

# Temporal Logic Imitation: Learning Plan-Satisficing Motion Policies from Demonstrations

Yanwei Wang  
MIT

Nadia Figueroa  
University of Pennsylvania

Shen Li  
MIT

Ankit Shah  
Brown University

Julie Shah  
MIT

**Abstract:** Learning from demonstration (LfD) has succeeded in tasks featuring a long time horizon. However, when the problem complexity also includes human-in-the-loop perturbations, state-of-the-art approaches do not guarantee the successful reproduction of a task. In this work, we identify the roots of this challenge as the failure of a learned continuous policy to satisfy the discrete plan implicit in the demonstration. By utilizing modes (rather than subgoals) as the discrete abstraction and motion policies with both mode invariance and goal reachability properties, we prove our learned continuous policy can simulate any discrete plan specified by a linear temporal logic (LTL) formula. Consequently, an imitator is robust to both task- and motion-level perturbations and guaranteed to achieve task success. **Project page:** <https://yanweiw.github.io/tli/>

**Keywords:** Certifiable Imitation Learning, Dynamical Systems, Formal Methods

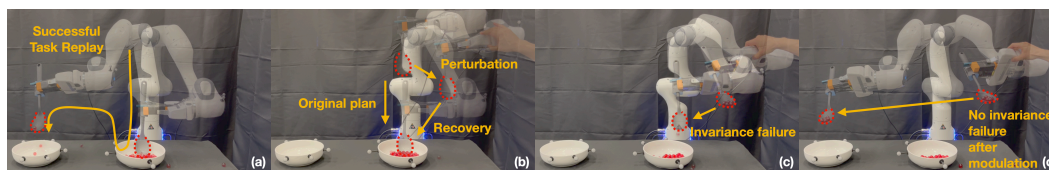


Figure 1: (a) A successful replay of the scooping task. The robot (b) is robust to motion-level perturbations; (c) experiences an invariance failure (i.e., drops material) after a task-level perturbation; and (d) re-scoops after a task-level perturbation, avoiding failure after DS motion policy modulation.

## 1 Introduction

In prior work, learning from demonstration (LfD) [1, 2] has successfully enabled robots to accomplish multi-step tasks by segmenting demonstrations (primarily of robot end-effector or tool trajectories) into sub-tasks/goals [3, 4, 5, 6, 7, 8], phases [9, 10], keyframes [11, 12], or skills/primitives/options [13, 14, 15, 16]. Most of these abstractions assume reaching subgoals sequentially will deliver the desired outcomes; however, successful imitation of many manipulation tasks with spatial/temporal constraints cannot be reduced to imitation at the motion level unless the learned motion policy also satisfies these constraints. This becomes highly relevant if we want robots to not only imitate but also generalize, adapt and be robust to perturbations imposed by humans, who are in the loop of task learning and execution. LfD techniques that learn stable motion policies with convergence guarantees (e.g., Dynamic Movement Primitives (DMP) [17], Dynamical Systems (DS) [18]) are capable of providing such desired properties but only at the motion level. As shown in Fig. 1 (a-b) a robot can successfully replay a soup-scooping task while being robust to physical perturbations with a learned DS. Nevertheless, if the spoon orientation is perturbed to a state where all material is dropped, as seen in Fig. 1 (c), the motion policy will still lead the robot to the target, unaware of the task-level failure or how to recover from it. To alleviate this, we introduce an imitation learning approach that is capable of i) reacting to such task-level failures with Linear Temporal Logic (LTL) specifications, and ii) modulating the learned DS motion policies to avoid repeating those failures as shown in Fig. 1 (d).

**Example** We demonstrate that successfully reaching a goal via pure motion-level imitation does not imply successful task execution. The illustrations in Fig. 2 represent a 2D simplification of the soup-scooping task, where task success requires a continuous trajectory to simulate a discrete plan of consecutive transitions through the colored regions. Human demonstrations, shown in Fig. 2 (a), are employed to learn a DS policy [19], depicted by the streamlines in Fig. 2 (b). The policy is

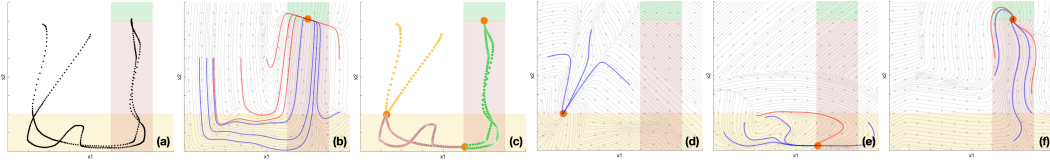


Figure 2: Mode abstraction of a 2D soup-scooping task:  $x_1$  and  $x_2$  denote the spoon’s orientation and distance to the soup. **(a)** Task: To move the spoon’s configuration from the white region (spoon without soup)  $\Rightarrow$  yellow region (spoon in contact with soup)  $\Rightarrow$  pink region (spoon holding soup)  $\Rightarrow$  green region (soup at target). (Note that transitions (white  $\Rightarrow$  pink) and (white  $\Rightarrow$  green) are not physically realizable.) Black curves denote successful demonstrations. **(b)** Learning DS policies [19] over unsegmented data can result in successful task replay (blue trajectories), but lacks a guarantee due to invalid transitions (red trajectories). **(c)** Trajectories are segmented into three colored regions (modes) with orange attractors. **(d-f)** Learning DSs on segments may still result in *invariance failures* (i.e., traveling outside of modes as depicted by red trajectories).

stress-tested by applying external perturbations, displacing the starting states of the policy rollouts. As shown, only blue trajectories succeed in the task, while the red ones fail due to discrete transitions that are not physically realizable (e.g., white  $\Rightarrow$  pink). As shown in Fig. 2 **(c-f)**, even if the demonstrations are further segmented by subgoals (and corresponding DS policies are learned), this issue is not mitigated. While one could treat this problem as covariate shift and solve it by asking a human for more demonstrations [20], in this work, we frame it as the mismatch between a learned continuous policy and a discrete task plan specified by the human in terms of a logical formula. Specifically, the core challenges illustrated by this example are two-fold: 1) subgoals only impose point constraints that are insufficient to represent the boundary of a discrete abstraction; and 2) the continuous policy can deviate from a demonstrated discrete plan when perturbed to unseen parts of the state space, and is incapable of replanning to ensure all discrete transitions are valid.

To address these challenges, our proposed approach employs “modes” as discrete abstractions. We define a *mode* as a set of robot and environment configurations that share the same sensor reading [21, 22]; e.g., in Fig. 2, each colored region is a unique mode, and every mode has a boundary that imposes path constraints on motion policies. Additionally, we use a task automaton as a receding-horizon controller that replans when a perturbation causes the system to travel outside a mode boundary and triggers an unexpected sensor change; e.g., detecting a transition from yellow  $\Rightarrow$  white instead of the desired yellow  $\Rightarrow$  pink will result in a new plan: white  $\Rightarrow$  yellow  $\Rightarrow$  pink  $\Rightarrow$  green. In this work, we synthesize a task automaton from a linear temporal logic formula (LTL) that specifies all valid mode transitions. We denote the problem of learning a policy that respects these mode transitions from demonstrations as *temporal logic imitation* (TLI). In contrast to temporal logic planning (TLP) [23], where the workspace is partitioned into connected convex cells with known boundaries, we do not know the precise mode boundaries. Consequently, the learned policy might prematurely exit the same mode repeatedly, causing the task automaton to loop without termination. To ensure any discrete plan generated by the automaton is feasible for the continuous policy, the bisimulation criteria [24, 25] must hold for the policy associated with each mode. Specifically, any continuous motion starting in any mode should stay in the same mode (**invariance**) until eventually reaching the next mode (**reachability**). The violations of these conditions are referred to as *invariance failures* and *reachability failures* respectively.

**Contributions** First, we investigate TLP in the setting of LfD and introduce TLI as a novel formulation to address covariate shift by proposing imitation with respect to a mode sequence instead of a motion sequence. Second, leveraging modes as the discrete abstraction, we prove that a state-based continuous behavior cloning (BC) policy with a global stability guarantee can be modulated to simulate any LTL-satisficing discrete plan. Third, we demonstrate that our approach LTL-DS, adapts to task-level perturbations via an LTL-satisficing automaton’s replanning and recovers from motion-level perturbations via DS’ stability during a multi-step, non-prehensile manipulation task.

## 2 Related Works

**Temporal Logic Motion Planning** LTL is a task specification language widely used in robot motion planning [26, 27, 28, 23]. Its ease of use and efficient conversion [29] to an automaton have spurred substantial research into TLP [25, 30, 31], which studies how to plan a continuous trajectory that satisfies an LTL formula. However, TLP typically assumes known workspace partitioning and boundaries *a priori*, both of which are unknown in the rarely explored TLI setting. While a robot can still plan in uncertain environments [32, 33], LfD bypasses the expensive search in high-dimensional space. Recent works [34, 35] have considered temporal logic formulas as side-information to demonstrations, but these formulas are treated as additional loss terms or rewards and

are not guaranteed to be satisfied. The key motivation for using LTL is to generate a reactive discrete plan, which can also be achieved by a finite state machine [14] or behavior tree [36].

**Behavior Cloning** We consider a subclass of LfD methods called state-based behavior cloning (BC) that learns the state-action distribution observed during demonstrations [37]. DAGGER [20], a BC-variant fixing covariate shift, could reduce the invariance failures depicted in Fig. 2, but requires online data collection, which our framework avoids with an LTL specification. To satisfy goal reachability, we employ a DS-based LfD technique [38]. Alternatives to this choice include certified NN-based methods [39, 40], DMPs [41], partially contracting DS [42], and Euclideanizing-flows [43]. To satisfy mode invariance, we modulate the learned DS to avoid invariance failure as state-space boundaries [44], similar to how barrier functions are learned to bound a controller [45, 46, 47]. **Multi-Step Manipulation** Prior LfD works [13, 14, 10, 48] tackle multi-step manipulation by segmenting demonstrations via a hidden Markov model. Using segmented motion trajectories, [13] learned a skill tree, [14] learned DMPs, [10] learned phase transitions, and [49] learned a task model. Most of these works assume a linear sequence of prehensile subtasks (pick-and-place) without considering how to replan when unexpected mode transitions happen. [48, 49] considered a non-prehensile scooping task similar to ours, but their reactivity only concerned collision avoidance in a single mode. [50, 6] improved BC policies with RL, but offered no guarantee of task success.

### 3 Temporal Logic Imitation: Problem Formulation

Let  $x \in \mathbb{R}^n$  represent the  $n$ -dimensional continuous state of a robotic system; e.g., the robot’s end-effector state in this work. Let  $\alpha = [\alpha_1, \dots, \alpha_m]^T \in \{0, 1\}^m$  be an  $m$ -dimensional discrete sensor state that uniquely identifies a mode  $\sigma = \mathcal{L}(\alpha)$ . We define a system state as a tuple,  $s = (x, \alpha) \in \mathbb{R}^n \times \{0, 1\}^m$ . Overloading the notation, we use  $\sigma \in \Sigma$ , where  $\Sigma = \{\sigma_i\}_{i=1}^M$ , to represent the set of all system states within the same mode—i.e.,  $\sigma_i = \{s = (x, \alpha) \mid \mathcal{L}(\alpha) = \sigma_i\}$ . In contrast, we use  $\delta_i = \{x \mid s = (x, \alpha) \in \sigma_i\}$  to represent the corresponding set of robot states. Note  $x$  cannot be one-to-one mapped to  $s$ , e.g., a level spoon can be either empty or holding soup. Each mode is associated with a goal-oriented policy, with goal  $x_i^* \in \mathbb{R}^n$ . A successful policy that accomplishes a multi-step task  $\tau$  with a corresponding LTL specification  $\phi$  can be written in the form:

$$\dot{x} = \pi(x, \alpha; \phi) = \sum_{i=1}^M \delta_{\Omega_\phi(\alpha)\sigma_i} f_i(x; \theta_i, x_i^*) \quad (1)$$

with  $\delta_{\Omega_\phi(\alpha)\sigma_i}$  being the Kronecker delta that activates a mode policy  $f_i(x; \theta_i, x_i^*) : \mathbb{R}^n \rightarrow \mathbb{R}^n$  encoded by learnable parameters  $\theta_i$  and goal  $x_i^*$ . Mode activation is guided by an LTL-equivalent automaton  $\Omega_\phi(\alpha) \rightarrow \sigma_i$  choosing the next mode  $\sigma_i$  based on the current sensor reading  $\alpha$ .

**Demonstrations** Let demonstrations for a task  $\tau$  be  $\Xi = \{\{x^{t,d}, \dot{x}^{t,d}, \alpha^{t,d}\}_{t=1}^{T_d}\}_{d=1}^D$  where  $x^{t,d}, \dot{x}^{t,d}, \alpha^{t,d}$  are robot state, velocity, and sensor state at time  $t$  in demonstration  $d$ , respectively, and  $T_d$  is the length of each  $d$ -th trajectory. A demonstration is successful if the continuous motion traces through a sequence of discrete modes that satisfies the corresponding LTL task specification.

**Perturbations** External perturbations, which many works in Sec. 2 avoid, constitute an integral part of our task complexity. Specifically, we consider (1) motion-level perturbations that displace a continuous motion within the same mode, and (2) task-level perturbations that drive the robot outside of the current mode. Critically, motion-level perturbations do not cause a plan change instantaneously, but they can lead to future unwanted mode transitions due to covariate shift.

**Problem Statement** Given (1) an LTL formula  $\phi$  specifying valid mode transitions for a task  $\tau$ , (2) sensors that detect each mode abstraction defined in  $\phi$ , and (3) successful demonstrations  $\Xi$ , we seek to learn a policy defined in Eq. 1 that generates continuous trajectories guaranteed to satisfy the LTL specification despite arbitrary external perturbations.

## 4 Preliminaries

### 4.1 LTL Task Specification

LTL formulas consist of atomic propositions (AP), logical operators, and temporal operators [51, 23]. Let  $\Pi$  be a set of Boolean variables; an infinite sequence of truth assignments to all APs in  $\Pi$  is called the trace  $[\Pi]$ . The notation  $[\Pi], t \models \phi$  means the truth assignment at time  $t$  satisfies the LTL formula  $\phi$ . Given  $\Pi$ , the minimal syntax of LTL can be described as:

$$\phi ::= p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \mathbf{X}\phi_1 \mid \phi_1 \mathbf{U}\phi_2 \quad (2)$$

where  $p$  is any AP in  $\Pi$ , and  $\phi_1$  and  $\phi_2$  are valid LTL formulas constructed from  $p$  using Eq. 2. The operator  $\mathbf{X}$  is read as ‘next,’ and  $\mathbf{X}\phi_1$  intuitively means the truth assignment to APs at the next time step sets  $\phi_1$  as true.  $\mathbf{U}$  is read as ‘until’ and, intuitively,  $\phi_1 \mathbf{U}\phi_2$  means the truth assignment to APs

sets  $\phi_1$  as true until  $\phi_2$  becomes true. Additionally, first-order logic operators  $\neg$  (not),  $\wedge$  (and),  $\vee$  (or), and  $\rightarrow$  (implies), as well as higher-order temporal operators  $\mathbf{F}$  (eventually), and  $\mathbf{G}$  (globally), are incorporated. Intuitively,  $\mathbf{F}\phi_1$  means the truth assignment to APs eventually renders  $\phi_1$  true and  $\mathbf{G}\phi_1$  means truth assignment to APs renders  $\phi_1$  always true from this time step onward.

#### 4.2 Task-Level Reactivity in LTL

To capture the reactive nature of a system given sensor measurements, the *generalized reactivity* (I) (GR(1)) fragment of LTL [29, 30] can be used. Let the set of all APs be  $\Pi = \mathcal{X} \cup \mathcal{Y}$ , where sensor states form environment APs  $\mathcal{X} = \{\alpha_1, \dots, \alpha_m\}$  and mode symbols form system APs  $\mathcal{Y} = \{\sigma_1, \dots, \sigma_l\}$ . A GR(1) formula is of the form  $\phi = (\phi_e \rightarrow \phi_s)$  [29], where  $\phi_e$  models the assumed environment behavior and  $\phi_s$  models the desired system behavior. Specifically,

$$\phi_e = \phi_i^e \wedge \phi_t^e \wedge \phi_g^e, \quad \phi_s = \phi_i^s \wedge \phi_t^s \wedge \phi_g^s \quad (3)$$

$\phi_i^e$  and  $\phi_i^s$  are non-temporal Boolean formulas that constrain the initial truth assignments of  $\mathcal{X}$  and  $\mathcal{Y}$  (e.g., the starting mode).  $\phi_t^s$  and  $\phi_t^e$  are LTL formulas categorized as safety specifications that describe how the system and environment should always behave (e.g., valid mode transitions).  $\phi_g^s$  and  $\phi_g^e$  are LTL formulas categorized as liveness specifications that describe what goal the system and environment should eventually achieve (e.g., task completion) [23]. The formula  $\phi$  guarantees the desired system behavior specified by  $\phi_s$  if the environment is *admissible*—i.e.,  $\phi_e$  is true—and can be converted to an automaton  $\Omega_\phi$  that plans a mode sequence satisfying  $\phi$  by construction [30].

#### 4.3 Motion-Level Reactivity in DS

Dynamical System [19] is a state-based BC method with a goal-reaching guarantee despite arbitrary perturbations. A DS policy can be learned from as few as a single demonstration and has the form:

$$\dot{x} = f(x) = \sum_{k=1}^K \gamma_k(x)(A^k x + b^k) \quad (4) \quad \begin{cases} (A^k)^T P + P A^k = Q^k, Q^k = (Q^k)^T \prec 0 \\ b^k = -A^k x^* \end{cases} \quad \forall k \quad (5)$$

where  $A^k \in \mathbb{R}^{n \times n}$ ,  $b^k \in \mathbb{R}^n$  are the  $k$ -th linear system parameters, and  $\gamma_k(x) : \mathbb{R}^n \rightarrow \mathbb{R}^+$  is the mixing function. To certify global asymptotic stability (G.A.S.) of Eq. 4, a Lyapunov function  $V(x) = (x - x^*)^T P (x - x^*)$  with  $P = P^T \succ 0$ , is used to derive the stability constraints in Eq. 5. Minimizing the fitting error of Eq. 4 with respect to demonstrations  $\Xi$  subject to constraints in Eq. 5 yields a non-linear DS with a stability guarantee [19]. To learn the optimal number  $K$  and mixing function  $\gamma_k(x)$  we use the Bayesian non-parametric GMM fitting approach presented in [19].

#### 4.4 Bisimulation between Discrete Plan and Continuous Policy

To certify a continuous policy will satisfy an LTL formula  $\phi$ , one can show the policy can simulate any LTL-satisficing discrete plan of mode sequence generated by  $\Omega_\phi$ . To that end, every mode's associated policy must satisfy the following bisimulation conditions [25, 23]:

**Condition 1 (Invariance).** *Every continuous motion starting in a mode must remain within the same mode while following the current mode's policy; i.e.,  $\forall i \forall t (s^0 \in \sigma_i \rightarrow s^t \in \sigma_i)$*

**Condition 2 (Reachability).** *Every continuous motion starting in a mode must reach the next mode in the demonstration while following the current mode's policy; i.e.,  $\forall i \exists T (s^0 \in \sigma_i \rightarrow s^T \in \sigma_j)$*

### 5 LTL-DS: Methodology

To solve the TLI problem in Sec. 3, we introduce a mode-based imitation policy—LTL-DS:

$$\dot{x} = \pi(x, \alpha; \phi) = \underbrace{\sum_{i=1}^M \delta_{\Omega_\phi(\alpha)\sigma_i}}_{\text{offline learning}} \underbrace{M_i(x; \Gamma_i(x), x_i^*)}_{\text{online learning}} \underbrace{f_i(x; \theta_i, x_i^*)}_{\text{offline learning}} \quad (6)$$

During offline learning, we synthesize the automaton  $\Omega_\phi$  from  $\phi$  as outlined in Sec. 4.2 and learn DS policies  $f_i$  from  $\Xi$  according to Sec. 4.3. While the choice of DS satisfies the reachability condition as explained later, nominal DS rollouts are not necessarily bounded within any region. Neither do we know mode boundaries in TLI. Therefore, an online learning phase is necessary, where for each mode policy  $f_i$  we learn an implicit function,  $\Gamma_i(x) : \mathbb{R}^n \rightarrow \mathbb{R}^+$ , that inner-approximates the mode boundary in the state-space of a robot  $x \in \mathbb{R}^n$ . With a learned  $\Gamma_i(x)$  for each mode, we can construct a modulation matrix  $M_i$  that ensures each modulated DS— $M_i f_i$ —is mode invariant.



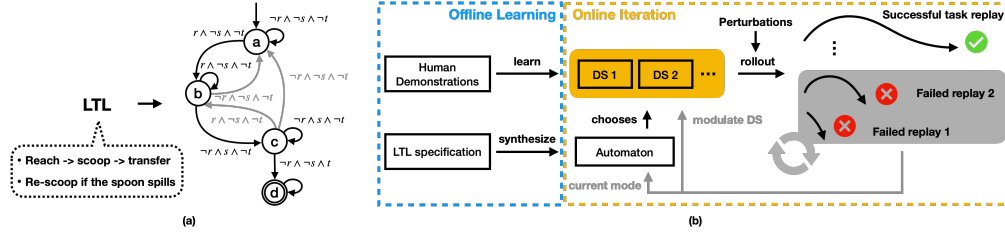


Figure 3: (a) Task automaton for a scooping task LTL. Mode  $a, b, c, d$  are reaching, scooping, transporting, and done mode respectively. Atomic proposition  $r, s, t$  denote sensing the spoon reaching the soup, soup on the spoon, and task success respectively. During successful demonstrations, only mode transitions in black,  $a \Rightarrow b \Rightarrow c \Rightarrow d$ , are observed. Additional valid transitions in gray,  $b \Rightarrow a$ ,  $c \Rightarrow a$ , and  $c \Rightarrow b$ , are given by the LTL to help recover from unexpected mode transitions. (b) System flowchart of LTL-DS.

## 5.1 Offline Learning Phase

**Synthesis of LTL-Satisficing Automaton** We convert an LTL to its equivalent automaton with [52], which plans the next mode given the current sensor reading. Assuming all possible initial conditions for the system are specified in the LTL, the automaton is always deployed from a legal state.

**Sensor-based Motion Segmentation and Attractor Identification** Given demonstrations in  $\Xi$  and accompanying sensor readings related to the set of  $\mathcal{M}$  modes, we can automatically segment the trajectories into  $\mathcal{M}$  clusters and corresponding attractor set  $X^*$ . Refer to Appendix C for details.

**Ensuring Goal Reachability with Learned DS Mode Policies** While any BC variant with a stability guarantee can satisfy reachability (see Sec. 2), we focus on the G.A.S. DS formulation and learning approach defined in Section 4.3 that ensures every  $x \in \mathbb{R}^n$  is guaranteed to reach  $x_i^*$ . By placing  $x_i^*$  within the boundary set of  $\delta_j$  for a mode  $\sigma_j$ , we ensure mode  $\sigma_j$  is reachable from every  $s$  in mode  $\sigma_i$ . Note  $f(x)$  cannot model sensor dynamics in  $\alpha$ . Yet, we employ mode abstraction to reduce the imitation of a system state trajectory in  $s$ —which includes the evolution of both the robot and sensor state—to just a robot state trajectory in  $x$ .

## 5.2 Online Learning Phase

**Iterative Mode Boundary Estimation via Invariance Failures** As shown in Fig. 2, DS can suffer from *invariance* failures in regions without data coverage. Instead of querying humans for more data in those regions [20], we leverage sparse events of mode exits detected by sensors to estimate the unknown mode boundary. Specifically, for each invariance failure, we construct a cut that separates the failure state,  $x^{T_f}$ , from the mode-entry state,  $x^0$ , the last in-mode state,  $x^{T_f-1}$ , and the mode attractor,  $x^*$ . We ensure this separation constraint with a quadratically constrained quadratic program (QCQP) that searches for the normal direction (pointing away from the mode) of a hyper-plane that passes through each  $x^{T_f-1}$  such that the plane’s distance to  $x^*$  is minimized. The intersection of half-spaces cut by hyper-planes inner approximates a convex mode boundary, as seen in Fig. 4. Adding cuts yields better boundary estimation, but is not necessary unless the original vector field flows out of the mode around those cuts. For more details, refer to Appendix E.3.

**Ensuring Mode Invariance by Modulating DS** We treat each cut as a collision boundary that deflects DS flows following the approach in [44, 53]. In our problem setting the mode boundary is analogous to a workspace enclosure rather than a task-space object. Let existing cuts form an implicit function,  $\Gamma(x) : \mathbb{R}^n \rightarrow \mathbb{R}^+$ , where  $\Gamma(x) < 1$ ,  $\Gamma(x) = 1$ ,  $\Gamma(x) > 1$  denote the estimated interior, the boundary and the exterior of a mode.  $0 < \Gamma(x) < \infty$  monotonically increases as  $x$  moves away from a reference point  $x^r$  inside the mode. For  $x$  outside the cuts, or inside but moving away from the cuts, we leave  $f(x)$  unchanged; otherwise, we modulate  $f(x)$  to not collide with any cuts as  $\dot{x} = M(x)f(x)$  by constructing a modulation matrix  $M(x)$  through eigenvalue decomposition:

$$\begin{cases} M(x) = E(x)D(x)E(x)^{-1}, & E(x) = [\mathbf{r}(x) \mathbf{e}_1(x) \dots \mathbf{e}_{d-1}(x)], & \mathbf{r}(x) = \frac{x-x^r}{\|x-x^r\|} \\ D(x) = \mathbf{diag}(\lambda_r(x), \lambda_{e_1}(x), \dots, \lambda_{e_{d-1}}(x)), & \lambda_r(x) = 1 - \Gamma(x), & \lambda_e(x) = 1 \end{cases} \quad (7)$$

The full-rank basis  $E(x)$  consists of a reference direction  $\mathbf{r}(x)$  stemming from  $x^r$  toward  $x$ , and  $d - 1$  directions spanning the hyperplane orthogonal to  $\nabla\Gamma(x)$ , which in this case is the closest cut to  $x$ . In other words, all directions  $\mathbf{e}_1(x) \dots \mathbf{e}_{d-1}(x)$  are tangent to the closest cut, except  $\mathbf{r}(x)$ . By modulating only the diagonal component,  $\lambda_r(x)$ , with  $\Gamma(x)$ , we have  $\lambda_r(x) \rightarrow 0$  as  $x$  approaches the closest cut, effectively zeroing out the velocity penetrating the cut while preserving velocity tangent to the cut. Consequently, a modulated DS will not repeat invariance failures that its nominal

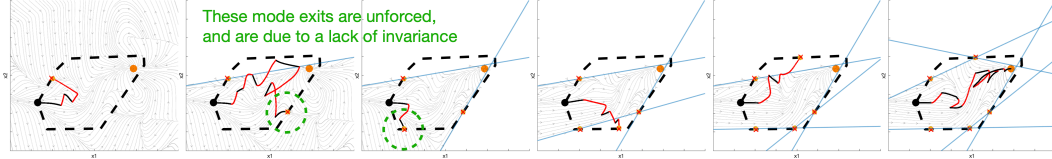


Figure 4: An illustration of iterative estimation of a mode boundary with cutting planes. A system enters a mode with an unknown boundary (dashed line) at the black circle, and is attracted to the goal at the orange circle. The trajectory in black shows the original policy rollout, and the trajectory in red is driven by perturbations. After the system exits the mode and before it eventually re-enters the same mode through replanning, a cut is placed at the last in-mode state (yellow circle) to bound the mode from the failure state (red cross). When the system is inside the cuts, it experiences modulated DS and never moves out of the cuts (flows moving into the cuts are not modulated); when the system is outside the cuts but inside the mode, it follows the nominal DS. Note only mode exits in black are invariance failures in need of modulation (green circles); mode exits in red are driven by perturbations to illustrate that more cuts lead to better boundary approximation.

counterpart experiences as long as the mode is bounded by cuts. Notice this modulation strategy is not limited to DS and can be applied to any state-based BC method to achieve mode invariance.

## 6 Proof

Next, we prove LTL-DS produces a continuous trajectory that satisfies an LTL specification. We start with assumptions and end with theorems. Detailed proofs are provided in Appendix A.

**Assumption 1.** *All modes are convex.*

This assumption leads to the existence of at least one cut—i.e., the supporting plane [54], which can separate a failure state on the boundary from any other state within the mode. A corollary is that the boundary shared by two modes, which we call a guard surface,  $G_{ij} = \delta_i \cap \delta_j$ , is also convex. Since all transitions out of a mode observed during demonstrations reside on the mode boundary, their average location, which we use as the attractor for the mode, will also be on the boundary.

**Assumption 2.** *There are a finite number of external perturbations of arbitrary magnitudes.*

Given zero perturbation, all BC methods should succeed in any task replay, as the policy rollout will always be in distribution. If there are infinitely many arbitrary perturbations, no BC methods will be able to reach a goal. In this work, we study the setting in between, where there are finitely many motion- and task-level perturbations causing unexpected mode exits. Environmental stochasticity is ignored, as its cumulative effects can also be simulated by external perturbations.

**Assumption 3.** *Perturbations only cause transitions to modes already seen in the demonstrations.*

While demonstrations of all valid mode transitions are not required, they must minimally cover all possible modes. If a system encounters a completely new sensor state during online interaction, it is reasonable to assume that no BC methods could recover from the mode unless more information about the environment is provided.

**Theorem 1.** (Key Contribution 1) *A nonlinear DS defined by Eq. 4, learned from demonstrations, and modulated by cutting planes as described in Section 5.2 with the reference point  $x^r$  set at the attractor  $x^*$ , will never penetrate the cuts and is G.A.S. at  $x^*$ . **Proof:** See Appendix A.  $\square$*

**Theorem 2.** (Key Contribution 2) *The continuous trace of system states generated by LTL-DS defined in Eq. 6 satisfies any LTL specification  $\phi$  under Asm. 1, 2, and 3. **Proof:** See Appendix A.  $\square$*

## 7 Experiments

### 7.1 Single-Mode Invariance and Reachability

We show quantitatively both reachability and invariance are necessary for task success. We compare DS and a NN-based BC policy (denoted as BC) to represent policies with and without a stability guarantee. Figure 5 shows that policy rollouts start to fail (turn red) as increasingly larger perturbations are applied to the starting states; however, DS only suffers from invariance failures, while BC suffers from both invariance and reachability failures (due to diverging flows and spurious attractors). Figure 5 (right) shows that all flows are bounded within the mode for both DS and BC after two cuts. In the case of DS, flows originally leaving the mode are now redirected to the attractor by the cuts; in the case of BC, while no flows leave the mode after modulation, spurious attractors are created, leading to reachability failures. This is a counterfactual illustration of Thm. 1, that policies without a stability guarantee are not G.A.S. after modulation. Figure 6 verifies this claim quantitatively and we empirically demonstrate that a stable policy requires only four modulation cuts to achieve a perfect success rate—which an unstable policy cannot be modulated to achieve.

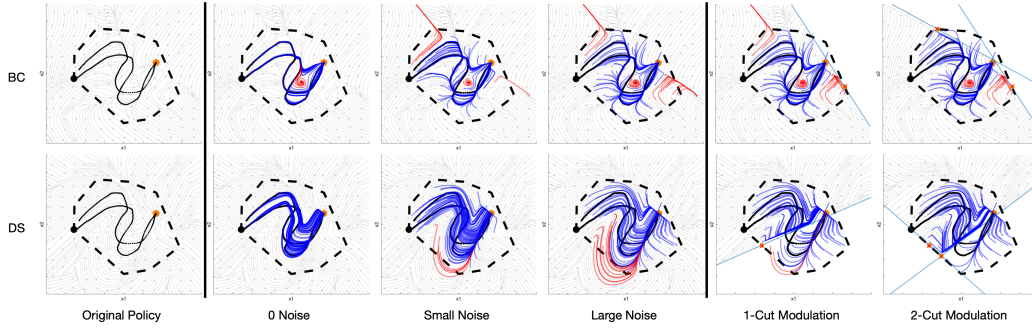


Figure 5: Policy rollouts from different starting states for a randomly generated convex mode. The top row shows BC results, and the bottom row depicts DS results. The left column visualizes the nominal policies learned from two demonstrations (black trajectories) reaching the orange attractor. The middle columns add different levels of Gaussian noise to the initial states sampled from the demonstration distribution. Blue trajectories successfully reach the attractor, while red trajectories fail due to either invariance failures or reachability failures. (Note that these failures only occur at locations without data coverage.) The right columns show that cutting planes (blue lines) separate failures (red crosses) from last-visited in-mode states (yellow circles) and consequently bound both policies to be mode-invariant. Applying cutting planes to BC policies without a stability guarantee cannot correct reachability failures within the mode. More results in Appendix E.

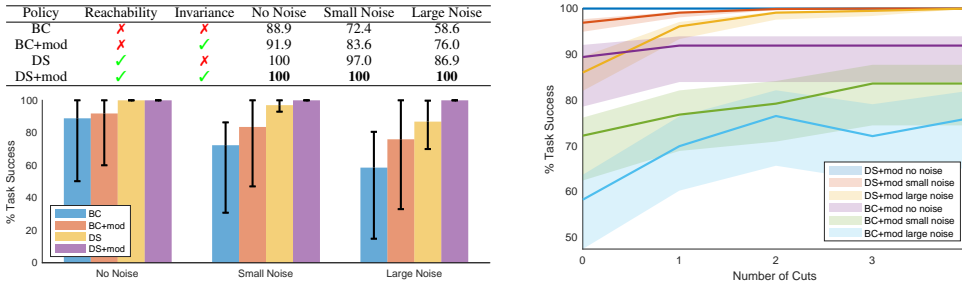


Figure 6: **(Left)** The success rate (%) of a single-mode reaching task. As we begin to sample out of distribution by adding more noise to the demonstrated states, the BC’s success rate degrades more rapidly than the DS’. After modulation, DS (+mod) maintains a success guarantee, which BC (+mod) falls short of due to the base policy’s lack of a stability guarantee. **(Right)** Empirically, the invariance of a single mode requires only a finite number of cuts for a nominal policy with a stability guarantee. Regardless of the noise level, DS achieves a 100% success rate after four cuts, while BC struggles to improve performance with additional cuts. Thick lines represent mean statistics and shaded regions show the interquartile range. More details in Appendix E.

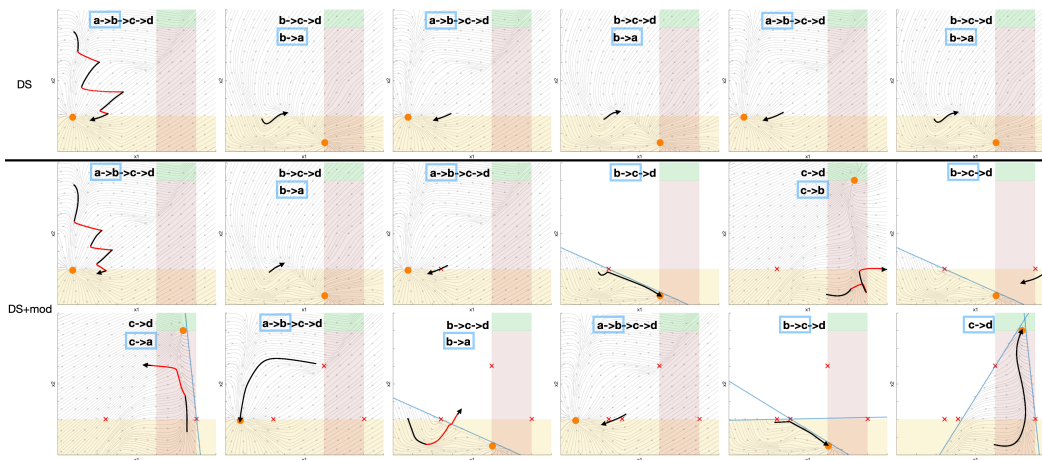


Figure 7: Rollouts of a multi-step scooping task under perturbations. The first row shows that sequencing DS policies with an automaton can lead to looping without boundary estimation. The second and third rows show that modulation prevents looping and enables the system to eventually reach the goal mode despite repeated perturbations. We show the desired discrete plan at the top of each sub-figure and annotate the current mode transition detected in the blue box. Black and red trajectories signify original and perturbed rollouts.

## 7.2 Multi-Modal Reactivity and Generalization to New Tasks

We now empirically demonstrate that having a reactive discrete plan alone is insufficient to guarantee task success without mode invariance for tasks with multiple modes. Consider the multi-modal soup-scooping task introduced in Fig. 2. Formally, we define three environment APs,  $r$ ,  $s$ ,  $t$ , sensing the spoon is in contact with the soup, has soup on it, and has arrived at a target location respectively. Given successful demonstrations, sensors will record discrete transitions  $(\neg r \wedge \neg s \wedge \neg t) \Rightarrow (r \wedge \neg s \wedge \neg t) \Rightarrow (\neg r \wedge s \wedge \neg t) \Rightarrow (\neg r \wedge \neg s \wedge t)$ , from which four unique sensor states are identified. We label each sensor state as a mode with robot AP  $a$  (reaching)  $\Rightarrow b$  (scooping)  $\Rightarrow c$  (transporting)  $\Rightarrow d$  (done). The Invariance of mode  $b$  enforces contact with soup during scooping, and the invariance of mode  $c$  constrains the spoon’s orientation to avoid spilling. We follow the TLP convention to assume LTL formulas are provided by domain experts (although they can also be learned from demonstrations [51, 55].) The specific LTL for the soup-scooping task is detailed in Appendix F, and can be converted into a task automaton as shown in Fig. 3. One might assume the automaton is sufficient to guarantee task success without modulation, as it only needs to replan a finite number of times assuming a finite number of perturbations; however, not enforcing mode invariance can lead to looping at the discrete level, and ultimately renders the goal unreachable, as depicted in the top row of Fig. 7. In contrast, looping is prevented when modulation is enabled, as the system experiences each invariance failure only once.

**Robot Experiments** First, we implement the soup-scooping task on a Franka Emika robot arm as shown in Fig. 1. We show in videos on our website that (1) DS allows our system to compliantly react to motion-level perturbations while ensuring system stability; (2) LTL allows our system to replan in order to recover from task-level perturbations; and (3) our modulation ensures the robot learns from previous invariance failures to avoid repeating them. To test robustness against unbiased perturbations, we collect 30 trials from 6 humans as seen in Appendix H. All trials succeed eventually in videos. We do not cherry-pick these results, and the empirical 100% success rate further corroborates our theoretic success guarantee. Second, we implement an inspection task as a permanent interactive exhibition at MIT Museum, with details documented in Appendix I. Lastly, we show a color tracing task testing different automaton structures with details in Appendix J.

**Generalization** LTL-DS can generalize to a new task by reusing learned DS if the new LTL shares the same set of modes. Consider another multi-step task of adding chicken and broccoli to a pot. Different humans might give demonstrations with different modal structures (e.g., adding chicken first vs adding broccoli first). LTL-DS can be reformulated to learn a policy for each mode transition (each mode can now have multiple policies), resulting in a collection of DS skills that can be flexibly recombined to solve new tasks. To generate different task LTLs, a human only needs to edit the  $\phi_t^s$  portion of the original LTL formula. We provide further details of this analysis in Appendix G.

## 8 Limitations

TLI assumes the existence of suitable mode abstractions, reactive logic formulas and perfect sensors to detect mode transitions, which can be difficult to obtain without non-trivial domain knowledge. Our work is based on the assumption that for well-defined tasks (e.g., assembly tasks in factory settings), domain expertise in the form of a logic formula is a cheaper knowledge source than collecting hundreds of motion trajectories to avoid covariate shift (we use up to 3 demonstrations in all experiments). Moreover, even when abstractions for a task are given by an oracle, an LfD method without either the invariance or the reachability property will not have a formal guarantee of successful task replay, which is this work’s focus. In future work, we will learn mode abstractions directly from sensor streams such as videos so that our approach gains more autonomy without losing reactivity.

## 9 Conclusion

In this paper, we formally introduce the problem of *temporal logic imitation* as imitating continuous motions that satisfy an LTL specification. We identify the fact that learned policies do not necessarily satisfy the bisimulation criteria as the main challenge of applying LfD methods to multi-step tasks. To address this issue, we propose a DS-based approach that can iteratively estimate mode boundaries to ensure invariance and reachability. Combining the task-level reactivity of LTL and the motion-level reactivity of DS, we arrive at an imitation learning system able to robustly perform various multi-step tasks under arbitrary perturbations given only a small number of demonstrations. We demonstrate our system’s practicality on a real Franka robot.



## Acknowledgments

We would like to thank Jon DeCastro, Chuchu Fan, Terry Suh, Rachel Holladay, Rohan Chitnis, Tom Silver, Yilun Zhou, Naomi Schurr, and Yuanzhen Pan for their invaluable advice and help.

## References

- [1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
- [2] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 2020.
- [3] S. Ekvall and D. Kragic. Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):33, 2008.
- [4] D. H. Grollman and O. C. Jenkins. Incremental learning of subtasks from unsegmented demonstration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 261–266. IEEE, 2010.
- [5] J. R. Medina and A. Billard. Learning stable task sequences from demonstration with linear parameter varying systems and hidden markov models. In *Conference on Robot Learning*, pages 175–184. PMLR, 2017.
- [6] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.
- [7] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.
- [8] S. Pirk, K. Hausman, A. Toshev, and M. Khansari. Modeling long-horizon tasks as sequential interaction landscapes. *arXiv preprint arXiv:2006.04843*, 2020.
- [9] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 309–315. IEEE, 2012.
- [10] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 1503–1510. IEEE, 2015.
- [11] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398, 2012.
- [12] C. Pérez-D’Arpino and J. A. Shah. C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4058–4065. IEEE, 2017.
- [13] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.
- [14] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, pages 10–15607. Berlin, Germany, 2013.
- [15] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.
- [16] N. B. Figueroa Fernandez. From high-level to low-level robot learning of complex tasks: Leveraging priors, metrics and dynamical systems. Technical report, EPFL, 2019.

- [17] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel. Dynamic movement primitives in robotics: A tutorial survey. *CoRR*, abs/2102.03861, 2021. URL <https://arxiv.org/abs/2102.03861>.
- [18] A. Billard, S. S. Mirrazavi Salehian, and N. Figueroa. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. MIT Press, Cambridge, USA, 2022.
- [19] N. Figueroa and A. Billard. A physically-consistent bayesian non-parametric mixture model for dynamical system learning. In *CoRL*, pages 927–946, 2018.
- [20] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [21] A. J. Van Der Schaft and J. M. Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer London, 2000.
- [22] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.
- [23] H. Kress-Gazit, M. Lahijanian, and V. Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:211–236, 2018.
- [24] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.
- [25] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.
- [26] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70, 2007.
- [27] E. M. Wolff, U. Topcu, and R. M. Murray. Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4332–4339. IEEE, 2013.
- [28] E. Plaku and S. Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI communications*, 29(1):151–162, 2016.
- [29] N. Piterman, A. Pnueli, and Y. Sa’ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.
- [30] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.
- [31] J. A. DeCastro and H. Kress-Gazit. Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors. *The International Journal of Robotics Research*, 34(3): 378–394, 2015.
- [32] A. M. Ayala, S. B. Andersson, and C. Belta. Temporal logic motion planning in unknown environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5279–5284. IEEE, 2013.
- [33] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi. Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics*, 32(3):583–599, 2016.
- [34] C. Innes and S. Ramamoorthy. Elaborating on learned demonstrations with temporal logic specifications. *arXiv preprint arXiv:2002.00784*, 2020.

- [35] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis. Learning from demonstrations using signal temporal logic in stochastic and continuous domains. *IEEE Robotics and Automation Letters*, 6(4):6250–6257, 2021.
- [36] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy. Reactive task and motion planning under temporal logic specifications. *arXiv preprint arXiv:2103.14464*, 2021.
- [37] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018. ISSN 1935-8253. doi:10.1561/23000000053.
- [38] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.
- [39] K. Neumann, A. Lemme, and J. J. Steil. Neural learning of stable dynamical systems based on data-driven lyapunov candidates. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1216–1222. IEEE, 2013.
- [40] C. Dawson, Z. Qin, S. Gao, and C. Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.
- [41] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.
- [42] H. Ravichandar, I. Salehi, and A. Dani. Learning partially contracting dynamical systems from demonstrations. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 369–378. PMLR, 13–15 Nov 2017. URL <https://proceedings.mlr.press/v78/ravichandar17a.html>.
- [43] M. A. Rana, A. Li, H. Ravichandar, M. Mukadam, S. Chernova, D. Fox, B. Boots, and N. Ratliff. Learning reactive motion policies in multiple task spaces from human demonstrations. In *Conference on Robot Learning*, pages 1457–1468. PMLR, 2020.
- [44] S. M. Khansari-Zadeh and A. Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 32(4):433–454, 2012.
- [45] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724. IEEE, 2020.
- [46] M. Saveriano and D. Lee. Learning barrier functions for constrained motion planning with dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 112–119. IEEE, 2019.
- [47] C. Dawson, S. Gao, and C. Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods. *arXiv preprint arXiv:2202.11762*, 2022.
- [48] G. Ye and R. Alterovitz. Demonstration-guided motion planning. In *Robotics research*, pages 291–307. Springer, 2017.
- [49] C. Bowen and R. Alterovitz. Closed-loop global motion planning for reactive execution of learned tasks. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1754–1760. IEEE, 2014.
- [50] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.
- [51] A. J. Shah, P. Kamath, S. Li, and J. A. Shah. Bayesian inference of temporal task specifications from demonstrations. 2018.

- [52] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and  $\omega$ -automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, Oct. 2016. doi:10.1007/978-3-319-46520-3\_8.
- [53] L. Huber, A. Billard, and J.-J. Slotine. Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters*, 4(2):1462–1469, 2019.
- [54] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [55] G. Chou, N. Ozay, and D. Berenson. Learning temporal logic formulas from suboptimal demonstrations: theory and experiments. *Autonomous Robots*, pages 1–26, 2021.
- [56] D. Kasenberg and M. Scheutz. Interpretable apprenticeship learning with temporal logic specifications. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4914–4921. IEEE, 2017.
- [57] A. Billard, S. Mirrazavi, and N. Figueroa. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. 2022.



## A Proofs

**Theorem 1.** (Key Contribution 1) *A nonlinear DS defined by Eq. 4, learned from demonstrations, and modulated by cutting planes as described in Section 5.2 with the reference point  $x^r$  set at the attractor  $x^*$ , will never penetrate the cuts and is G.A.S. at  $x^*$ .*

*Proof* Let the region bounded by cuts be  $\mathcal{D}$ , which is non-empty as it contains at least one demonstration. If  $x \notin \mathcal{D}$ , i.e.,  $x$  is outside the cuts, the nominal DS  $f(x)$  will not be modulated. Since  $f(x)$  is G.A.S. at  $x^*$  and  $x^* \in \mathcal{D}$ , a robot state at  $x$  will enter  $\mathcal{D}$  in a finite amount of time. If  $x \in \mathcal{D}$  and  $[E(x)^{-1}f(x)]_1 < 0$ , which corresponds to  $f(x)$  having a negative component in the direction of  $\mathbf{r}^*(x) = \frac{x-x^*}{\|x-x^*\|}$ ,  $f(x)$  is moving away from cuts and toward the attractor. In this case, we leave  $f(x)$  unmodulated and the original G.A.S. property holds true. If  $x \in \mathcal{D}$  and  $[E(x)^{-1}f(x)]_1 \geq 0$ , where the nominal DS could flow out of the cuts, we apply modulation—and, by construction,  $M(x)f(x)$  stays inside the cuts. To prove the stability of the modulated DS, we show that the Lyapunov candidate,  $V(x) = (x - x^*)^T P(x - x^*)$ , in satisfying  $\dot{V}(x) = \frac{\partial V(x)}{\partial x} f(x) < 0$  for  $f(x)$ , also satisfies the Lyapunov condition for  $M(x)f(x)$  (omitting matrix dependency upon  $x$  to reduce clutter):

$$\begin{aligned}
\dot{V}(x) &= \frac{\partial V(x)}{\partial x} M f(x) = \frac{\partial V(x)}{\partial x} E D E^{-1} f(x) \\
&= \frac{\partial V(x)}{\partial x} E \mathbf{diag}(1 - \Gamma(x), 1, \dots, 1) E^{-1} f(x) \\
&= \frac{\partial V(x)}{\partial x} f(x) - \frac{\partial V(x)}{\partial x} E \mathbf{diag}(\Gamma(x), 0, \dots, 0) E^{-1} f(x) \\
&< 0 - \frac{\partial V(x)}{\partial x} \mathbf{r}^*(x)^T \Gamma(x) [E^{-1} f(x)]_1 \\
&< 0 - \underbrace{2(x - x^*)^T P \frac{x - x^*}{\|x - x^*\|}}_{>0 \text{ as } P > 0} \underbrace{\Gamma(x)}_{>0} \underbrace{[E^{-1} f(x)]_1}_{\geq 0} < 0
\end{aligned} \tag{8}$$

Therefore,  $M(x)f(x)$  is G.A.S.  $\square$

The following lemmas support the proof of **Theorem 2**.

**Lemma 1.** *LTL-DS generates a discrete reactive plan of mode sequence that satisfies any LTL formula provided to the algorithm.*

*Proof* Since the task automaton is converted from an LTL formula, all resulting discrete plans of mode sequence (including the replanned sequence caused by perturbations) are correct by construction as long as the environment is admissible.  $\square$

**Lemma 2.** *If a mode transition  $\sigma_i \Rightarrow \sigma_j$  has been observed in the demonstrations,  $\sigma_j$  is reachable from  $\sigma_i$  by DS  $f_i$ .*

*Proof* Since  $\sigma_i \Rightarrow \sigma_j$  has been demonstrated,  $\sigma_i$  and  $\sigma_j$  must be connected; let them share a guard,  $G_{ij}$ . Assigning a globally stable DS  $f(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^n$  to each mode  $\sigma_i$  with region  $\delta_i \subset \mathbb{R}^n$  guarantees asymptotic convergence of all  $x$  in  $\delta_i$  to the attractor,  $x_i^*$  by DS  $f_i$ . Placing  $x_i^*$  on guard  $G_{ij}$  ensures that  $x_i^* \in \delta_j$ , and thus  $\forall s \sigma_i \Rightarrow \sigma_j$  as  $x \rightarrow x_i^*$ .  $\square$

**Lemma 3.** *If an unseen mode transition  $\sigma_i \Rightarrow \sigma_j$  occurs unexpectedly, the system will not be stuck in  $\sigma_j$ .*

*Proof* While the transition  $\sigma_i \Rightarrow \sigma_j$  has not been seen in demonstrations, Asm. 3 ensures that mode  $\sigma_j$  has been observed and its associated DS  $f_j$  has been learned. Since the LTL GR(1) fragment does not permit clauses in the form of  $(\mathbf{F} \mathbf{G} \phi)$ , which states  $\phi$  is eventually globally true (i.e., the system can stay in  $\sigma_j$  forever), every discrete plan will have to in finite steps result in  $\sigma_j \Rightarrow \sigma_k$  for some  $k, j \neq k$ . Having learned  $f_j$  also validates the existence of  $x_j^*$ —and, thus, a continuous trajectory toward  $G_{jk}$ .  $\square$

**Theorem 2.** (Key Contribution 2) *The continuous trace of system states generated by LTL-DS satisfies any LTL specification  $\phi$  under Asm. 1, 2, and 3.*

*Proof* Lemma 1 proves any discrete plan generated by LTL-DS satisfies the LTL specification. Lemmas 2 and 3 and Asm. 2 ensure the reachability condition of all modes. Thm. 1 certifies the modulated DS will be bounded inside the cuts, and thus the mode these cuts inner-approximate. Consequently, a finite number of external perturbations only require a finite number of cuts in order to ensure mode invariance. Given that bisimulation is fulfilled, the continuous trace generated by LTL-DS simulates a LTL-satisficing discrete plan, and thus satisfies the LTL.  $\square$

## B Motivation for Mode-based Imitation

Our work aims to achieve generalization in regions of the state space not covered by initial demonstrations. A well-studied line of research is to collect more expert data [20] so that the policy will learn to recover from out-of-distribution states. Our central observation in Fig. 2 is that there exists some threshold that separates trajectories deviating from expert demonstrations (black) into successes (blue) and failures (red). The threshold can be embodied in mode boundaries, which lead to the notion of a discrete mode sequence that acts as the fundamental success criteria for any continuous motions. In fact, online data collection to improve policies in DAGGER [20] can be seen as implicitly enforcing mode invariance. We take the alternative approach of explicitly estimating mode boundaries and shift the burden from querying for more data to querying for a task automaton in the language of LTL.

## C Sensor-based Motion Segmentation and Attractor Identification

time step	1	2	3	4	5	6	7	8	9	10
demo 1	$x^{1,1}$	$x^{2,1}$	$x^{3,1}$	$x^{4,1}$	$x^{5,1}$	$x^{6,1}$	$x^{7,1}$	$x^{8,1}$	$x^{9,1}$	$x^{10,1}$
demo 2	$x^{1,2}$	$x^{2,2}$	$x^{3,2}$	$x^{4,2}$	$x^{5,2}$	$x^{6,2}$	$x^{7,2}$	$x^{8,2}$	$x^{9,2}$	$x^{10,2}$

Table 1: Demonstrations are segmented into three AP regions (shown by color) based on three unique sensor states for DS learning. We use the average location of the last states (transition states to the next AP) in each AP as the attractor for the corresponding DS.

Let  $\{\{x^{t,k}, \dot{x}^{t,k}, \alpha^{t,k}\}_{t=1}^{T_k}\}_{k=1}^K$  be  $K$  demonstrations of length  $T_k$ . The motion trajectories in  $x^{t,k}$  are clustered and segmented into the same AP region if they share the same sensor state  $\alpha^{t,k}$ . For example, in Table. 1 two demonstrations of ten time steps form three AP regions (colored by red, blue and green) based on three unique sensor readings. To obtain the attractor for each of the three DS to be learned, we average the last state in the trajectory segment. For instance, the average location of  $x^{2,1}$  and  $x^{4,2}$ ,  $x^{6,1}$  and  $x^{9,2}$ ,  $x^{10,1}$  and  $x^{10,2}$  become the attractor for the red, blue and green AP’s DS respectively.

## D Relation of TLI to Prior Work

This work explores a novel LfD problem formulation (temporal logic imitation) that is closely related to three research communities. First, there is a large body of work on learning task specifications in the form of LTL formulas from demonstrations [51, 55, 56]. We do not repeat their endeavor in this work and assume the LTL formulas are given. Second, given LTL formulas there is another community (temporal logic planning) that studies how to plan a continuous trajectory that satisfies the given LTL [26, 27, 28, 23]. Their assumption of known abstraction boundaries and known dynamics allow the planned trajectory to satisfy the invariance and reachability (bisimulation) criteria respectively, thus certifying the planned continuous trajectory will satisfy any discrete plan. Our observation is that the bisimulation criteria can also be used to certify that a LfD policy can simulate the discrete plan encoded by any LTL formula, which we dub as the problem of TLI. To the best of our knowledge, our work is the first to formalize TLI and investigate its unique challenges inherited from the LfD setting. On the one hand, we no longer have dynamics to plan with but we have reference trajectories to imitate. To satisfy reachability, it is necessary to leverage a third body of work—LfD methods with global stability guarantee (DS) [38, 19, 57]. On the other hand, we note LfD methods typically do not satisfy mode invariance due to unknown mode boundaries that are also innate to the LfD setting. Thus, we propose learning an approximate mode boundary leveraging sparse sensor events and then modulating the learned policies to be mode invariant. We prove

DS policies in particular after modulation will still satisfy reachability, and consequently certifying they will satisfy any given LTL formulas. Figure 8 summarizes how TLI’s relationship to prior work, where gray dashed boxes represent prior work and yellow dashed box highlights our contribution.

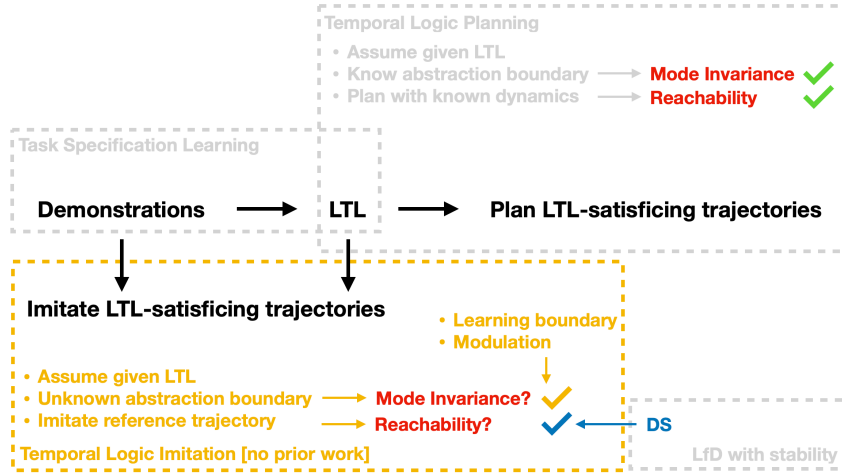


Figure 8: How TLI fits in relation to prior work, where gray dashed boxes represent prior work and yellow dashed box highlights our contribution.

## E Single-mode Experiments

### E.1 Experiment Details

We randomly generate convex modes and draw 1 – 3 human demonstrations, as seen in Fig. 5 (left). To check mode invariance, we sample starting states from the demonstration distribution perturbed by Gaussian noise with standard deviation of 0%, 5%, and 30% of the workspace dimension. Sampling with zero noise corresponds to sampling directly on the demonstration states, and sampling with a large amount of noise corresponds to sampling from the entire mode region. To enforce invariance, we iteratively sample a failure state and add a cut until all invariance failures are corrected. A task replay is successful if and only if an execution trajectory both reaches the goal and stays within the mode. For each randomly generated convex mode, we sampled 100 starting states and computed the average success rate for 50 trials. We show DS+modulation ensures both reachability and invariance for five additional randomly sampled convex modes in Fig. 9.

### E.2 BC Policy Architecture and Training Details

For the Neural-network-based BC policy, we use an MLP architecture that consists of 2 hidden layers both with 100 neurons followed by ReLU activations. We use tanh as the output activation, and we re-scale the output from tanh layer to [-50 – 50]. Each demonstration trajectory consists of about 200 pairs of states and velocities as the training data to the network. Since we are training a state-based policy that predicts velocities from input states, we treat these data points as i.i.d. For training, we use Adam as the optimizer with a learning rate of 1e-3 for max 5000 epochs.

### E.3 QCQP Optimization Details

To find the normal direction of a hyperplane that goes through a last-in-mode states  $x^{T_f-1}$  in Sec. 5.2, we solve the following optimization problem, where  $w$  is the normal direction we are searching over;  $f = 1, 2, \dots$  indexes a set of failure states; and  $T_f$  is the corresponding time-step of first

detecting an invariance failure ( $T$  alone is used as matrix transpose.)

$$\begin{aligned}
& \min_w (w^T(x^* - x^{T_f-1}))^2 \\
& \text{s.t. } \|w\| = 1 \\
& w^T(x^* - x^{T_f-1}) \leq 0 \\
& w^T(x^0 - x^{T_f-1}) \leq 0 \\
& w^T(x^{T_{f'}-1} - x^{T_f-1}) \leq 0 \quad \forall f'
\end{aligned} \tag{9}$$

While specialized QCQP packages can be used to solve this optimization problem, we use a generic nonlinear Matlab function `fmincon` to solve for  $w$  in our implementation.

## F Multi-modal Experiments

After abstractions—environment APs,  $r, s, t$ , and robot APs,  $a, b, c, d$ —for the soup-scooping task in Sec. 7.2 are defined, the reactive LTL formula can be written as  $\phi = ((\phi_i^e \wedge \phi_t^e \wedge \phi_g^e) \rightarrow (\phi_i^s \wedge \phi_t^s \wedge \phi_g^s))$ .  $\phi_i^s$  and  $\phi_i^e$  specify the system’s initial mode,  $a$ , and the corresponding sensor state.  $\phi_g^s$  and  $\phi_g^e$  set mode  $d$  as the eventual goal for the robot, with no particular goal for the environment.  $\phi_t^e$  specifies the environmental constraints that determine which sensor states are true in each mode, as well as the fact that the system can only be in one mode at any times.  $\phi_i^e$  specifies all valid transitions for each mode.

$$\begin{aligned}
& \phi_i^e = \neg r \wedge \neg s \wedge \neg t; \quad \phi_g^e = True; \\
& \phi_t^e = \mathbf{G}((a \leftrightarrow (\neg r \wedge \neg s \wedge \neg t)) \wedge (b \leftrightarrow (r \wedge \neg s \wedge \neg t)) \\
& \quad \wedge (c \leftrightarrow (\neg r \wedge s \wedge \neg t)) \wedge (d \leftrightarrow (\neg r \wedge \neg s \wedge t)) \\
& \quad \wedge \mathbf{G}((a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \\
& \quad \vee (\neg a \wedge \neg b \wedge c \wedge \neg d) \vee (\neg a \wedge \neg b \wedge \neg c \wedge d)); \\
& \phi_i^s = a; \quad \phi_g^s = \mathbf{GF}d; \\
& \phi_t^s = \mathbf{G}((a \rightarrow (\mathbf{X}a \vee \mathbf{X}b)) \wedge (b \rightarrow (\mathbf{X}a \vee \mathbf{X}b \vee \mathbf{X}c)) \\
& \quad \wedge (c \rightarrow (\mathbf{X}a \vee \mathbf{X}b \vee \mathbf{X}c \vee \mathbf{X}d)) \wedge (d \rightarrow \mathbf{X}d));
\end{aligned}$$

**Automatic construction of GR(1) LTL formulas** One benefit of using the GR(1) fragment of LTL is that it provides a well-defined template for defining a system’s reactivity [30]<sup>1</sup>. While in this work we follow the TLP convention that assumes the full GR(1) formulas are given, the majority of these formulas can actually be automatically generated if Asm. 3 holds true. Specifically, once the abstraction,  $r, s, t, a, b, c, d$  is defined, formulas  $\phi_i^e, \phi_g^e$  are correspondingly defined as shown above, and they remain the same for different demonstrations. If a demonstration displaying  $a \Rightarrow b \Rightarrow c \Rightarrow d$  is subsequently recorded, formulas  $\phi_i^e, \phi_i^s, \phi_g^s$  as shown above can then be inferred. Additionally a partial formula  $\phi_i^s = \mathbf{G}((a \rightarrow (\mathbf{X}a \vee \mathbf{X}b)) \wedge (b \rightarrow (\mathbf{X}b \vee \mathbf{X}c)) \wedge (c \rightarrow (\mathbf{X}c \vee \mathbf{X}d)) \wedge (d \rightarrow \mathbf{X}d))$  can be inferred. Synthesis from this partial  $\phi = ((\phi_i^e \wedge \phi_t^e \wedge \phi_g^e) \rightarrow (\phi_i^s \wedge \phi_t^s \wedge \phi_g^s))$  results in a partial automaton in Fig. 3 with only black edges. During online iteration, if perturbations cause unexpected transitions,  $b \Rightarrow a$  and/or  $c \Rightarrow a$  and/or  $c \Rightarrow b$ , which are previously not observed in the demonstration,  $\phi_i^s$  will be modified to incorporate those newly observed transitions as valid mode switches, and a new automaton will be re-synthesized from the updated formula  $\phi$ . The gray edges in Fig. 3 reflect those updates after invariance failures are experienced. Asm. 3 ensures the completeness of the demonstrations with respect to modes, i.e., the initially synthesized automaton might be missing edges but not nodes compared to an automaton synthesized from the ground-truth full formula. For general ground-truth LTL formulas not part of the GR(1) fragment or demonstrations not necessarily satisfying Asm. 3, we cannot construct the formulas using the procedure outlined above. In that case, we can learn the formulas from demonstrations in a separate stage [51, 55].

In this work, we assume full LTL formulas are provided by domain experts. Since they are full specifications of tasks, the resulting automata will be complete w.r.t. all valid mode transitions (e.g., including both the black and gray edges in Fig. 3), and will only need to be synthesized once. Given the soup-scooping LTL defined above, we ran 10 experiments, generating 1–3 demonstrations

<sup>1</sup>The main benefit is that GR(1) formulas allow fast synthesis of their equivalent automata [29].



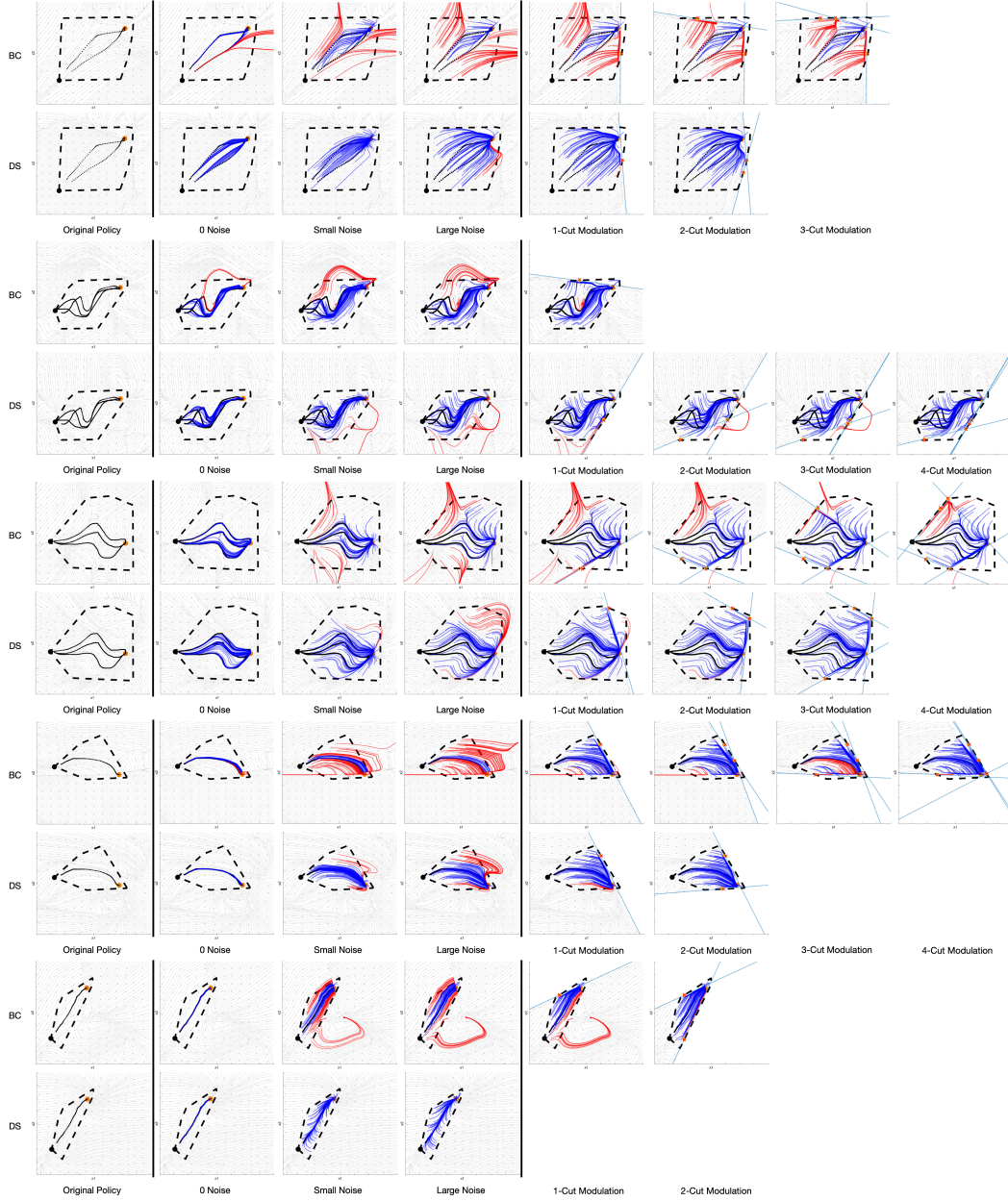


Figure 9: For each convex mode, we use 1-3 demonstrations for learning shown in black. Successful rollouts are shown in blue while unsuccessful rollouts are shown in red. We apply modulation to the large noise case and within four cuts all DS policies are modulated to be mode invariant. While BC policies can also be modulated to be mode invariant, they still suffer from existing reachability failures prior to modulation as well as new reachability failures introduced by modulation. For example, in BC flows that are originally flowing out of the mode can lead to spurious attractors at the cuts after modulation. We prove this will not happen for DS due to its stability guarantee.

for each, and learning a DS per mode. We applied perturbations uniformly sampled in all directions of any magnitudes up to the dimension of the entire workspace in order to empirically verify the task-success guarantee. We follow the QCQP optimization defined in Appendix B to find cuts to modulate the DS. Simulation videos can be found on the project page.

## G Generalization Results

LTL-DS can generalize to a new task by reusing learned DS if the new LTL shares the same set of modes. Consider another multi-step task of adding chicken and broccoli to a pot. Different humans might give demonstrations with different modal structures (e.g., adding chicken vs adding broccoli first) as seen in Fig. 10 (a). LTL-DS learns individual DS which can be flexibly combined to solve new tasks with new task automatons, as illustrated in Fig. 10 (c-f). To get these different task automatons, a human just needs to edit the  $\phi_t^s$  portion of the LTL formulas differently.

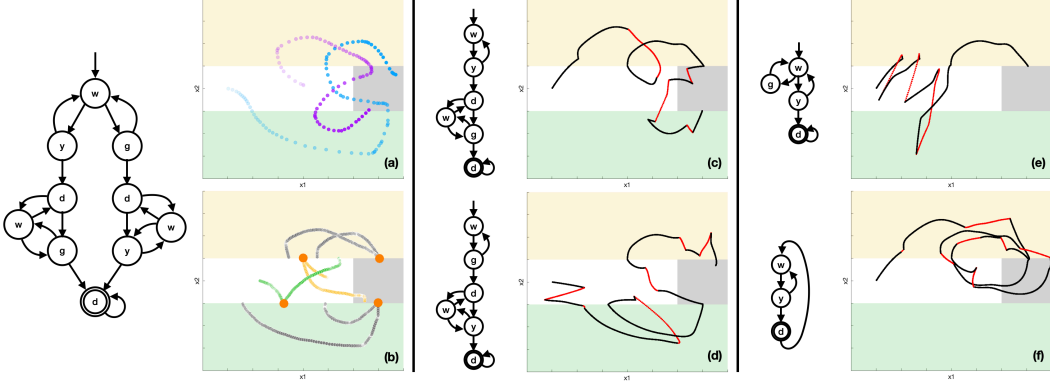


Figure 10: Reuse learned DS for new tasks by editing their task automaton via LTL. (a) Two demonstrations (purple and blue) solve a task of adding both chicken (yellow region) and broccoli (green region) to a pot (dark region) in different order as visualized by the automaton on the left. (b) Demonstrations are segmented into parts by mode region, which are then shifted to their average attractor locations (orange circles) for DS learning. Since a mode can proceed to different next modes in this case, we adapt the mode-based DS policy to be associated with each unique mode transition instead. The learned DS can generalize to new tasks given new automatons specifying (c) the order: white ( $w$ )  $\Rightarrow$  yellow ( $y$ )  $\Rightarrow$  dark ( $d$ )  $\Rightarrow$  green ( $g$ )  $\Rightarrow$  dark ( $d$ ), (d) the order: white  $\Rightarrow$  green  $\Rightarrow$  dark  $\Rightarrow$  yellow  $\Rightarrow$  dark, (e) the goal: getting only chicken, or (f) the goal: getting chicken continuously. The trajectory in red shows perturbations and the trajectory in black shows recovery according to the automaton structure.

We describe LTL formulas for variants of the cooking task of adding chicken and broccoli to a pot as visualized in Fig. 10. We use mode AP  $w$ ,  $y$ ,  $g$ ,  $d$  to define configurations of empty spoon (white region), transferring chicken (yellow region), transferring broccoli (green region), and finally dropping food in the pot (dark region). We follow the description of scooping task LTL to define  $\phi_i^e$ ,  $\phi_t^e$ ,  $\phi_g^e$ ,  $\phi_i^s$ ,  $\phi_g^s$  for the cooking tasks, which are shared by them all. We focus on  $\phi_t^s$  here as it captures mode transitions and is different for a different task. We denote the  $\phi_t^s$  portion of LTL for the new task of adding chicken first, adding broccoli first, adding chicken only, and adding chicken continuously as  $\phi_{cb}$ ,  $\phi_{bc}$ ,  $\phi_c$ , and  $\phi_{cc}$  respectively.

$$\begin{aligned}
 \phi_{cb} &= \mathbf{G}((w_1 \rightarrow \mathbf{X}y) \wedge (y \rightarrow (\mathbf{X}w_1 \vee \mathbf{X}d_1)) \\
 &\quad \wedge (d_1 \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}g)) \wedge (w_2 \rightarrow \mathbf{X}g) \\
 &\quad \wedge (g \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}d_2)) \wedge (d_2 \rightarrow \mathbf{X}d_2)); \\
 \phi_{bc} &= \mathbf{G}((w_1 \rightarrow \mathbf{X}g) \wedge (g \rightarrow (\mathbf{X}w_1 \vee \mathbf{X}d_1)) \\
 &\quad \wedge (d_1 \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}y)) \wedge (w_2 \rightarrow \mathbf{X}y) \\
 &\quad \wedge (y \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}d_2)) \wedge (d_2 \rightarrow \mathbf{X}d_2)); \\
 \phi_c &= \mathbf{G}((w \rightarrow (\mathbf{X}y \vee \mathbf{X}g)) \wedge (y \rightarrow (\mathbf{X}w \vee \mathbf{X}d)) \\
 &\quad \wedge (g \rightarrow \mathbf{X}w) \wedge (d \rightarrow \mathbf{X}d)); \\
 \phi_{cc} &= \mathbf{G}((w \rightarrow \mathbf{X}y) \wedge (y \rightarrow (\mathbf{X}w \vee \mathbf{X}d)) \wedge (d \rightarrow \mathbf{X}w));
 \end{aligned}$$

Note mode  $w_1$  and  $w_2$  denote visiting the white region before and after some food has been added to the pot and they share the same motion policy. The same goes for mode  $d_1$  and  $d_2$ . These formulas can be converted to task automatons in Fig. 10. We show animations of these tasks on the project page.

## H Robot Experiment 1: Soup-Scooping

We implemented the soup-scooping task on a Franka Emika robot arm. As depicted in Fig. 1, the task was to transport the soup (represented by the red beads) from one bowl to the other. Two demonstration trajectories were provided to the robot via kinesthetic teaching, from which we learned a DS to represent the desired evolution of the robot end-effector for each mode. The target velocity,  $\dot{x}$ , predicted by the DS was integrated to generate the target pose, which was then tracked by a Cartesian pose impedance controller. The robot state,  $x$ , was provided by the control interface. Sensor AP  $r$  tracked the mode transition when the spoon made contact with the soup, and sensor AP  $t$  tracked the mode transition when the spoon reached the region above the target bowl.  $r$  and  $t$  became true when the distance between the spoon and the centers of the soup bowl and the target bowl (respectively) were below a hand-tuned threshold. Sensor AP  $s$  became true when red beads were detected either from a wrist camera via color detection or through real-time human annotation. We visualize the modulation of robot DS in three dimensions— $y$ ,  $z$ , and pitch—in Fig. 11.

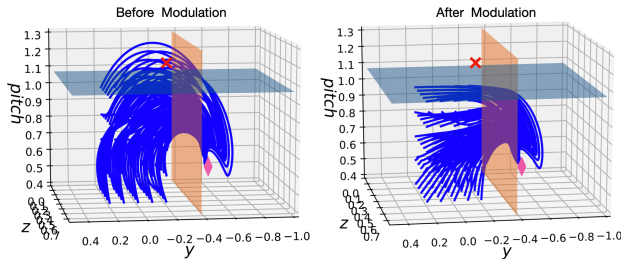


Figure 11: Modulation of sampled robot trajectories. The orange plane represents a guard surface between the transporting mode and the done mode, and the blue plane represents the mode boundary for the transporting mode. The red crosses denote an invariance failure, and the pink diamonds denote the attractor. Before modulation, there are trajectories prematurely exiting the mode; after modulation, all trajectories are bounded inside the mode.

**Unbiased human perturbations** Since external perturbations are an integral part of our task complexity, we recruited six human subjects without prior knowledge of our LTL-DS system to perturb the robot scooping setup. Each subject is given five trials of perturbations. In total, we collected 30 trials as seen in Fig. 12, each of which is seen as an unbiased i.i.d. source of perturbations. On our project page, we show all 30 trials succeed eventually in videos. We did not cherry-pick these results, and the empirical 100% success rate further corroborates our theoretic success guarantee. Interestingly, common perturbation patterns (as seen in the videos) emerge from different participants. Specifically, we see **adversarial perturbations** where humans fight against the robot and **cooperative perturbations** where humans help the robot to achieve the goal of transferring at least one bead from one bowl to the other. In the case of adversarial perturbations, DS reacts and LTL replans. In the case of collaborative perturbations, DS is compliant and allows humans to also guide the motion. In the case where humans are not perturbing yet the robot makes a mistake (e.g. during scooping), LTL replans the scooping DS until the robot enters the transferring mode successfully. The fact that we do not need to hard code different rules to handle invariance failures caused by either perturbations or the robot’s own execution failures in the absence of perturbations highlights the strength of our LTL-powered sensor-based task reactivity.

## I Robot Experiment 2: Inspection Line

To further validate the LTL-DS approach we present a second experimental setup that emulates an inspection line, similar to the one used to validate the LPV-DS approach [19] – which we refer to as the vanilla-DS and use to learn each of the mode motion policies. In [19] this task was presented to validate the potential of the vanilla-DS approach to encode a highly-nonlinear trajectory going from (a) grasping region, (b) passing through inspection entry, (c) follow the inspection line and (d) finalizing at the release station with a single DS. In this experiment we show that, even though it is impressive to encode all of these modes (and transitions) within a single continuous DS, if the sensor state or the LTL task-specification are not considered the vanilla-DS approach will fail to achieve the high-level goal of the task which is, to pass slide the object along the inspection line.





Figure 12: Ending snapshots (100% success rate, see videos for action) of six randomly recruited human subjects performing unbiased perturbations in a total of 30 trials without cherry-picking. Common perturbation patterns (we annotate with the same colored text) emerge from different participants. Specifically, we see *adversarial perturbations* where humans fight against the robot and *cooperative perturbations* where humans help the robot to achieve the goal of transferring at least one bead from one bowl to the other.



To showcase this, in this work we focus only on (b)  $\rightarrow$  (c)  $\rightarrow$  (d) with (a) following a pre-defined motion and grasping policy for experimental completeness.

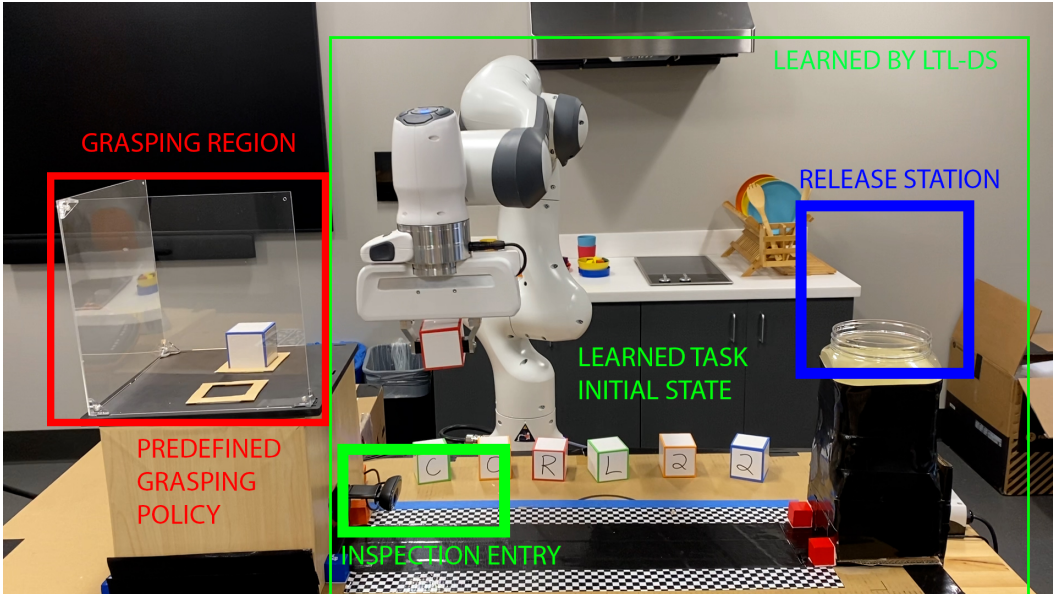


Figure 13: Robot Experiment 2: Inspection Line

**Inspection Task Details** The video of this experiment can be found on our website.

- *Sensor model*: We implement the *sensor model* of the inspection task as an object detector on the inspection track and distances to attractors (defined from AP region-based segmentation described in new Appendix I). As we created a black background for the inspection task and the camera is fixed, with a simple blob detector we can detect if the robot is inside or outside of the inspection line. Hence, the *sensor state* is a binary variable analogous to that of the scooping task.
- *Task specification*: The proposed inspection task can be represented with 2 modes (a) **Go to inspection entry**  $\rightarrow$  (b) **follow inspection line and release**. The AP regions are the bounding boxes around the *inspection entry* and *release station* shown in Fig. 13 which correspond to the attractor regions for each mode. Mode (a) requires the robot to reach the mode attractor and detecting the presence of the cube once it has been reached. Mode (b) requires the cube to slide along the inspection track (reaching the end) and then lift the cube to drop it at the release station.
- *Offline Learning*: We use two demonstrations of the inspection task, together with an LTL specification and run our offline learning algorithm used for the soup-scooping task (without any modifications), as shown in the supplementary video from 0:00-0:18s. Without any adversarial perturbations or environmentally induced failures, the vanilla-DS approach is capable of accomplishing the defined inspection task without invariance failures as shown in 0:19-0:32s.
- *Invariance Failures of Vanilla-DS*: Even though the vanilla-DS approach is now used to learn a less complex trajectory (in terms of trajectory non-linearity), as we excluded the grasping region, we can see that it easily fails to achieve the inspection task when subject to large adversarial perturbations that lead the robot towards an *out-of-distribution* state. This means that the robot was perturbed in such a way that it is now far from the region where the demonstrations were provided. Yet, it is robust to small adversarial perturbations that keep the robot *in-distribution*, as shown in the supplementary video from 0:33-1:18min. The latter is the strength of DS-based motion policies in general and these are the type of perturbations that are showcased in [19]. However, since the DS is only guaranteed to reach the target by solely imposing Lyapunov stability constraints it always reaches it after a large adversarial perturbation, with the caveat of not accomplishing the actual inspection task. Note that this limitation is not specific to the vanilla-DS approach [19], it is a **general limitation** of goal-reaching LfD methods that only care about guaranteeing stability **at the motion-level** be it through Lyapunov or Contraction theory. Hence, by formulating the problem as TLI and introducing sensor states and LTL specification

into the imitation policy we can achieve convergence at the motion-level and task-level.

- **Invariance Guarantee with LTL-DS:** As shown in the supplementary video from 1:19–1:43min we collect a set of invariance failures to construct our mode boundary. Further, from 1:43–2:00min we show the approximated mode boundary from 4 recorded failure states that approximate the vertical boundary and then from 10 recorded failure states which now approximate the horizontal boundary of the mode. The blue trajectories in those videos correspond to rollouts of the vanilla-DS learned from the demonstrations in that mode.

From 2:00–3:40min we show two continuous runs of the inspection task, each performing two inspections. We stress test the learned boundary and LTL-DS approach by performing small and large adversarial perturbations. As shown in the video, when adversarial perturbations are small the DS motion policy is robust and still properly accomplishes the inspection task. When adversarial perturbations are large enough to push the robot outside of the learned boundary, the LTL-DS brings the robot back to the inspection entry mode and tries the inspection line again and again until the inspection task is achieved as defined by the LTL specification - **guaranteeing task completion.**

**Comment on Task Definition:** In order to learn and encode the entire task (from grasp to release) with LTL-DS we would need to include a grasping controller within our imitation policy. It is possible to extend the LTL-DS approach to consider grasping within the imitation policy, yet due to time limitations we focus solely on the parts of the task that can be learned by the current policy – that requires only controlling for the motion of the end-effector. We are concurrently working on developing an approach to learn a grasping policy entirely through imitation, which to the best of our knowledge does not exist within the problem domains we target. In a near future, we plan to integrate these works in order to allow LTL-DS to solve problems that include actuating grippers in such a feedback control framework. Note that, the vanilla-DS approach does not consider the grasping problem either and the experimental setup presented in [19] uses a simple open-loop gripper controller that is triggered when the DS reaches the attractor, such triggering is hand-coded and not learned either in their setup.

## J Robot Experiment 3: Color Tracing

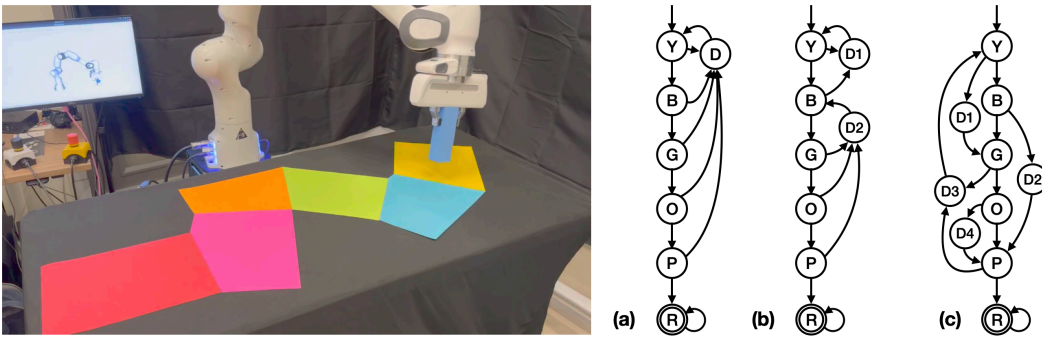


Figure 14: The goal is for the end-effector to move through (while staying within) (Y)ellow, (B)lue, (G)reen, (O)range, (P)ink, and eventually reach (R)ed. If a sensor detects the end-effector is perturbed into the (D)ark region, the system needs to replan according to a given task automaton such as (a), (b), (c). Note D1, D2, D3, D4 refer to different modes (entering the dark region from different colors) that appear visually the same.

This experiment demonstrates LTL-DS’ handling of long-horizon multi-step tasks with non-trivial task structures. Given a single demonstration of kinesthetically teaching the robot end-effector to move through the colored tiles, the system learns a DS for each of the colored mode. The learned DS can then be flexibly recombined according to different LTL-equivalent task automata to react differently given invariance failures. Specifically, we show in the videos on our website three different replanning: (a) mode exit at any colored tile transitions to re-entry at the yellow tile; (b) mode exit at any colored tile after the blue tile transitions to re-entry at the blue tile; and (c) mode exit at the yellow tile transitions to the blue tile, while mode exit at the blue tile transitions to the pink tile.