasting to propositional logic, yielding worst-case complexity that is exponential in the bit-width but practical for small fixed sizes.

3 Architecture and System Model

3.1 Shield as a Mealy Machine

As we mentioned, in this paper we use the shielding modulo theories framework (Rodríguez et al. 2025a). Concretely, given specification φ in LTL$_{\text{BV}}$, we perform reactive synthesis and we obtain a shield, which is modeled as a reactive transducer in the form of a Mealy machine $\mathcal{S} = (Q, q_0, \delta, \lambda)$, where:

- Q is a finite (or finitely represented) set of states,
- $q_0 \in Q$ is the initial state,
- $\delta : Q \times A \times E \rightarrow Q$ is the transition function,
- $\lambda : Q \times A \times E \rightarrow A$ is the output function.

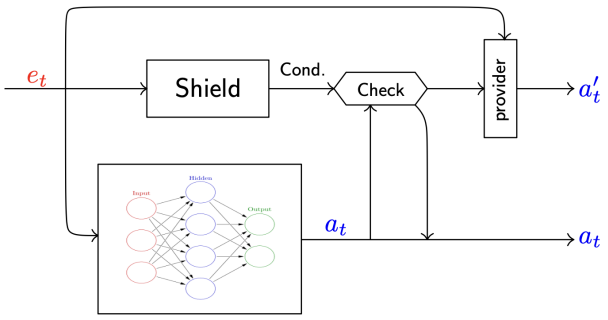

Figure 3: A general shielding architecture, where the neural network represents U (in this paper, an LLM or a human), `shield` is the Mealy machine as in Sec. 3.1, e_t is the environment input, a_t the suggested LLM output and a'_t is the correction (all of them bitvectors). `Provider` represents the Skolem function that provides the correction, which needs to receive e_t as input (Sec. 4.2).

The full architecture is a combination of two elements: the aforementioned shield S and a system under study U, which is intended to take the majority of the decisions but might be faulty with respect to φ because it is either a learned component or a policy expressed in NL.

We are now ready to explain how our architecture (see Fig. 3) works at each timestep t:

1. Environment (usually captured with sensors) provides input $e_t \in E$, which is fed to both S and U.
2. Then, U issues an action $a_t \in A$ in the domain of a first-order theory (in this case, bitvectors).
3. In case $\delta(q, e_t, a_t) = q'$ is such that q' is safe w.r.t. φ, then S passes a_t; else, it computes corrected action a'_t.
4. Robot executes either a_t or a'_t.

Although we have the same fundamental schema, in this paper, there is a critical variation between some steps:

1. First step is as aforementioned.
2. Then, U (in this paper, either a human or an LLM) **does not** issue a first-order theory action, but instead an NL command l_t.
3. Then, an LLM translates l_t into a bitvector action $a_t \in A$.
4. Last step continues as previously; and, if corrected, a'_t might feed U (see Sec. 5.3).

This architecture ensures that the robot's executed behavior always satisfies the formal specification φ, regardless of the LLM's output or environmental disturbances. Note that, at each t, the shield evolves according to a_t (resp. a'_t) as follows: $q_{t+1} = \delta(q_t, a_t, e_t)$, with the internal state q_t compactly representing the information about the past that is relevant for enforcing the specification. This includes, for example, whether unsafe conditions were recently observed or whether certain actions were repeated: e.g., "If slipping persists for several timesteps, force a stop." and "Avoid repeating unstable contact patterns." Such properties cannot be captured by memoryless correction policies and require a stateful reactive model. Importantly, the internal state of U might (and usually will) be different.

3.2 Bitvector Encoding

Following the encoding from SayTap, we represent quadruped locomotion commands using *foot contact patterns*. A foot contact pattern specifies which of the four legs are in contact with the ground at a given phase of the gait cycle. These patterns capture the essential structure of common gaits such as walking, trotting, pacing, and galloping.

Let the legs be indexed as $\{\mathsf{FL}, \mathsf{FR}, \mathsf{BL}, \mathsf{BR}\}$ (front-left, front-right, back-left, back-right). A contact pattern is a 4-bit vector $p \in \{0,1\}^4$, where each bit indicates whether the corresponding foot is in contact with the ground; e.g.

- Walk: alternating single-foot support.
- Trot: diagonal pairs in contact.
- Gallop: asymmetric patterns.
- Stand: all four feet in contact.

A full action is not a single pattern but a *symbolic description* of how patterns evolve over time, together with discrete parameters such as speed and turning direction.

Bitvector Encoding of the System. We encode each LLM-generated action as a fixed-width bitvector $a_t \in \mathsf{BV}_k$. The fields of this bitvector are interpretable and correspond to locomotion-relevant parameters.

A concrete layout used in our implementation is:

Bits	Meaning
$[0:3]$	Current foot contact pattern (4 bits)
$[4:7]$	Next foot contact pattern (4 bits)
$[8:9]$	Gait type (walk, trot, pace, gallop)
$[10:13]$	Discrete speed level (0–15)
$[14]$	Turn left/right
$[15]$	Crouch
$[16]$	Jump

This encoding allows us to express both instantaneous and short-horizon locomotion intent, such as transitions between foot contact patterns.

Bitvector Encoding of the Environment. The environment is modeled as a source of exogenous inputs, also encoded using fixed-width bitvectors. Each environment input $e_t \in \mathsf{BV}_m$ represents discretized sensor readings and contextual information.

A concrete environment encoding is:

Bits	Meaning
$[0:1]$	Terrain type (flat, rocky, slippery, soft)
$[2:6]$	Slope angle (quantized)
$[7:10]$	Obstacle proximity (0–15)
$[11:14]$	Wind intensity
$[15:18]$	Battery level
$[19]$	External disturbance flag

This discretization enables the shield to reason symbolically about safety-critical conditions such as steep slopes, slippery surfaces, high wind intensity, or low battery levels.

The environment is assumed to be adversarial but bounded: at each timestep, it may choose any valuation within the specified finite domain.

Again, it is important to note that, unlike SayTap, which focuses primarily on expressive control generation, our framework models the environment explicitly as a reactive adversary in a two-player game between the environment, choosing $e_t \in \mathrm{BV}_m$, and the system (or shield), producing $a_t \in \mathrm{BV}_k$. This modeling choice is essential for reactive synthesis. Safety guarantees must hold for *all* admissible environment behaviors, including worst-case sequences like:

- sudden transitions from flat to slippery terrain,
- abrupt increases in slope,
- rapidly decreasing obstacle distance,

Such properties cannot be captured purely at the level of instantaneous control commands. They require reasoning about sequences of environment valuations and system responses. $\mathrm{LTL}_{\mathrm{BV}}$ enables precisely this interaction between environment evolution and locomotion decisions.

4 Reactive Synthesis Modulo Bitvectors

4.1 LTL Modulo Bitvectors

Syntax and Semantics. In this paper, we have instantiated the general framework of $\mathrm{LTL}_{\mathcal{T}}$ with $\mathcal{T} = \mathrm{BV}$, obtaining $\mathrm{LTL}_{\mathrm{BV}}$, where atomic propositions are replaced by literals over fixed-width bitvectors. More formally, let $A = \mathrm{BV}_k$ denote the action space and $E = \mathrm{BV}_m$ the environment space. A trace is an infinite sequence

$$\pi = (e_0, a_0)(e_1, a_1)(e_2, a_2)\dots$$

with $e_t \in E$ and $a_t \in A$ for all $t \geq 0$. Equivalently, π induces at each time step t a valuation ν_t over all bitvector variables.

Atomic formulas are BV-literals over action and environment variables, such as:

$$\mathsf{speed}(a) \leq 3 \qquad \mathsf{terrain}(e) = \mathsf{slippery},$$

where *subfields*[1] are defined via extraction or masking.

As an illustration, consider a trace in which $\mathsf{terrain}(e_0) = \mathsf{slippery}$, $\mathsf{speed}(a_0) = 5$, and $\mathsf{speed}(a_1) = 2$. Then the formula $\square\big(\mathsf{terrain}(e) = \mathsf{slippery} \rightarrow \mathsf{speed}(a) \leq 3\big)$ is violated at time 0, since the bitvector comparison fails under ν_0.

Examples of $\mathrm{LTL}_{\mathrm{BV}}$ Properties. We now illustrate some properties expressible in LTL_{BV}. For readability, we use symbolic names for subfields, understood as bitvector expressions via extraction or masking.

- Speed restriction on slippery terrain (system):

$$\square\big(\mathsf{terrain}(e) = \mathsf{slippery} \rightarrow \mathsf{speed}(a) \leq 3\big).$$

- No gallop on steep slopes (system):

$$\square\big(\mathsf{slope}(e) > 8 \rightarrow \mathsf{gait}(a) \neq \mathsf{gallop}\big).$$

- Battery-aware limitation (system):

$$\square\big(\mathsf{battery}(e) < 2 \rightarrow \mathsf{jump}(a) = 0\big).$$

- Eventual stabilization after disturbance (system):

$$\square\big(\mathsf{disturbance}(e) = 1 \rightarrow \Diamond\, \mathsf{gait}(a) = \mathsf{walk}\big).$$

- Cooldown after high speed (system):

$$\square\big(\mathsf{speed}(a) > 10 \rightarrow \Diamond\, \mathsf{speed}(a) \leq 5\big).$$

- Bounded slope variation (environment):

$$\square\big(|\mathsf{slope}(e_{t+1}) - \mathsf{slope}(e_t)| \leq 2\big),$$

  encoded using bitvector subtraction and comparison.

- Obstacle persistence (environment):

$$\square\big(\mathsf{obstac}(e) = 1 \rightarrow \bigcirc\, \mathsf{obstac}(e) = 1 \vee \bigcirc\, \mathsf{obstac}(e) = 0\big),$$

  constraining admissible next valuations.

These examples illustrate that $\mathrm{LTL}_{\mathrm{BV}}$ naturally combines temporal reasoning with word-level arithmetic and bitwise constraints.

4.2 Synthesis Modulo Bitvectors

Decidability and Complexity. Reactive synthesis for $\mathrm{LTL}_{\mathrm{BV}}$ is decidable because the underlying theory BV_k has a finite domain k, and satisfiability of the $\forall^*\exists^*$ fragment of BV_k is decidable[2]. Informally, the intuition is that, since every variable ranges over BV_k for fixed k, each sort has cardinality 2^k. Hence, the combined action–environment alphabet $\Sigma = \mathrm{BV}_m \times \mathrm{BV}_k$ is finite; thus, every $\mathrm{LTL}_{\mathrm{BV}}$ specification induces a finite (though exponentially large) arena.

The worst-case complexity is dominated by classical LTL synthesis (2EXPTIME-complete), together with an exponential blowup in the bit-width blasting or Boolean abstraction (Rodríguez and Sánchez 2024b).

Controller Construction. In practice, in embedded control appearing in this paper, bitvector widths are small, and synthesis remains tractable. Therefore, full controller construction for $\mathrm{LTL}_{\mathrm{BV}}$ is possible via standard state-of-the-art methods like (Rodríguez, Gorostiaga, and Sánchez 2024). A key step here is the synthesis of *Skolem functions*, which realize the output λ of the Mealy machine (Sec. 3.1). Concretely, given a response of the system of the form $\forall x.\ \exists y.\ R(x, y)$, a Skolem function h witnesses correctness if $\forall x.\ R(x, h(x))$ is valid.

Example 1. *Consider an environment variable $x \in BV_k$ whose most significant bit* $\mathsf{msb}(x)$ *encodes a critical condition (e.g., a hazard flag). Now, assume the specification requires:* $\square\big(\mathsf{msb}(x) = 1 \rightarrow y = x + 1\big)$*, where, $y \in BV_k$*

[1] Here, we call *subfields* a designated group of bits inside a larger bitvector that encodes one logical component of the word (e.g., speed, gait, terrain).

[2] We refer the reader to (Rodríguez and Sánchez 2023) where authors show that, if the theory admits decidable validity for formulas of the form $\exists^*\forall^*$, then $\mathrm{LTL}_{\mathcal{T}}$ realizability and synthesis can be reduced to Boolean LTL synthesis via an equi-realizable abstraction. For bounded theories (such as bounded integers or BV_k itself), this condition holds trivially, since quantifiers range over finite domains and can be blasted into Booleans (and satisfiability of quantified Boolean formulas is well known to be decidable)

is controlled by the the system. For this particular example, the Mealy machine would be a single machine[3] *whose (potentially single) Skolem function would realize the following first-order safety–critical constraint:*

$$\forall x.\, \exists y.\, \big(\mathsf{msb}(x) = 1 \rightarrow y = x + 1\big)$$

A Skolem function witnessing realizability is:

$$h(x) = \begin{cases} y = x + 1 & \textit{if } \mathsf{msb}(x) = 1, \\ 0 & \textit{otherwise.} \end{cases}$$

Indeed, note that a simpler condition-free function $h(x) = x + 1$ *would also be admissible here.*

As we can see, controller synthesis amounts to constructing a word-level function that branches on bitvector predicates (such as the most significant bit) and produces structured output words. The resulting Mealy machine can therefore be viewed as computing Skolem functions over BV_k (and respective states), realizing the temporal specification.

Thus, one can synthesize a full bitvector-valued controller that directly maps environment words to action words, yielding a standalone Mealy machine over BV.

5 Shielded Language-Based Control

5.1 Why Not Full Control Synthesis?

Although possible, full reactive synthesis of the locomotion controller is not desirable in our setting, because the physical world in which the quadruped operates is highly complex: factors like terrain geometry, contact dynamics, disturbances, and perception uncertainty cannot be faithfully captured in a small finite abstraction. A complete controller synthesized from $\mathrm{LTL}_{\mathrm{BV}}$ would therefore be:

- either overly conservative,
- or based on a drastically simplified model,
- or intractably large.

Moreover, locomotion involves rich planning objectives (speed optimization, path efficiency, task completion) that are not naturally expressible as temporal constraints alone.

For these reasons, we deliberately restrict the outcome of the synthesis process to get a shield: a corrective Mealy machine that enforces constraints over externally proposed actions. Instead, high-level control decisions are instead delegated to a human operator in the loop (Sec. 5.2) or an LLM-based planner (Sec. 5.3).

5.2 Human-in-the-Loop Control via Language

We first consider the setting in which a human operator issues a NL command ℓ_t at every control step t and the system translates this command into a structured bitvector action. Note that commands may be well-formed locomotion instructions, such as:

- "Trot slowly and turn left."
- "Switch to walk and stop near the obstacle."
- "Crouch and move forward carefully."

However, since the interface is an unrestricted language, the operator may also provide underspecified, hallucinated or *esoteric* inputs such as:

- "Yeah, we are going to a picnic!"
- "Let's dance like a kangaroo."
- "Can you reach my goal?"

Such inputs must be mapped deterministically into a valid action in BV_k, ensuring that the downstream shield always receives a well-typed bitvector.

Translation model (LLM1). We implement a first LLM, which we call the **translation model** (LLM1; instantiated in our experiments with Claude Sonnet 4.5 (Anthropic 2024)), whose sole responsibility is translation:

$$\mathrm{LLM1} : \mathrm{NL} \longrightarrow \mathrm{BV}_k.$$

LLM1 uses:

- A precise description of the bitvector layout (fields for gait, speed, turning, posture, etc.).
- Several few-shot examples mapping NL commands to concrete bitvectors.

In this way, regardless of linguistic variability, LLM1 (together with a parser) ensures that every timestep produces a well-defined $a_t \in \mathrm{BV}_k$.

Explanation model (LLM2). We introduce a second model, the **explanation model** (LLM2; also instantiated with Claude Sonnet 4.5), whose role is producing post-hoc explanation of corrections.[4] That is, it implements the function:

$$\mathrm{LLM2} : (\ell_t, a_t, a'_t, q_t, h, S) \longrightarrow \mathrm{NL\ Explanation},$$

where h is the prefix of the play (including the environment input at t itself), S is the shield, and q_t is the current internal state of S.

Then, LLM2 is instructed to:

- Explain the correction in human-understandable terms.
- Reference safety constraints explicitly.
- Preserve the user's intent whenever possible.

For example:

Original command: "Gallop forward quickly."
Correction: Switched to trot at moderate speed, with value...
Explanation: The terrain is slippery and galloping at high speed could cause instability. The system reduced speed and switched to a more stable gait to ensure safety, via value...

[3] This is not the case in the quadruped's shield that results from specifications that have a realistic temporal nature.

[4] Prior work has investigated explanations and representations for reactive systems and shields in a different sense (Bassan et al. 2023; Balachander, Filiot, and Raskin 2023; Rodríguez et al. 2025b; Brizzio et al. 2026a; Brizzio et al. 2026b). Here we refer instead to explaining *online corrections* produced by an already-realized shield.

5.3 Autonomous LLM-Based Planning

Planner model (LLM-P). We now replace the human operator with an autonomous NL-based planner, the **planner model** (LLM-P; instantiated with Claude Sonnet 4.5 in our experiments), which produces output $\ell_t = \text{LLM-P}(h)$, where h is again the prefix, but LLM-P also considers its context, including (but not limited to) the optional mission-level goals, desired previous executions etc. The purpose of LLM-P is not to generate bitvectors, but to produce semantically meaningful instructions such as:

- "Climb the slope cautiously."
- "Increase speed until the obstacle is within two meters."
- "Switch to a stable gait due to rough terrain."

Note that, the output of LLM-P is purely linguistic, and therefore subject to the same translation and correction mechanisms as in the human-in-the-loop case of Sec. 5.2.

Closed-Loop Explanation Feedback. A key architectural difference w.r.t Sec. 5.2 is that the output of **LLM2** is now fed back into LLM-P (imagine an arrow from `Provider` to the neural network in Fig. 3). Thus, the control loop becomes:

$$\text{LLM-P} \rightarrow \text{LLM1} \rightarrow \text{Shield} \rightarrow \text{LLM2} \rightarrow \text{LLM-P}.$$

This creates a feedback mechanism in which the planner gradually adapts to the constraints enforced by the shield, which is expected to have two practical effects:

- It reduces repeated violations, as LLM-P learns that certain patterns systematically trigger corrections, without needing a human-in-the-loop for learning like in *RLHF* (Bai et al. 2022)[5].
- It improves transparency, interpretability and human alignment, since both planner decisions and safety overrides are expressed in NL.

6 Experimental Evaluation

We evaluate our framework on a simulated quadruped locomotion platform with the goal of answering the following research questions:

- **RQ1:** Does shielding eliminate safety violations produced by NL-driven control?
- **RQ2:** What is the runtime overhead of the shield?
- **RQ3:** How does the system behave under non-standard NL inputs like hallucinations or under-specifications?
- **RQ4:** Does explanation feedback reduce repeated violations in the LLM-based planner setting?

[5]Moreover, this might also have an impact on shield overhead, because we can now embed learning into the testing phase. However, this direction is out of the scope of this paper.

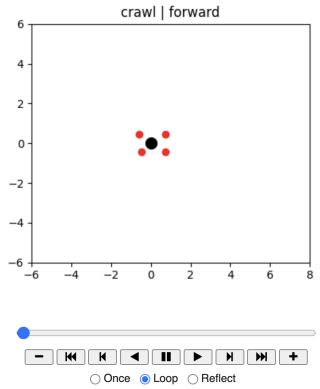

Figure 4: A 2D view of our simulated environment. Note also below how we offer several options like speed of simulation or to go forward or backwards.

6.1 Implementation Details

Simulation Environment. We implemented our framework in a quadruped simulation environment inspired by SayTap-style locomotion. The environment provides discretized observations encoded into the bitvector structure described in Sec. 3.2. Each run consists of 1,000 control steps (the control loop runs at 50 Hz, so each scenario corresponds to 20 s of simulated time), with randomized perturbations whose statistics are summarized below:

- **Terrain transitions.** The terrain bitvector field switches at every step with probability $p_{\text{terr}} = 0.05$ (i.e., on average $\sim$50 transitions per scenario), uniformly to one of $\{$flat, rocky, slippery, soft$\}$.
- **Obstacles.** An obstacle within sensor range is sampled at each step with probability $p_{\text{obs}} = 0.10$; obstacle proximity then evolves with a bounded $|\Delta| \leq 2$ random walk so successive readings remain physically plausible.
- **Slope.** The slope is a bounded random walk over $\{0, \ldots, 15\}$ with $|\Delta| \leq 2$ per step; the walk reflects at the boundary.
- **External disturbances.** The disturbance flag is raised at each step with probability $p_{\text{dist}} = 0.02$ and stays raised for a uniformly random duration in $[1, 5]$ steps.

The quadruped's code is fully implemented in Python, and only imports `numpy` and visualization libraries, thanks to which we can view our robot in 3D (like in Fig. 2) or in 2D (see Fig. 4).

Shield Implementation. The shield is synthesized offline, given a LTL_{BV} specification and a partition between players. Our tool is called `ShieldBit` and it implements a counter-example guided reactive synthesis modulo theories approach (Rodríguez, Gorostiaga, and Sánchez 2025) in Python, which uses `Z3` (de Moura and Bjørner 2008) and `Strix` (Meyer, Sickert, and Luttenberger 2018) as SMT-solving and reactive synthesis backends respectively. The resulting Mealy machine is stored in the standard `HOA` format (Babiak et al. 2015) that is intended to act as a controller,

	Human-in-the-Loop		LLM-P (No Feedback)		LLM-P (+Feedback)	
Platform	LLM-only	Shielded	LLM-only	Shielded	LLM-only	Shielded
Q1 (Indoor)	84	0	97	0	63	0
Q2 (Research)	102	0	118	0	74	0
Q3 (Heavy-Payload)	136	0	151	0	98	0
Q4 (Rough-Terrain)	173	0	194	0	121	0
Total	**495**	**0**	**560**	**0**	**356**	**0**

Table 1: Total number of safety violations over 120 scenarios per platform (1,000 steps per scenario).

Platform	LLM1 Inference	Shield Execution	Total (Shielded)	Physics Simulation	Control Budget
Q1 (Indoor)	3.2	0.08	3.28	6.4	20
Q2 (Research)	3.5	0.09	3.59	6.7	20
Q3 (Heavy-Payload)	3.7	0.10	3.80	7.1	20
Q4 (Rough-Terrain)	3.9	0.11	4.01	7.6	20
Average	**3.58**	**0.095**	**3.68**	**6.95**	**20**

Table 2: Average runtime per control step (milliseconds). The control budget corresponds to a 50 Hz control frequency (20 ms per cycle).

but instead is used as a shield by realizing its outputs only when a correction is necessary. At runtime, the shield execution works as expected:

1. reading incoming signals e_t and a_t,
2. performing a transition-table lookup (because of `HOA`),
3. possibly replacing a_t with a'_t via a Skolem function call.

Platforms. Within the playground, we considered four simulated quadruped (*quad.*) variants:

ID	Description	Use Case
Q1	Lightweight indoor quad.	HRI/navigation
Q2	Standard research quad.	General-purpose
Q3	Heavy-payload quad.	Transport tasks
Q4	Rough-terrain quad.	Outdoor exploration

Each platform differs in stability envelope, admissible gait transitions, and maximum speed.

Baselines. We compare two control architectures:

1. **LLM-only:** LLM-generated bitvector actions are directly the actuator's outputs.
2. **Shielded (ours):** The synthesized Mealy machine shield enforcing full temporal LTL_{BV} specifications.

For the autonomous setting, we additionally compare:

- LLM-P without explanation feedback.
- LLM-P with LLM2's feedback loop enabled.

6.2 LLM Construction and Prompting

As explained in Sec. 5, we use three language models in our architecture: a *translation model* (LLM1), an *explanation model* (LLM2), and a *planner model* (LLM-P). For all three roles we use **Claude Sonnet 4.5** (Anthropic 2024) as the underlying instruction-tuned LLM, with role-specific prompts.

Translation model (LLM1). LLM1 is prompted with:

- the bitvector layout specification,
- allowed symbolic values,
- few-shot NL-to-bitvector examples.

A deterministic post-processing layer (again, in Python) ensures field validity and compiles the structured output into $a_t \in \text{BV}_k$.

Explanation model (LLM2). LLM2 receives the tuple $(\ell_t, a_t, a'_t, q_t, h_t)$ and produces a NL explanation of the correction. It is prompted with:

- the triggered safety rule,
- relevant environment features,
- symbolic differences between a_t and a'_t,
- few-shot human-crafted explanation examples.

Planner model (LLM-P). LLM-P is the same base model adapted via Chain-of-Thought (CoT) (Wei et al. 2022) prompting on SayTap-style traces and videos. This prompting strategy encourages structured reasoning from environment summaries to high-level locomotion instructions, without encoding formal safety rules directly.

Importantly, none of the LLMs have direct access to the formal LTL_{BV} specification; safety is enforced exclusively by the shield.

6.3 Results by Research Question

We now discuss our results.

RQ1: Safety Violations. Across all platforms and 120 randomized scenarios per platform, Tab. 1 summarizes the total number of safety violations.

As expected, the LLM-only architecture produces frequent violations, especially under slippery terrain, steep slopes, and disturbance events. After observing them (plus

Platform	Invalid Bitvectors	Unsafe (LLM-only)	Unsafe (Shielded)	Avg. Intervention Rate (%)	Avg. Speed (LLM-only)	Avg. Speed (Shielded)	Instability Events (LLM-only)
Q1 (Indoor)	0	21	0	6.1	9.8	6.4	17
Q2 (Research)	0	27	0	6.8	10.3	6.9	22
Q3 (Heavy-Payload)	0	34	0	7.4	11.1	7.2	29
Q4 (Rough-Terrain)	0	41	0	7.9	12.6	7.8	36
Total / Avg.	**0**	**123**	**0**	**7.05**	**10.95**	**7.08**	**26**

Table 3: Performance under esoteric or underspecified natural-language inputs (40 scenarios per platform, 1,000 steps each).

Platform	No Feedback	+Feedback	Absolute Reduction	Reduction (%)	Avg. Corrections / Episode	Avg. Episode Length
Q1 (Indoor)	29	11	18	62%	4.1	10000
Q2 (Research)	37	15	22	59%	4.8	10000
Q3 (Heavy-Payload)	44	19	25	57%	5.3	10000
Q4 (Rough-Terrain)	58	24	34	59%	6.7	10000
Average	**42**	**17**	**25**	**59%**	**5.2**	**10000**

Table 4: Repeated unsafe proposals in autonomous LLM-P setting.

using explanations of LLM-2), we realized that these violations are primarily due to excessive speed, inappropriate gait selection, or lack of recovery after instability. In contrast, the shield eliminates *all* observed safety violations across all configurations. This confirms that the LTL_{BV} specification is effectively enforced at runtime.

An interesting and somewhat unexpected observation is that violations increase noticeably when switching from human-in-the-loop control to autonomous LLM-P planning without feedback. Although LLM-P is fine-tuned on SayTap-style traces, its goal-directed behavior occasionally prioritizes task completion over stability, leading to more aggressive commands (similar to reinforcement learning without safety constraints (Achiam et al. 2017)). This highlights the importance of formal enforcement, even when using planning models that are supposed to incorporate safety constraints a priori. However, although this has not been reported in numbers, it is important to note that the time the LLM-planner takes is much less than what a human-in-the loop needs.

Furthermore, enabling explanation feedback seems to reduce the number of unsafe proposals at the source (before shielding), indicating that linguistic adaptation can partially internalize safety constraints, even though correctness ultimately does not rely on it. This suggests that *better* methods for explanation feedback (e.g., showing the complete CoT of LLM-2 to LLM-P) might have an impact in the amount of shielded actions, which is always desired either for minimizing policy disruption or reducing runtime overhead. Indeed, we now dive deeper in the latter.

RQ2: Runtime Overhead. Tab. 2 reports average per-step runtime, where, as expected, shield execution time is negligible compared to LLM inference time (note that LLM inference time dominates the control loop budget by two orders of magnitude relative to shield execution). The shield performs only a transition lookup in the `HOA` and a small number of bitvector operations, resulting in sub-millisecond overhead. We acknowledge that this process might be slower in longer instances, but that would already convey a scalability discussion first.

More interestingly, the runtime remains stable across all four quadruped variants, despite differences in admissible gait transitions and stability envelopes. This confirms that shield complexity depends primarily on the specification size, rather than on the physical model.

RQ3: Esoteric Inputs. Tab. 3 summarizes behavior under intentional esoteric commands like "Show us how nervous you are".

As expected, LLM1 consistently produces valid bitvectors, even when the input command is unrelated to locomotion. This confirms that structured prompting and deterministic post-processing successfully constrain the output space.

However, the LLM-only execution occasionally maps esoteric commands to dynamically unstable behaviors. For example, vague or playful instructions sometimes result in high-speed or asymmetric gaits. This behavior was not explicitly trained but emerges unexpectedly often, due to general-purpose LLM priors.

Also, we want to note that intervention rates under esoteric inputs are only marginally higher than under standard commands. This suggests that LLM1 tends to default to relatively conservative locomotion modes when semantic intent is unclear, rather than producing extreme actions.

Most importantly, shielded execution maintains full safety in all cases, demonstrating robustness to linguistic variability and potential hallucinations.

RQ4: Explanation Feedback. This question was partially responded in Tab. 1. Instead, Tab. 4 focuses on showing repeated violation counts with and without LLM2 feedback in the autonomous setting over longer episodes ($x10$) and the usage of random seeds for each execution.

As expected, feeding explanations back into LLM-P reduces repeated unsafe proposals. The reduction averages approximately 59% across platforms, indicating that linguistic feedback encourages adaptation.

Interestingly, the reduction is consistent across all quadruped variants, including rough-terrain Q4. This sug-

gests that the planner adapts to abstract safety concepts (e.g., "slippery implies slower") rather than memorizing platform-specific behaviors. Note that feedback does not eliminate unsafe proposals entirely, which indicates that NL explanations (although they might be potentially improved, like previously commented), while helpful, do not fully substitute for formal constraints. LLM-P remains a probabilistic model and may occasionally explore boundary behaviors.

Crucially, safety is unaffected by this limitation, since the shield enforces correctness independently. The feedback loop therefore improves efficiency and interpretability, without weakening formal guarantees.

In summary, Tables 1–4 demonstrate that reactive synthesis modulo bitvectors provides strong safety guarantees with negligible runtime overhead, while preserving the flexibility and expressiveness of language-based control. The combination of formal shielding and language-based planning yields a system that is both safe and adaptable.

7 Conclusion

We presented a framework for enforcing safety-critical properties on language-driven robot control using shield synthesis modulo the theory of bitvectors. By interposing a synthesized shield between an LLM and a quadruped controller, we showed how high-level symbolic commands can be corrected online to satisfy temporal specifications, even under adversarial environmental conditions or determination to achieve complex goals. Our approach operates directly at the word level, avoiding bit-blasting, and yields a correct-by-construction controller with negligible runtime overhead while feeding the system back with NL explanations whenever corrections happen.

Beyond demonstrating the practical feasibility of shield synthesis modulo theories for trustworthy quadruped deployment, this work illustrates how formal methods can be used to guardrail learning-based systems without restricting their expressiveness, especially in systems driven by LLMs.

We frame this work as a *first step* towards deployment: all reported experiments are done in a simulation, and bridging to physical hardware will require addressing standard sim-to-real considerations (discretization mismatch, sensor noise, contact dynamics) – this is ongoing work, on a quadruped platform other than SayTap's, since the latter is not open source. The framework is also not limited to quadrupeds: adapting it to bipeds (e.g., on humanoid benchmarks such as HumanoidBench (Sferrazza et al. 2024)) requires only redefining the contact-pattern, gait, and stability encodings, while the synthesis pipeline remains unchanged. Our shield is moreover complementary to symbolic planning frameworks such as HTNs (Erol, Hendler, and Nau 1994; Nau et al. 2003): just as it wraps LLM-P, it can wrap any HTN whose leaf actions are encodable as bitvectors. On the formal side, we plan to extend the framework to richer logics, such as richer background theories combining bitvectors with reals or arrays, navigating the well-known trade-off between expressiveness and synthesis complexity. Finally, integrating perception-level uncertainty with formal guarantees (Ladner, Shoukry, and Althoff 2026) is a next step.

AI Declaration

AI was used for polishing of text and preparation of code. All AI-generated content was reviewed, verified, and validated by the authors, who take full responsibility for the final version of this work.

Acknowledgments

This work was funded in part by the DECO Project (PID2022-138072OB-I00) funded by MCIN/AEI/10.13039/501100011033 and by the ESF+, FPI PREP2022-000334 and Grant CEX2024-001471-M funded by MICIU/AEI/10.13039/501100011033.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization.

Ahn, M., et al. 2022. Do as I can, not as I say: Grounding language in robotic affordances. In *Proc. of the 6th Conference on Robot Learning (CoRL 2022)*.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2017. Safe reinforcement learning via shielding. *arXiv* abs/1708.08611.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Azzopardi, S.; Stefano, L. D.; Piterman, N.; and Schneider, G. 2025. Full ltl synthesis over infinite-state arenas. In *Proc. of the 37th International Conference on Computer Aided Verification (CAV'25)*, LNCS.

Babiak, T.; Blahoudek, F.; Duret-Lutz, A.; Klein, J.; Křetínský, J.; Müller, D.; Parker, D.; and Strejček, J. 2015. The hanoi omega-automata format. In *Computer Aided Verification*, 479–486. Cham: Springer International Publishing.

Bai, Y.; Jones, A.; Ndousse, K.; Askell, A.; Chen, A.; DasSarma, N.; Drain, D.; Fort, S.; Ganguli, D.; Henighan, T.; Joseph, N.; Kadavath, S.; Kernion, J.; Conerly, T.; El-Showk, S.; Elhage, N.; Hatfield-Dodds, Z.; Hernandez, D.; Hume, T.; Johnston, S.; Kravec, S.; Lovitt, L.; Nanda, N.; Olsson, C.; Amodei, D.; Brown, T.; Clark, J.; McCandlish, S.; Olah, C.; Mann, B.; and Kaplan, J. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback.

Balachander, M.; Filiot, E.; and Raskin, J.-F. 2023. LTL reactive synthesis with a few hints. In *Proc. of the 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2023)*, LNCS, 309–328. Springer.

Barrett, C.; Dill, D.; and Levitt, J. 1998. A decision procedure for bit-vector arithmetic. In *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 522–527.

Barrett, C.; Fontaine, P.; and Tinelli, C. 2016. The satisfiability modulo theories library (smt-lib.

Bassan, S.; Amir, G.; Corsi, D.; Refaeli, I.; and Katz, G. 2023. Formally explaining neural networks within reactive systems. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD 2023)*, 1–13. IEEE.

Bloem, R.; Könighofer, B.; Könighofer, R.; and Wang, C. 2015. Shield synthesis: Runtime enforcement for reactive systems. In *Proc. of the 21st International Conference in Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, volume 9035 of *LNCS*, 533–548. Springer.

Bradley, A. R., and Manna, Z. 2007. *The Calculus of Computation*. Springer-Verlag.

Brizzio, M.; Rodríguez, A.; Sánchez, C.; and Degiovanni, R. 2026a. AIGLE: A tool for compact, legible AIGER circuits from safety specifications. In *Proc. of the 23rd International Conference on Principles of Knowledge Representation and Reasoning (KR 2026)*.

Brizzio, M.; Rodríguez, A.; Sánchez, C.; and Degiovanni, R. 2026b. Mode-based requirements decomposition for scalable reactive synthesis. In *Proc. of the 34th IEEE International Requirements Engineering Conference (RE 2026)*.

Cassandras, C. G., and Lafortune, S. 2008. *Introduction to Discrete Event Systems*. Springer, 2nd edition.

Chen, Z.; Kang, M.; and Li, B. 2025. ShieldAgent: Shielding agents via verifiable safety policy reasoning.

Corsi, D.; Amir, G.; Rodríguez, A.; Katz, G.; Sánchez, C.; and Fox, R. 2024. Verification-guided shielding for deep reinforcement learning. *RLJ* 4:1759–1780.

Corsi, D.; Mallik, K.; Rodríguez, A.; and Sánchez, C. 2025. Efficient dynamic shielding for parametric safety specifications. In *Proc. of the 23rd International Symposium on Automated Technology for Verification and Analysis (ATVA 2025)*, LNCS.

de Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4693 of *LNCS*, 337–340. Springer.

Driess, D.; Xia, F.; Sajjadi, M. S. M.; Lynch, C.; Chowdhery, A.; Ichter, B.; Wahid, A.; Tompson, J.; Vuong, Q.; Yu, T.; Huang, W.; Chebotar, Y.; Sermanet, P.; Duckworth, D.; Levine, S.; Vanhoucke, V.; Hausman, K.; Toussaint, M.; Greff, K.; Zeng, A.; Mordatch, I.; and Florence, P. 2023. PaLM-E: An embodied multimodal language model.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Finkbeiner, B. 2016. Synthesis of reactive systems. In *Dependable Software Systems Engineering*, volume 45 of *NATO Science for Peace and Security Series - D: Information and Communication Security*. IOS Press. 72–98.

Geatti, L.; Gianola, A.; and Gigante, N. 2022. Linear temporal logic modulo theories over finite traces. In *Proc. of the 31st Int'l Joint Conf. on Artificial Intelligence, (IJCAI'22)*, 2641–2647. ijcai.org.

Heck, L.; Macák, F.; Andriushchenko, R.; Češka, M.; and Junges, S. 2026. Shields to guarantee probabilistic safety in MDPs. In *Proc. of the 38th International Conference on Computer Aided Verification (CAV 2026)*, LNCS.

Heim, P., and Dimitrova, R. 2025a. Issy: A comprehensive tool for specification and synthesis of infinite-state reactive systems. In *Proc. of the 37th International Conference on Computer Aided Verification (CAV'25)*, LNCS.

Heim, P., and Dimitrova, R. 2025b. Translation of temporal logic for efficient infinite-state reactive synthesis. *Proc. ACM Program. Lang.* 9(POPL).

Jacobs, S.; Basset, N.; Bloem, R.; Brenguier, R.; Colange, M.; Faymonville, P.; Finkbeiner, B.; Khalimov, A.; Klein, F.; Michaud, T.; Pérez, G. A.; Raskin, J.; Sankur, O.; and Tentrup, L. 2017. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. In *Proc. of the 6th Workshop on Synthesis (SYNT@CAV 2017)*, volume 260 of *EPTCS*, 116–143.

Jansen, N.; Könighofer, B.; Junges, S.; Serban, A.; and Bloem, R. 2020. Safe reinforcement learning using probabilistic shields (invited paper). In *Proc. of the 31st International Conference on Concurrency Theory (CONCUR 2020)*, LIPIcs, 3:1–3:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T. B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; and Amodei, D. 2020. Scaling laws for neural language models.

Katis, A.; Fedyukovich, G.; Guo, H.; Gacek, A.; Backes, J.; Gurfinkel, A.; and Whalen, M. W. 2018. Validity-guided synthesis of reactive systems from assume-guarantee contracts. In *Proc. of the 24th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'18), Part II*, volume 10806 of *LNCS*, 176–193. Springer.

Kim, K.; Corsi, D.; Rodríguez, A.; Lanier, J.; Parellada, B.; Baldi, P.; Sánchez, C.; and Fox, R. 2025. Realizable continuous-space shields for safe reinforcement learning. In *Proc. of the 7th Annual Learning for Dynamics & Control Conference (L4DC'25)*, PMLR.

Könighofer, B.; Bloem, R.; Jansen, N.; Junges, S.; and Pranger, S. 2025. Shields for safe reinforcement learning. *Commun. ACM* 68(11):80–90.

Kroening, D., and Strichman, O. 2016. *Decision Procedures*.

Ladner, T.; Shoukry, Y.; and Althoff, M. 2026. Perception with guarantees: Certified pose estimation via reachability analysis. In *Proc. of the 38th International Conference on Computer Aided Verification (CAV 2026)*, LNCS.

Lee, J.; Hwangbo, J.; Wellhausen, L.; Koltun, V.; and Hutter, M. 2020. Learning quadrupedal locomotion over challenging terrain. *Science Robotics* 5(47):eabc5986.

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; tau Yih, W.; Rocktäschel, T.; Riedel, S.; and Kiela, D. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks.

Manna, Z., and Pnueli, A. 1995. *Temporal verification of reactive systems - safety*. Springer.

Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit reactive synthesis strikes back! In *Proc. of the 30th Int'l Conf. on Computer Aided Verification (CAV'18) Part I*, volume 10981 of *LNCS*, 578–586. Springer.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2006. Synthesis of reactive(1) designs. In *Proc. of VMCAI'06*, 364–380. Springer.

Pnueli, A., and Rosner, R. 1989. On the synthesis of an asynchronous reactive module. In *Proc. of the 16th Int'l Colloqium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *LNCS*, 652–671. Springer.

Pnueli, A. 1977. The temporal logic of programs. In *Proc. of the 18th IEEE Symp. on Foundations of Computer Science (FOCS'77)*, 46–67. IEEE CS Press.

Rodríguez, A., and Sánchez, C. 2023. Boolean abstractions for realizabilty modulo theories. In *Proc. of the 35th Int'l Conf. on Computer Aided Verification (CAV'23)*, volume 13966 of *LNCS*. Springer, Cham.

Rodríguez, A., and Sánchez, C. 2024a. Adaptive Reactive Synthesis for LTL and LTLf Modulo Theories. In *Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024)*, 10679–10686. AAAI Press.

Rodríguez, A., and Sánchez, C. 2024b. Realizability modulo theories. *Journal of Logical and Algebraic Methods in Programming* 140:100971.

Rodríguez, A., and Sánchez, C. 2026. Maximum realizability for LTL modulo theories. In *Proc. of the 27th International Symposium on Formal Methods (FM 2026)*, LNCS.

Rodríguez, A.; Gorostiaga, F.; and Sánchez, C. 2024. Predictable and performant reactive synthesis modulo theories via functional synthesis. In *Proc. of 22nd the Int'l Symposium on Automated Technology for Verification and Analysis (ATVA'24)*. Springer.

Rodríguez, A.; Gorostiaga, F.; and Sánchez, C. 2025. Counter Example Guided Reactive Synthesis for LTL Modulo Theories. In *Proc. of the 37th International Conference on Computer Aided Verification (CAV'25)*, LNCS.

Rodríguez, A.; Amir, G.; Corsi, D.; Sánchez, C.; and Katz, G. 2025a. Shield synthesis for LTL modulo theories. In *Proc. of the 39th AAAI Conf. on Artificial Intelligence (AAAI'25)*, 15134–15142. AAAI Press.

Rodríguez, A.; Shaik, I.; Corsi, D.; Fox, R.; and Sanchez, C. 2025b. Explanations for unrealizability of infinite-state safety shields. In *Proc. of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR'25)*.

Samuel, S.; D'Souza, D.; and Komondoor, R. 2021. Gensys: a scalable fixed-point engine for maximal controller synthesis over infinite state spaces. In *Proc. of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21)*, 1585–1589. ACM.

Samuel, S.; D'Souza, D.; and Komondoor, R. 2023. Symbolic fixpoint algorithms for logical LTL games. In *Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2023)*, 698–709. IEEE.

Sferrazza, C.; Huang, D.-M.; Lin, X.; Lee, Y.; and Abbeel, P. 2024. HumanoidBench: Simulated humanoid benchmark for whole-body locomotion and manipulation.

Subramanian, S.; Berzish, M.; Zheng, Y.; Tripp, O.; and Ganesh, V. 2016. A solver for a theory of strings and bitvectors.

Tang, Y.; Yu, W.; Tan, J.; Zen, H.; Faust, A.; and Harada, T. 2023. Saytap: Language to quadrupedal locomotion.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2022. Chain-of-thought prompting elicits reasoning in large language models.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2022. ReAct: Synergizing reasoning and acting in language models.