# 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 original 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 original 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):

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

Therefore, $M(x)f(x)$ is G.A.S. □

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

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

*Proof* Since the task automaton is converted from a 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. □

**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 $f_{\sigma_i}$.*

*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 $f(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ 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, $x^*_{\sigma_i}$ by DS $f_{\sigma_i}$. Placing $x^*_{\sigma_i}$ on guard $G_{ij}$ ensures that $x^*_{\sigma_i} \in \delta_j$, and thus $\forall s \; \sigma_i \Rightarrow \sigma_j$ as $x \to x^*_{\sigma_i}$. □

**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 $f_{\sigma_j}$ has been learned. Since the LTL GR(1) fragment does not permit clauses in the form of (**F G**$\phi$), which states $\phi$ is eventually globally true, the discrete plan will eventually result in $\sigma_j \Rightarrow \sigma_k$ for some $k$, $j \neq k$. Learning $f_{\sigma_j}$ also validates the existence of $x^*_{\sigma_j}$—and, thus, a continuous trajectory toward $G_{jk}$. □

**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. □

## B Motivation for Mode-based Imitation

Our work hopes to achieve generalization in regions of the state space not covered by initial demonstrations. A well-studied line of thought is to collect more expert data [20] so that the policy will now also mimic expert demonstrations in those regions. Our central observation in Fig. 2 is that there exists some thresholds that separate all trajectories deviating from expert demonstrations (black) into successes (blue) and failures (red). The thresholds 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 motion trajectories. In fact, online data collection to correct mistakes made by a policy 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 DS alone does 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 DS to be mode invariant. We prove DS after modulation will still satisfy reachability, and consequently certifying our LfD method will satisfy any given LTL formulas. Figure 8 summarizes how TLI fits in relation to prior work, where gray dashed boxes represent prior work and yellow dashed box highlights our contribution.

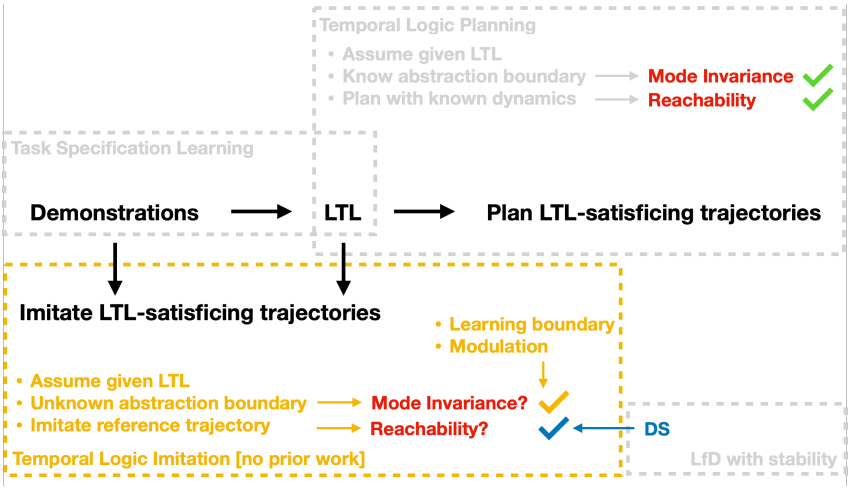


Figure 8: How TLI fits in relation to prior work, where gray dashed boxes represent prior work and yellow dashed box highlights our contribution.

# 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 state-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 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; $f = 1, 2, ...$ 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.)

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

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 the abstraction—environment APs, $r, s, t$, and robot APs, $a, b, c, d$—is defined for the soup-scooping task in Sec. 7.2, 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 corresponding sensor state. $\phi_g^s$ and $\phi_g^e$ set the goal to eventually be in mode $d$ for the system, 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. $\phi_t^s$ 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][1]. 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 $\phi_t^e, \phi_g^e$ 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 $\phi_t^s = \mathbf{G}((a \rightarrow (\mathbf{X}a \vee \mathbf{X}b)) \wedge (b \rightarrow (\mathbf{X}b \vee \mathbf{X}c) \wedge (c \rightarrow (\mathbf{X}c \vee \mathbf{X}d)) \wedge (d \rightarrow \mathbf{X}d))$ 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, $\phi_t^s$ 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 automatons 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

[1]The main benefit is that GR(1) formulas allow fast synthesis of their equivalent automatons [29].

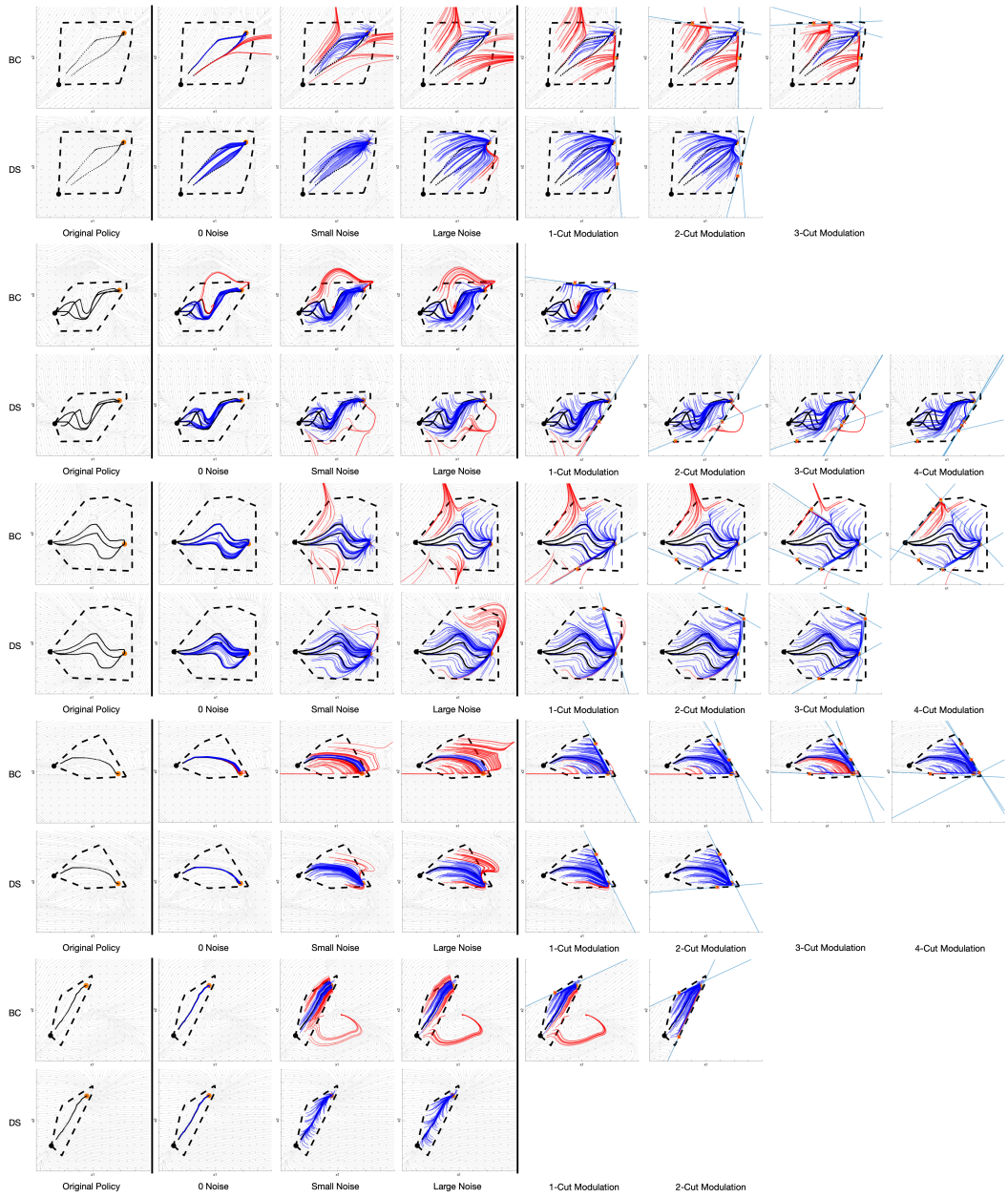


Figure 9: For each convex mode, we use 1-3 demonstrations for learning shown in black. Successful rollouts are shown in blue while unsuccessful rollouts are shown in red. We apply modulation to the large noise case and within four cuts all DS policies are modulated to be mode invariant. While BC policies 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

DS, once learned, could generalize to a new task sharing the same set of modes observed in demonstrations given a new LTL formula. 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.

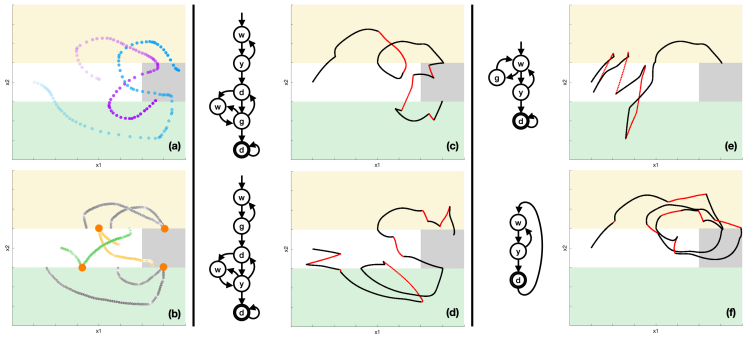


Figure 10: Reuse learned DS for new tasks by editing their task automaton via LTL. **(a)** Two demonstrations (purple and blue) solve a task of adding both chicken (yellow region) and broccoli (green region) to a pot (dark region) in different order. **(b)** Demonstrations are segmented into parts by mode region, which are then shifted to their average attractor locations (orange circles) for DS learning. The learned DS can generalize to new tasks given new automatons specifying **(c)** the order: white (w) ⇒ yellow (y) ⇒ dark (d) ⇒ green (g) ⇒ dark (d), **(d)** the order: white ⇒ green ⇒ dark ⇒ yellow ⇒ 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.

$$
\begin{aligned}
\phi_{cb} = \mathbf{G}((w_1 \rightarrow \mathbf{X}y) \wedge (y \rightarrow (\mathbf{X}w_1 \vee \mathbf{X}d_1)) \\
\wedge (d_1 \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}g)) \wedge (w_2 \rightarrow \mathbf{X}g) \\
\wedge (g \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}d_1 \vee \mathbf{X}d_2)) \wedge (d_2 \rightarrow \mathbf{X}d_2));
\end{aligned}
$$

$$
\begin{aligned}
\phi_{bc} = \mathbf{G}((w_1 \rightarrow \mathbf{X}g) \wedge (g \rightarrow (\mathbf{X}w_1 \vee \mathbf{X}d_1)) \\
\wedge (d_1 \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}y)) \wedge (w_2 \rightarrow \mathbf{X}y) \\
\wedge (y \rightarrow (\mathbf{X}w_2 \vee \mathbf{X}d_1 \vee \mathbf{X}d_2)) \wedge (d_2 \rightarrow \mathbf{X}d_2));
\end{aligned}
$$

$$
\begin{aligned}
\phi_{c} = \mathbf{G}((w \rightarrow (\mathbf{X}y \vee \mathbf{X}g)) \wedge (y \rightarrow (\mathbf{X}w \vee \mathbf{X}d)) \\
\wedge (g \rightarrow \mathbf{X}w) \wedge (d \rightarrow \mathbf{X}d));
\end{aligned}
$$

$$
\phi_{cc} = \mathbf{G}((w \rightarrow \mathbf{X}y) \wedge (y \rightarrow (\mathbf{X}w \vee \mathbf{X}d)) \wedge (d \rightarrow \mathbf{X}w));
$$

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 demon-

stration 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.

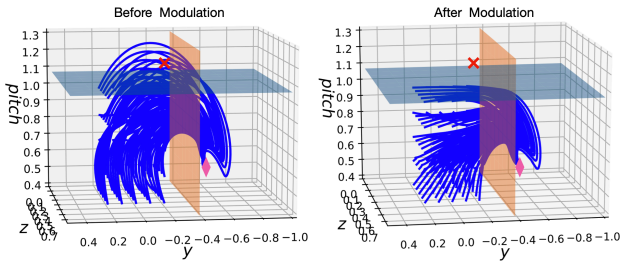


Figure 11: Modulation of sampled robot trajectories. The orange plane represents a guard surface between the transporting mode and the done mode, and the blue plane represents the mode boundary for the transporting mode. The red crosses denote an invariance failure, and the pink diamonds denote the attractor. 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 five 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 25 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 25 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 **collaborative 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. 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.

**Inspection Task Details** The video of this experiment can be found at the bottom of our website and in the attached supplementary files as `ltlds-inspection-corl22.mp4`.

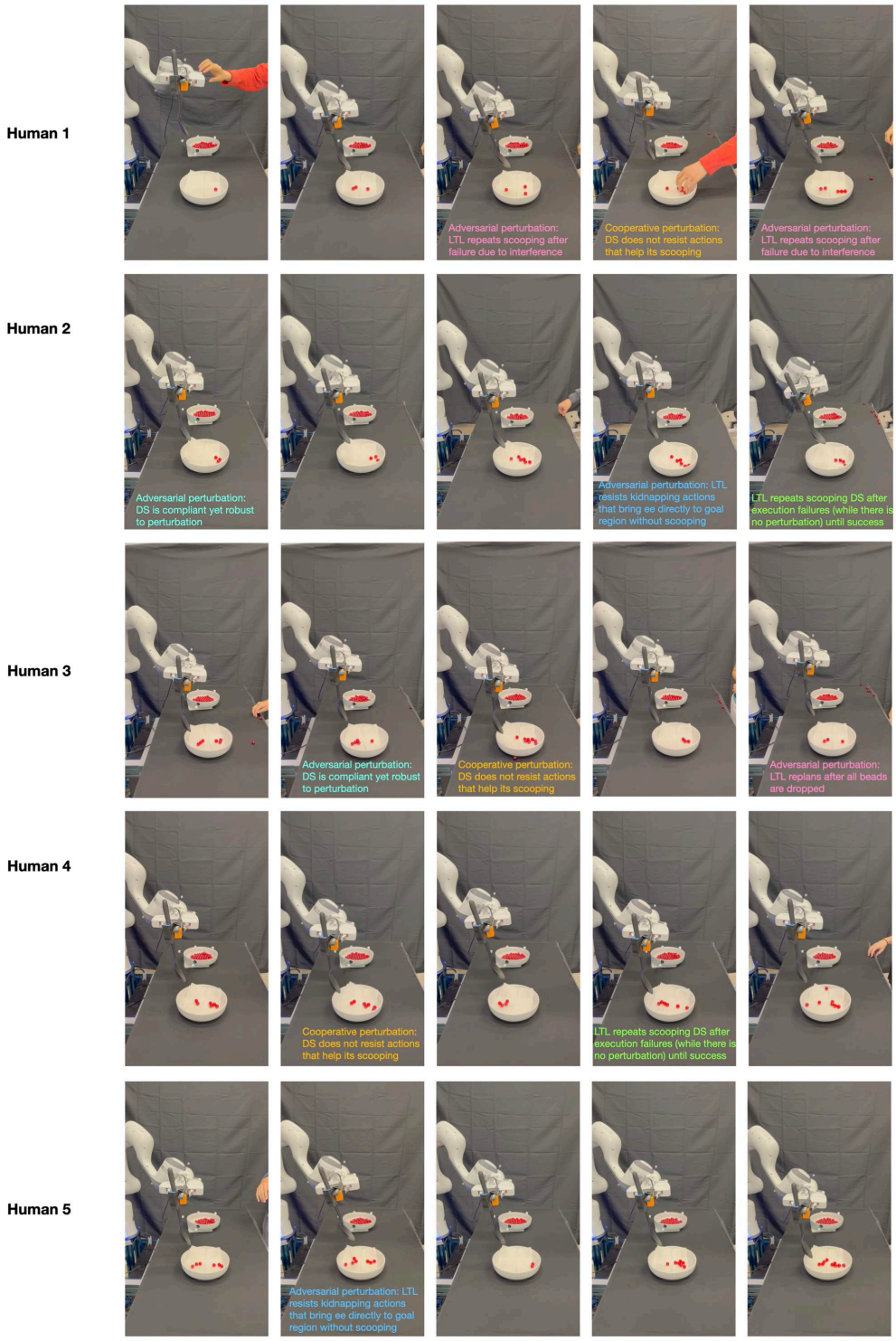


Figure 12: Ending snapshots (100% success rate, see videos for action) of five randomly recruited human subjects performing unbiased perturbations in a total of 25 trials without cherry-picking. Common perturbation patterns (we annotate with the same colored text) emerge from different participants. Specifically, we see *adversarial perturbations* where humans fight against the robot and *collaborative perturbations* where humans help the robot to achieve the goal of transferring at least one bead from one bowl to the other.

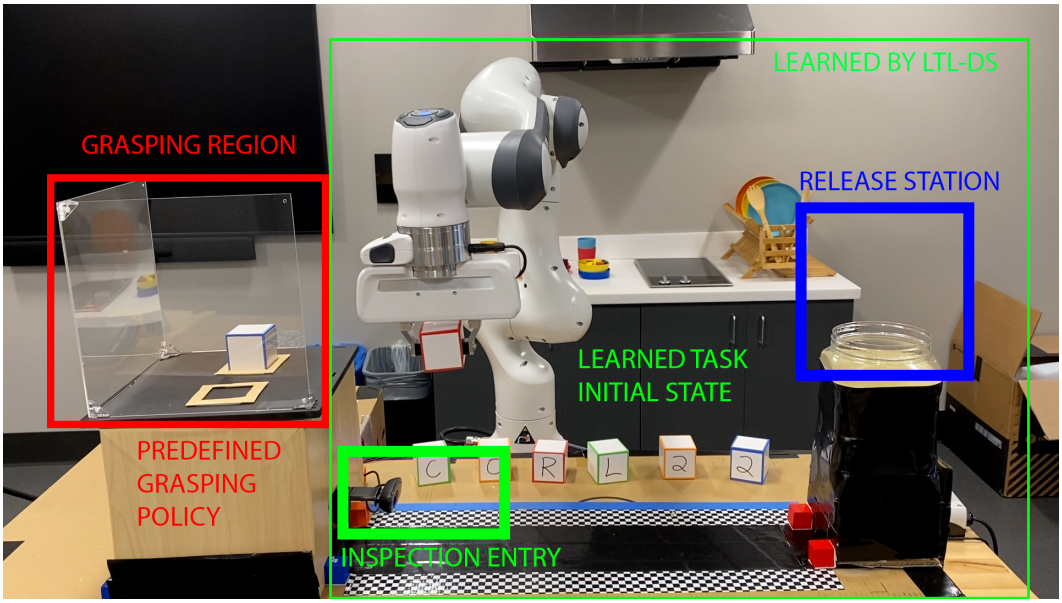


Figure 13: Robot Experiment 2: Inspection Line

- *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 DS-based motion policies 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 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.