SymSkill: Symbol and Skill Co-Invention for Data-Efficient and Real-Time Long-Horizon Manipulation

Yifei Simon Shao, Yuchen Zheng, Sunan Sun, Pratik Chaudhari, Vijay Kumar and Nadia Figueroa
GRASP Laboratory, University of Pennsylvania, Philadelphia, PA, 19104 USA
yishao, zhengyc, sunan, pratikac, kumar, nadiafig@seas.upenn.edu

***Abstract*— Multi-step manipulation in dynamic environments remains challenging. Two major families of methods fail in distinct ways: (i) imitation learning (IL) is reactive but lacks *compositional generalization*, as monolithic policies do not decide which skill to reuse when scenes change; (ii) classical task-and-motion planning (TAMP) offers compositionality but has prohibitive *planning latency*, preventing real-time failure recovery. We introduce `SymSkill`, a unified learning framework that combines the benefits of IL and TAMP, allowing compositional generalization and failure recovery in real-time. Offline, `SymSkill` jointly learns predicates, operators, and skills directly from unlabeled and unsegmented demonstrations. At execution time, upon specifying a conjunction of one or more learned predicates, `SymSkill` uses a symbolic planner to compose and reorder learned skills to achieve the symbolic goals, while performing recovery at both the motion and symbolic levels in real time. Coupled with a compliant controller, `SymSkill` enables safe and uninterrupted execution under human and environmental disturbances. In RoboCasa simulation, `SymSkill` can execute 12 single-step tasks with 85% success rate. Without additional data, it composes these skills into multi-step plans requiring up to 6 skill recompositions, recovering robustly from execution failures. On a real Franka robot, we demonstrate `SymSkill`, learning from 5 minutes of unsegmented and unlabeled play data, is capable of performing multiple tasks simply by goal specifications. The source code and additional analysis can be found on https://sites.google.com/view/symskill.**

I. Introduction

Enabling robots to perform complex, long-horizon manipulation in the real world remains challenging. Recent imitation-learning (IL) approaches [1], [2] excel at reproducing skills given large, high-quality datasets, but tend to learn monolithic policies rather than reusable skills and predicates that compose into multi-step plans. Historically, Task and Motion Planning (TAMP) bridges this gap by decomposing problems into symbolic planning over predicates/operators and continuous motion generation [3]. However, two factors limit TAMP scalability in practice. 1) Symbols and skills are often hand-engineered and tuned per environment, which is labor-intensive. 2) TAMP takes tens to hundreds of seconds to solve a large problem in a realistic contact-rich simulation environments [4], making it infeasible to plan in dynamic environments with moving objects, or achieve real-time failure recovery at the symbolic or motion level.

Symbol and Skill Co-Invention methods, such as [5], combine the benefits of IL and TAMP by learning reusable symbols and skills from robot demonstrations and planning symbolically to decide which skill to execute at runtime. As shown in [5], [6], there is a delicate trade-off between inventing long-horizon operators that are too general for useful planning at inference time and inventing operators that are too granular, risking skills learned from insufficient data performing poorly. As a result, both works above use a *propose and down-select* hill-climbing optimization for selecting predicates. However, even when predicates are invented in relative frames, the learning process can take minutes to hours as the number of objects and demonstrations increases, and may still fail to discover semantically meaningful predicates. To address the aforementioned challenge, we take a different approach by sidestepping expensive optimization altogether. Our key insight is that interactions with objects follow only a handful of common patterns: 1) robots typically approach each object in a limited set of ways, and 2) moving object come to rest in one of a few meaningful poses relative to a stationary object. Inspired by recent works that use Vision-Language Models (VLMs) to identify task-relevant objects, we employ a VLM in a lightweight role: identifying the relevant stationary object in each demonstration. This allows us to transform trajectories into the stationary object's frame for predicate and skill learning, without relying on

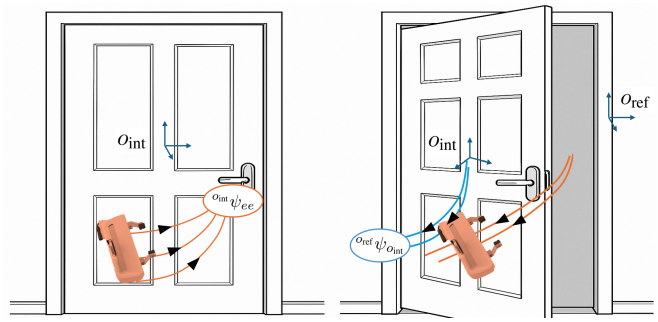

Fig. 1: Illustration of the `SymSkill` predicate and skill co-invention process on a DoorOpen task. **Left:** In the `premotion` segment (end-effector only motion), the object in motion in the next segment is treated as the object of interest o_{int}, and its frame serves as the reference for both predicate and skill learning. End-effector trajectories in this frame are used to fit SE(3) LPV-DS skills, and their endpoints are clustered to yield object-gripper relative pose predicates ${}^{o_{\text{int}}}\psi_{ee}$. **Right:** In the `motion` segment (gripper + object moving), a reference object o_{ref} is selected by querying a VLM on frames from the segment. Gripper trajectories are then expressed in the o_{ref} frame and used to fit a DS skill. Endpoints of the manipulated object trajectory in the o_{ref} frame are clustered to yield object–object relative pose predicates ${}^{o_{\text{ref}}}\psi_{o_{\text{int}}}$.

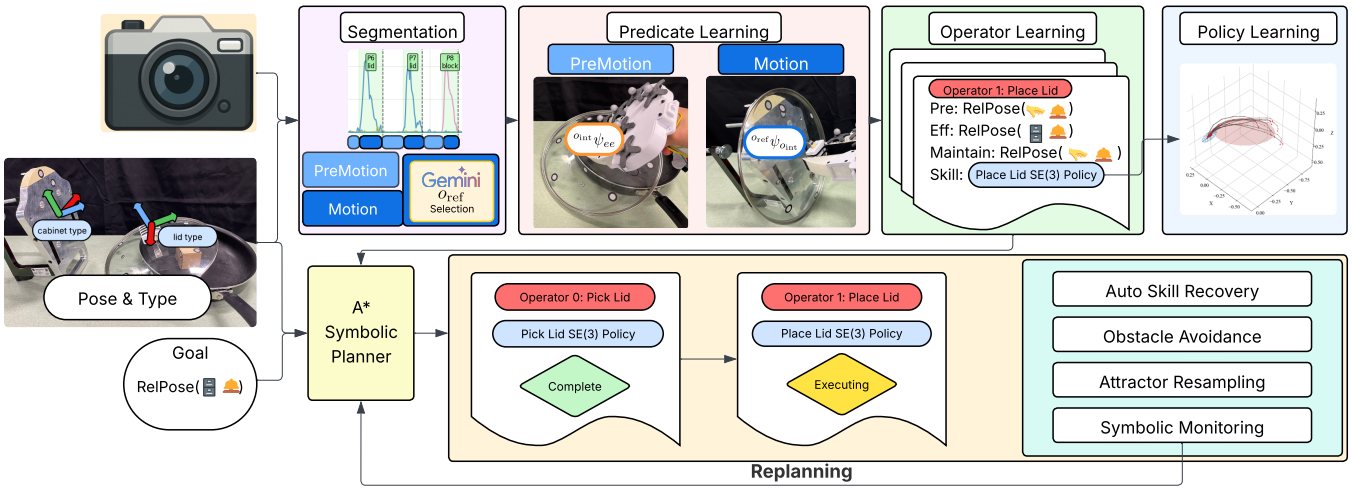

Fig. 2: `SymSkill` offline pipeline (top half) and the online pipeline (bottom half). Subsection V-A (purple) describes segmentation and reference frame selection. Subsection V-B (orange) describes how predicates are learned for each segment. Subsection V-C (green) learns the operators for online planning. Subsection V-D (blue) describes how each operator's skill is learned. Subsection V-E (yellow) describes how `SymSkill` operates online.

VLMs for policy generation or reasoning online.

To this end, we propose `SymSkill`, a unified framework that learns *predicates*, *operators* and *goal-oriented skills* in an unsupervised manner with unsegmented robot demonstrations data — requiring as few as 5 demonstrations per task. At the symbolic level, `SymSkill` identifies which object each trajectory segment moves toward using VLM and automatically defines predicates as relative pose classifiers. At the motion level, we adopt a dynamical system (DS)-based approach to learn stable motion policies from minimal demonstration data in near real-time. At execution time, given a symbolic goal, specified with the learned predicates, `SymSkill` uses a symbolic planner to compose skills into long-sequence plans that generalize across number of objects. Due to the fast planning speed, `SymSkill` supports real-time error recovery at both the symbolic and skill levels. Coupled with a compliant passive DS controller, `SymSkill` ensures the execution is always stable, safe, and uninterrupted by replanning. In RoboCasa simulation, `SymSkill` learns 24 reusable skills from 12 short-horizon tasks and achieves a 85% success rate. Without additional data, it composes these skills to perform multi-step composite tasks with success. We also validate the approach on real-world robots, performing tasks by learning from 5 minutes of play data.

Contributions 1) a framework for joint discovery and learning of symbols and goal-oriented DS skills from unlabeled and unsegmented demonstrations of short and long-horizon tasks, 2) online execution and failure recovery with reactive planning at the task and motion level, and 3) an open-source implementation for out-of-the-box robot-learning in RoboCasa [7] with original demonstrations.

II. Problem Statement

We consider the problem of learning from play in *deterministic*, *fully observed* manipulation domains. Let $\mathcal{O}$ be the set of objects, where each object $o \in \mathcal{O}$ is assigned to a type $\lambda(o)$ drawn from a predefined finite set Λ. Let $\mathcal{F} = \{ee\} \cup \{o : o \in \mathcal{O}\}$ denote the set of kinematic frames of end-effector and object frames.

A pose $\mathbf{T} \in SE(3)$ comprises position and orientation; ${}^{A}\mathbf{T}_{B} \in SE(3)$ denotes the pose of frame B expressed in frame A. At time t, the continuous state in world frame is

$$\mathbf{x}_t \;=\; \Big(\mathbf{T}_{ee},\, \{\mathbf{T}_{(o)}\}_{o\in\mathcal{O}},\, \{\lambda(o)\}_{o\in\mathcal{O}}\Big).$$

Consistent with related works that also assume access to complete object states in simulation or via fiducial-based perception systems [5], [6], we assume a perception module that provides $\mathbf{x}$ of all objects at each timestep.

Problem Setup. We are given N unlabeled and *unsegmented* robot demonstrations,

$$\mathcal{D} = \{\tau_i\}_{i=1}^{N}, \qquad \tau_i = \{\mathbf{x}_t\}_{t=0}^{T_i}.$$

Each τ_i of length T_i contains one or more demonstration trajectories of arbitrary object manipulating in the scene. For each trajectory, we record a time-synchronized RGB video of the workspace that keeps all task-relevant objects in view. A policy, outputting on the full end-effector pose trajectory, has the form of

$$\big[v, \omega\big]^T = f(\cdot), \tag{1}$$

where v, w are linear and angular velocity action of the end effector respectively. We further assume gripper action $g = \{\text{open}, \text{closed}\}$ to be either open or closed throughout the policy. At test time, given initial state $\mathbf{x}_0$ and goal state $\mathbf{x}_G$, we seek to apply sequentially a number of policy tuple $\langle f, g\rangle$ so that the $\mathbf{x}_G$ is achieved, while monitoring and recovering from failure in real-time. The robot action defined by policy f is tracked by the following passive impedance controller

$$F_{ee} = G - D(\dot{\mathbf{T}}_{ee} - f(\cdot)), \tag{2}$$

where $G \in \mathbb{R}^6$ is the gravity compensation term, $\dot{\mathbf{T}}_{ee} \in \mathbb{R}^6$ is the end-effector velocity and D is the damping gain ensuring the control input is energy dissipating in the directions orthogonal to the desired velocities, as in [8].

III. Preliminary

A. Learning Stable SE(3) Policy in Relative Frame

When learning skills, we use dynamical system-based motion policy [9]–[11]. By leveraging redundancy of solutions from demonstration data, a learned dynamical system (DS) can be used as a stable motion policy that is robust to both temporal and spatial uncertainty. Specifically, we implement SE(3) LPV-DS [12] combined with convex policy learning [13], which requires only a small amount of data to generate policies in near real-time. The framework consists of a linear Parameter Varying DS (LPV-DS), f_p, for position control and a Quaternion-DS, f_o, for orientation control:

$$v = f_p(x;\ \Theta_p), \quad \omega = f_o(\mathbf{q};\ \Theta_o), \tag{3}$$

where the inputs are position $x \in \mathbb{R}^3$ and orientation $\mathbf{q} \in SO(3)$ represented as quaternions, and each function is parameterized by Θ_*. Using the LPV-DS as an example, the function f_p has the form of a mixture of continuous linear time-invariant (LTI) system:

$$v = \sum_{k=1}^{K} \gamma_k(x) \mathbf{A}_k \left(x - x^*\right), \tag{4}$$

where K represents the total number of LTI systems and $\gamma_k(x)$ is the mixing function that assigns the weight of each LTI system. $\gamma_k(x)$ is characterized by the Gaussian Mixture Model (GMM) parameters $\{\pi_k, \mu_k, \mathbf{\Sigma}_k\}_{k=1}^K$, which are estimated by fitting a GMM to the reference trajectories. Subsequently, each LTI system $\mathbf{A}_k$ is learned by solving a semi-definite program (SDP) with constraints enforcing globally asymptotic stability. For more details on SE(3) LPV-DS, please refer to [10]–[12].

B. Symbolic Abstraction and Task and Motion Planning

A *predicate*, in this work, is a function, $\psi(A, B)$, that takes a tuple of frames as input and maps to a truth value as,

$$\psi_{\lambda_1,\lambda_2}(A, B) \rightarrow \{\text{True}, \text{False}\} \tag{5}$$

such that $\lambda(A) = \lambda_1$ and $\lambda(B) = \lambda_2$. Instantiating all predicates over all type-consistent tuples in $\mathbf{x}_t$ yields the symbolic state s_t (the set of true ground atoms).

An *operator* $\alpha = \langle \text{params}, \text{pre}, \text{eff}, \text{maintain}, \textit{skill} \rangle$ is a typed template defined over objects/frames. It consists of: (i) **parameters** params $= [\lambda_1, \lambda_2, \dots]$ specifying the required types of objects/frames, (ii) **preconditions** pre(α), the set of predicates that must hold in the symbolic state before the operator can be executed, (iii) **effects** eff(α), consisting of *add effects* (predicates made true) and *delete effects* (predicates made false) after execution, and (iv) **maintenance conditions** maintain(α), the set of predicates that must hold throughout execution. (v) ***skill*** a low-level policy tuple $\langle f, g \rangle$, such as the DS policy for $f(\cdot)$ (Sec. III-A) and grasping action g, that realizes this transition on the robot.

Formally, an operator defines a transition from an initial abstract state s_0 to a successor state s_1,

$$\alpha([o_1, o_2, \dots], s_0) \ \rightarrow \ s_1, \tag{6}$$

where all parameters are grounded by assigning objects to types: $[\lambda(o_1) = \lambda_1,\ \lambda(o_2) = \lambda_2, \dots]$. If the grounded preconditions are satisfied in s_0, the operator can be applied, yielding s_1 through its effects while enforcing the maintenance conditions. The typical planning process is a slower than real-time search and optimization process, with methods like interleaved planning [14] or Search & Sample (SeSame) [15].

IV. Related Work

We categorize related work below, and compare the most relevant works to ours in Table I. Note that all methods in the table learn predicates in relative frames, which has proven a necessity for generalizable manipulation learning frameworks. Of all the methods, ours is the only one that plans in real time and requires fewer than 10 demonstrations.

Data Generation for Visuomotor Policies: Related to our approach are recent works that leverage relative frames for data generation [17]–[19]. These methods typically segment human demonstrations into sub-trajectories and then *stitch* them, either through simulation or direct perception editing, to augment data and train visuomotor policies from moderately sized datasets. While effective for scaling data, they do not learn the underlying *task dependencies* from demonstrations, but instead reproduce rigid subtask sequences.

Hierarchical Imitation Learning: Current imitation learning (IL) strategies such as Diffusion Policy [1] excels at reproducing complex multi-modal skills, but they often degenerate on long-horizon tasks that require sequencing multiple skills. To address this, hierarchical IL methods [20], [21] decompose demonstrations into a high-level planner over skills and low-level controllers for execution. While this structure improves tractability and performance, the high-level planners provide no symbolic guarantees that the composed sequence of skills will achieve goal completion. Instead, their plans are statistical predictions from latent distributions, lacking logical verification or explicit reasoning over task dependencies.

Symbol Learning with Skill Label: One thread of work invents symbols with pre-defined skills and skill-labeled data [22]–[24]. More recently, [25] proposed to learn the neural effect predicates of operators and classifiers for these predicates together. [16] (NOD-TAMP in Table I) uses NDF features [26] for learning grasping predicates. However, labeling is tedious, requiring teleoperating with pre-programmed skills, or performing direct operational-space teleoperation followed by skill labeling.

Symbol Learning with Unsegmented Data: This class of methods propose a candidate pool of predicates using enumeration or VLM, and then sub-select using an objective function [6], [27]. When learning from a limited number of demonstrations or when the number of features for each object is high, these approaches often fail due to limited data or extended running time, as shown in Results. Notably, [4] (LAMP in Table I) proposes Relational Critical Regions (RCR) as predicates without performing the optimization.

TABLE I: Comparison of predicate and skill learning methods.

Approach	Predicates	Skills	# of Demos	Planning Time
SymSkill (Ours)	Relative Pose Cluster (Start/End Motion)	SE(3) LPV-DS [12]	1-10	<100ms
NSIL [5]	Relative Pose Cluster (Low Relative Velocity)	MLP BC	200	<100ms
LAMP [4]	Relational Critical Regions	Motion Planning (MP)	200	> 50 s
NOD-TAMP [16]	NDF Features	Optimization + MP	1-10	> 50 s

However, it still opts to use motion planning as the skill, making real-time failure recovery difficult.

Predicates/Operator/Skill Co-invention: Only one other work performs co-invention similar to us. [5] (NSIL in Table I) uses relative low-velocity regions of the trajectory as meaningful candidate predicates. However, as shown in our experiment, this method fails to produce correct and semantically meaningful predicates and still requires the error-prone down-select optimization process mentioned above.

V. Methods

`SymSkill` jointly learns predicates ψ, operators α, and skills $\mathbf{\Theta}$ from unsegmented demonstrations $\mathcal{D}$ and leverages them for real-time task execution. Demonstrations are segmented into end-effector–only (`premotion`) and end-effector–object (`motion`) segments, expressed in relative frames (Sec.V-A). From these segments, we cluster endpoints to invent relative-pose predicates (Sec.V-B). Then the operators are derived by tracking predicate transitions (Sec.V-C). Lastly, DS policies skill for each operator is learned (Sec.V-D). At test time, symbolic goals are achieved by composing operators into skill sequences. Closed-loop DS policy ensures stability and disturbance rejection, while online monitoring and replanning enable real-time recovery. Fig. 2 shows the offline and online pipeline of `SymSkill`.

A. Demo Segmentation and Reference-Frame Selection

We assume a demonstration $\tau = \{\mathbf{x}_t\}_{t=0}^{T}$ comprises of unordered episodes of skills, each with `premotion` $\rightarrow$ `motion` segments. A `premotion` segment is the motion of the end-effector gripper towards an object prior to making contact, while during a `motion` segment we assume at most one non-gripper object moves concurrently with the gripper. This holds in typical single-arm demonstrations for both rigid-object transport and single-joint articulated-object interactions.

For each demonstration τ, we compute linear and angular velocities for all frames o and detect change points using a fixed threshold on either velocities. For gripper end-effector ee and object $o \in \mathcal{O}$, let t^{start} and t^{stop} denote the times at which some o begins and ceases motion. We call this object the *motion object* o_{int} for that episode. We then extract two contiguous segments:

$$\underbrace{\mathcal{S}^{\text{pre}}_{o_{\text{int}}} = [t_0,\, t^{\text{start}})}_{\text{gripper-only motion (premotion)}} \quad \text{and} \quad \underbrace{\mathcal{S}^{\text{mot}}_{o_{\text{int}}} = [t^{\text{start}},\, t^{\text{stop}}]}_{\text{gripper+object motion (motion)}} .$$

Here t_0 is the maximal time before t^{start} such that no object other than ee is moving in $[t_0, t^{\text{start}})$.

For `premotion` segments, we express trajectories in the frame of the motion object and treat the frame of o_{int} as the reference frame:

$$\texttt{premotion:} \quad \left\{ {}^{o_{\text{int}}}\mathbf{T}_{ee}(t) \right\}_{t \in \mathcal{S}^{\text{pre}}_{o_{\text{int}}}}.$$

For `motion` segments, both ee and o_{int} are in motion in the world frame. We do not assume rigid contact between them, since manipulating articulated items often involves non-prehensile movements. We assume o_{int} motion is typically organized around one or a few *reference objects*, each denoted as o_{ref} (e.g., transporting a cup into a sink, rotating a door w.r.t. its cabinet). To obtain all reference objects for `motion` segments individually for each motion episode, while capturing semantically meaningful reference objects, we query the Gemini-2.5-Pro [28] VLM on n evenly spaced frames from $\mathcal{S}^{\text{mot}}_{o_{\text{int}}}$ with a *structured* output constrained to scene objects, as in Fig. 3. This structured output limits hallucination and enforces selection among known candidates. With all o_{ref} fixed, we retain motion-segment trajectories in that frame:

$$\texttt{motion:} \quad \left\{ {}^{o_{\text{int}}}\mathbf{T}_{ee}(t),\ {}^{o_{\text{ref}}}\mathbf{T}_{ee}(t),\ {}^{o_{\text{ref}}}\mathbf{T}_{o_{\text{int}}}(t) \right\}_{t \in \mathcal{S}^{\text{mot}}_{o_{\text{int}}}}.$$

We assume that objects of the same type can be manipulated in a similar manner, and that interactions between the same object type and reference type share common trajectory structures that can be exploited during learning. For now, we assume each object has a predefined type and $\lambda(o_{\text{int}}) \in \Lambda, \lambda(o_{\text{ref}}) \in \Lambda$, but we can also easily expand to a open-object setting using a VLM for classification, as in [27].

Outputs: aggregating across demonstrations produces:

$$\mathcal{D}_{\text{pre}}(\lambda_{o_{\text{int}}}) = \left\{ \left({}^{o_{\text{int}}}\mathbf{T}_{ee}(t) \right)_{t \in \mathcal{S}^{\text{pre}}_{o_{\text{int}}}} \;\middle|\; \lambda(o_{\text{int}}) = \lambda_{o_{\text{int}}} \right\} \tag{7}$$

$$\mathcal{D}_{\text{motion}}(\lambda_{o_{\text{int}}}, \lambda_{o_{\text{ref}}}) = \left\{ \left({}^{o_{\text{int}}}\mathbf{T}_{ee}(t), {}^{o_{\text{ref}}}\mathbf{T}_{ee}(t), {}^{o_{\text{ref}}}\mathbf{T}_{o_{\text{int}}}(t) \right)_{t \in \mathcal{S}^{\text{mot}}_{o_{\text{int}}}} \;\middle|\; \lambda(o_{\text{int}}) = \lambda_{o_{\text{int}}} \cap \lambda(o_{\text{ref}}) = \lambda_{o_{\text{ref}}} \right\}. \tag{8}$$

B. Relative Pose Predicate Learning

We seek to capture distributions of relative poses that serve as meaningful symbolic predicates. We consider the relative pose of the end-effector with respect to the motion object, ${}^{o_{\text{int}}}\mathbf{T}_{ee}$, aggregated across $\mathcal{D}_{\text{pre}}(\lambda_{o_{\text{int}}})$. Rather than taking the last frame of each trajectory, which is unreliable under small datasets or non-prehensile motions, we fit normal distributions over the collection of poses observed in `motion` segments $\{{}^{o_{\text{int}}}\mathbf{T}_{ee}(t)\}_{t \in \mathcal{S}^{\text{mot}}_{o_{\text{int}}}}$. These are two independent Gaussians over translation ${}^{o_{\text{int}}}p_{ee} \sim \mathcal{N}(\mu^{o_{\text{int}},ee}_{\text{pos}}, \Sigma^{o_{\text{int}},ee}_{\text{pos}})$ and orientation $\log({}^{o_{\text{int}}}R_{ee}) \sim \mathcal{N}(\mu^{o_{\text{int}},ee}_{\text{ori}}, \Sigma^{o_{\text{int}},ee}_{\text{ori}})$. Given a new

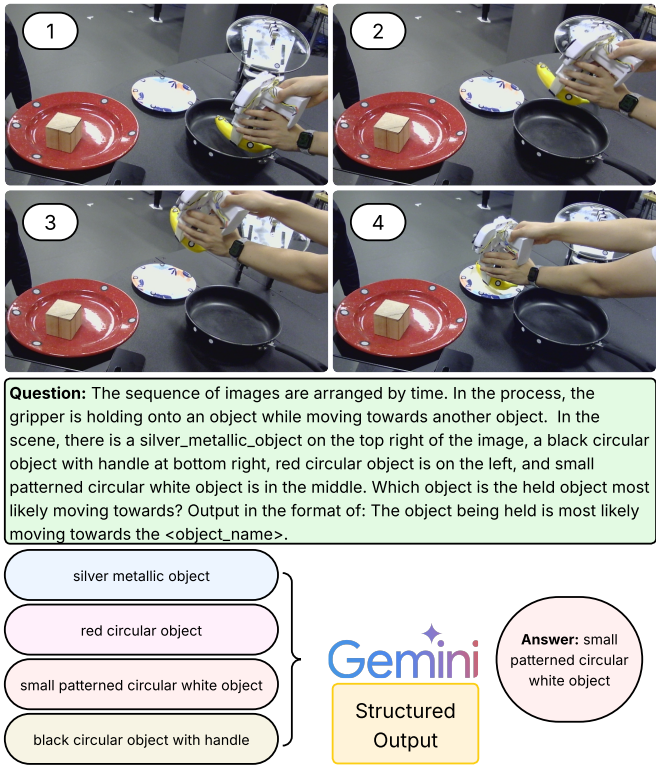

Fig. 3: The VLM prompt used for the real-world learning-from-play experiment proceeds as follows. First, the initial image is used to obtain text descriptions of all objects in view. Next, four equally spaced images from each `motion` segment are provided to Gemini together with the required output enumeration object, using the structured output feature. The returned text is then mapped back to the corresponding object name.

relative pose, we compute Mahalanobis distances to the respective means: $d_{\text{pos}}({}^{o_{\text{int}}}p_{ee})$, $d_{\text{ori}}(\log({}^{o_{\text{int}}}R_{ee}))$. We declare the predicate ${}^{o_{\text{int}}}\psi_{ee}$ to hold if both distances $\epsilon_{\text{pos}}, \epsilon_{\text{ori}}$:

$$ {}^{o_{\text{int}}}\psi_{ee}(\mathbf{x}) = \mathbf{1}[d_{\text{pos}} \leq \epsilon_{\text{pos}} \wedge d_{\text{ori}} \leq \epsilon_{\text{ori}}]\,. $$

fall below thresholds. Similarly, object–object relative pose predicates ${}^{o_{\text{ref}}}\psi_{o_{\text{int}}}$ are obtained by fitting Gaussian distributions over $\{{}^{o_{\text{ref}}}\mathbf{T}_{o_{\text{int}}}(t)\}_{t\in\mathcal{S}^{\text{mot}}_{o_{\text{int}}}}$, augmented with a short (≈2s) post-motion window to stabilize end-pose estimation. The resulting ellipsoids not only define predicates but also serve as samplers for downstream goal-pose resampling during online recovery (Sec. V-E).

Outputs: Collecting these components yields the *predicate libraries*

$$ \Psi_{\text{pre}}(\lambda_{o_{\text{int}}}) = \{{}^{o_{\text{int}}}\psi_{ee}\}, \qquad \Psi_{\text{motion}}(\lambda_{o_{\text{int}}}, \lambda_{o_{\text{ref}}}) = \{{}^{o_{\text{ref}}}\psi_{o_{\text{int}}}\}. $$

C. Operator Learning using Learned Predicates

After we learn the relative-pose predicates, we re-evaluate all demonstration trajectories with $\Psi_{\text{pre}}(\lambda_{o_{\text{int}}})$ and $\Psi_{\text{motion}}(\lambda_{o_{\text{int}}}, \lambda_{o_{\text{ref}}})$ and invent symbolic operators using the method of [29]: We first convert each demonstration into an abstract state sequence by evaluating all learned predicates at every *demonstration segmentation boundary*. We denote the abstract states immediately before and after a transition as s_0 and s_1, respectively. Across these sequences, we identify recurring transition groups by finding segments

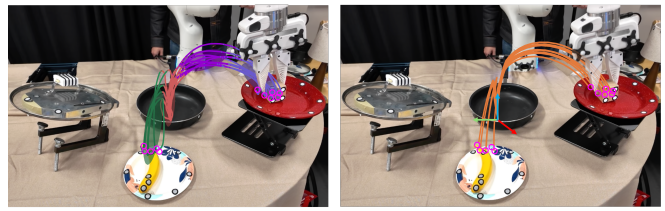

Fig. 4: The visualization of demonstrations and SE(3) LPV-DS policy rollout for Op3 in Tab.III. The left figure shows multiple collected trajectories placing a thing type item from various locations into the pan. The multimodal nature of the data is captured by 4 distinct Gaussians shown in different colors following the policy learning outlined in Sec. III-A. The right figure shows the reconstruction results of the learned policy starting from the same initial conditions, where the policy pose attractor in the pan frame is marked as an axis. All demonstrations converge on the attractor.

with the same *effects*, where effects are defined as

$$ \text{add} = \bigcap(s_1 \setminus s_0), \ \text{del} = \bigcap(s_0 \setminus s_1), \ \text{eff} = \{\text{add}, \text{del}\}. $$

The precondition is then obtained as the intersection of all preceding states,

$$ \text{pre} = \bigcap s_0. $$

Because our system must monitor continuous states $\mathbf{x}$ online, we augment each operator with a set of *maintain conditions* to be the intersection of all continuous-state predicates that hold throughout the interval between s_0 and s_1,

$$ \text{maintain} = \bigcap_{t(s_0)\leq t<t(s_1)} \mathbf{x}(t). \tag{9} $$

Together, we obtain a new operator

$$ \alpha = \langle \text{params}, \text{pre}, \text{eff}, \text{maintain}, \textit{skill} \rangle, $$

where params(α) are ordered and typed inputs that are automatically aggregated from all elements above. Tab. III shows the operators learned for the real-world learning-from-play experiment.

Outputs: We call the collection of operators Ω, where each operator α has trajectory segments from the dataset. Each operator's *skill* will be learned in the next subsection.

D. SE(3) Skill Learning

Each operator $\alpha \in \Omega$ requires a *skill* = $\langle f, g \rangle$ for controlling the pose and gripper action of end-effector. We parameterize the policy f_α as a concatenated function of Eq. (3). For operators that model the `premotion` segments, we follow the learning procedures outlined in Sec. III-A, and use the demonstration data $\{{}^{o_{\text{int}}}\mathbf{T}_{ee}\}$ from Eq. (7) to obtain the corresponding policy:

$$ {}^{o_{\text{int}}}f_\alpha({}^{o_{\text{int}}}\mathbf{T}_{ee}; \Theta_p, \Theta_o). \tag{10} $$

For operators consisting of `motion` trajectories, the policies are expressed in o_{ref} frame following the same learning procedure using the demonstration data $\{{}^{o_{\text{ref}}}\mathbf{T}_{ee}\}$ from Eq. (8):

$$ {}^{o_{\text{ref}}}f_\alpha({}^{o_{\text{ref}}}\mathbf{T}_{ee}; \Theta_p, \Theta_o). \tag{11} $$

For `motion` segments, specifically the ones including non-prehensile motions, we find that policies using relative pose trajectories between ee, o_{ref} perform significantly better than

using relative pose trajectories between $o_{\text{int}}, o_{\text{ref}}$, hence justifying the use of $\{{}^{o_{\text{ref}}}\mathbf{T}_{ee}\}$ in Eq. (11). As introduced in Sec. II, the output of each learned policy is tracked by a task-space passive controller [8] as in Eq. (2). One visualized policy is shown in Fig. 4.

E. Online Execution Monitoring and Adaptation

The online algorithm requires a symbolic goal state s_g[1], expressed as a conjunction of one or more learned predicates. Given the current continuous state $\mathbf{x}_0$, we first compute its symbolic abstraction s_0. We then perform symbolic planning using A* search with the learned operators, producing a plan skeleton $\alpha_1, \alpha_2, \ldots, \alpha_n$ from s_0 to s_g, if one exists. We then sequentially execute *skill* in α, requiring little computation during execution. Since each *skill* is a stable feedback policy, when f_α outputs zero velocity, we advance to the next operator.

During execution, we monitor i) that the maintain conditions hold and ii) new expected effect satisfaction when each *skill* ends. If failure occurs we replan from the current state. See project website for online algorithm. We summarize the elements that enable reliable recovery and eventual plan completion.

Obstacle Avoidance For each object in the scene $\mathcal{O}_{-o_{\text{int}}}$, excluding the ones that the gripper is holding or approaching, we model them as an ellipsoid and apply the local modulation introduced in [9] during skill execution:

$$f' = \mathbf{M}(\mathcal{O}_{-o_{\text{int}}}) f(\mathbf{T}; \mathbf{\Theta}), \tag{12}$$

where the modulated policy f' incorporates the obstacle avoidance behavior and the modulation matrix $\mathbf{M}$ is constructed through eigenvalue decomposition with the normal and tangent directions of the defined ellipsoid boundaries.

Resampling after failure If a robot fails to execute a task on a given object, attempting it again without replanning will typically lead to another failure. Inspired by TAMP, a policy f can be modified online by performing a frame transform when detecting failure during skill execution. Formally, we directly transform the policy: $f' = \mathbf{T} f$, where $\mathbf{T}$ is the pose sampled from the effect normal distribution as introduced in Sec. V-B. 1) When the maintain effect is lost, such as losing the grasp of an object, we assume the *previous* skill needs to be resampled; 2) When effects of current α is not satisfied at the end of *skill*, we assume the attractor of the *current* skill needs to be resampled. Therefore, depending on the operator sequence, we draw samples from ${}^{o_{\text{int}}}\psi_{ee}$ or ${}^{o_{\text{ref}}}\psi_{o_{\text{int}}}$ to apply transformation. This strategy enables autonomous recovery from external disturbances, such as the robot regrasping a dropped object or reopening a closed cabinet door.

VI. Experimental Results

We evaluate our method in `RoboCasa` [7] simulation environment, and on the real Franka robot with motion capture and a webcam during learning.

[1] s_g is either directly specified or can be specified by symbolic abstraction of a goal state $\mathbf{x}_g$.

TABLE II: RoboCasa simulation result on 10 trials per task

Task Success Rate %	Proposed	Proposed w/o Monitoring	Proposed w/ DP
OpenSingleDoor	**100**	100	0
CloseSingleDoor	**100**	80	0
PnPCounterToCab	**80**	70	0
PnPCabToCounter	**100**	40	0
PnPStoveToCounter	**70**	30	0
PnPCounterToStove	**20**	0	0
OpenDrawer	**100**	100	0
CloseDrawer	**70**	50	40
TurnOnStove	**100**	100	0
TurnOffStove	**80**	30	0
TurnOnSinkFaucet	**100**	100	0
TurnOffSinkFaucet	**100**	90	0
Average	**85.0**	**65.0**	**3.3**

A. Single Step Simulation Result

We exclusively use the demonstrations collected by the authors of the `RoboCasa` paper to ensure reproducibility. For single-step tasks, we reduce the variation in the demonstrations by filtering to keep only one variant of fixture per task, such as those `OpenSingleDoor` demonstrations with a cabinet that opens to the left. At test time, we also only generate environment with reduced task variation. Each task still have some randomness such as object initial poses. Table II shows the result of the proposed method by learning from 5-10 demonstrations per task: *Proposed w/o monitoring* removes the online predicate monitoring component by executing the learned policies in sequence. *Proposed w/ DP* shows `SymSkill` when the low level policy is replaced by state-input U-Net-based Diffusion Policy (DP).

`SymSkill` correctly segments trajectories and identifies the object in motion. The VLM is almost always able to determine the reference object correctly. For `RoboCasa` tasks, we take the identified o_{int} and o_{ref} and select the most frequent assignment across demonstrations, which yields perfect accuracy. Goal is specified by abstracting the symbolic state at the end of the majority of demonstrations. Failure cases arise primarily in PnP tasks, where the randomly generated containers are sometimes too tall (e.g., a salad bowl). In such cases, the arm carrying the item collides with the container, causing task failure.

With only 5–10 demonstrations per task, DP is severely data-limited. `Premotion` skill demonstrations originate from a wide variety of initial poses but cover only a narrow funnel toward the object, leaving most of the state space out of distribution—particularly for PnP tasks. At test time, action noise often drives the state further out of distribution, leading to near-zero or erratic behaviors; executions therefore typically fail before reaching the `motion` segment. For a few `motion` skills with low variability in the learned reference frame (e.g., pulling a door handle along an almost 1-D path), DP can produce qualitatively correct motions. However, because both `premotion` and `motion` must succeed, almost all task success rate is 0% . We also evaluated DP with data augmentation from the DS policy, as detailed on the project website, but found no success with DP either. In contrast, the $\mathrm{SE}(3)$ LPV-DS controller induces a convergent vector

TABLE III: Learned Operators from play data: each couples symbolic transitions with SE(3) DS skills. Operators are arranged by semantic affinity.

Operators	Human-Interpretable Summary	Preconditions	Effects	Maintain Conditions
Op7	Pick lid *from* cabinet	GripperOpen, Lid-in-cabinet	Gripper-in-lid, ¬Lid-in-cabinet, ¬GripperOpen	Lid-in-cabinet, GripperOpen
Op11	Pick lid *from* cookware	GripperOpen, Lid-in-cookware	Gripper-in-lid, ¬Lid-in-cookware, ¬GripperOpen	Lid-in-cookware, GripperOpen
Op1	Place lid → cabinet	Gripper-in-lid	Lid-in-cabinet, ¬Gripper-in-lid, GripperOpen	Gripper-in-lid
Op8	Place lid → cookware	Gripper-in-lid	Lid-in-cookware, ¬Gripper-in-lid, GripperOpen	Gripper-in-lid
Op9	Pick thing *from* drawer	GripperOpen, Thing-in-container, Thing-in-drawer	Gripper-in-thing, ¬Thing-in-drawer, ¬GripperOpen	Thing-in-container, Thing-in-drawer, GripperOpen
Op5	Pick thing *from* cookware	GripperOpen, Lid-in-cabinet, Thing-in-cookware	Gripper-in-thing, ¬Thing-in-cookware, ¬GripperOpen	Thing-in-cookware, Lid-in-cabinet, GripperOpen
Op10	Pick thing *from* container	GripperOpen, Thing-in-container	Gripper-in-thing, ¬Thing-in-container, ¬GripperOpen	Thing-in-container, GripperOpen
Op4	Place thing → drawer	Gripper-in-thing, Thing-in-cookware	Thing-in-drawer, ¬Gripper-in-thing, GripperOpen	Gripper-in-thing, Thing-in-cookware
Op3	Place thing → cookware	Gripper-in-thing, Lid-in-cabinet	Thing-in-cookware, ¬Gripper-in-thing, GripperOpen	Gripper-in-thing, Lid-in-cabinet
Op6	Place thing → container	Gripper-in-thing	Thing-in-container, ¬Gripper-in-thing, GripperOpen	Gripper-in-thing

field in the learned reference frame; its closed-loop stability prevents stalling and ensures steady progress to the goal even under perturbations.

For the symbol–skill co-invention baseline, we re-implemented Neural-Symbolic Imitation Learning (NSIL) [5] from scratch. NSIL constructs relative-pose trajectories for every pair of objects and identifies low-velocity segments as candidate predicates. It then incrementally selects predicates using hill-climbing beam search [6]. Since task-specific tuning is required, we focus on a qualitative comparison for two tasks: *OpenSingleDoor* and *PnPCounterToCab*. In our experiments, NSIL struggled in settings with multiple objects, where several plausible but spurious predicates could equally explain the demonstrations. Its reliance on near-optimal demonstrations further led to discarding semantically useful predicates. Moreover, the method proved sensitive to non-prehensile interactions, where meaningful contacts were often not included as candidate predicates. As a result, NSIL failed to recover reusable, semantically grounded predicates for these tasks. Finally, the limited amount of data in RoboCasa further makes it impossible for learning a policy.

B. Performing Multi-Step Task With No Additional Data

We created a new task, `StoreCheese`, in RoboCasa. The task is successful when the robot picks the cheese from the cabinet, places it on the counter, and closes the cabinet door. To execute this task, we load the previously learned symbols and skills from *OpenSingleDoor*, *CloseSingleDoor*, and *PnPCabToCounter*, and update operator preconditions via predicate evaluation across demonstrations. The operator from *PnPCabToCounter* task thus has the predicate *OpenSingleDoor-RelPose(Door, Cabinet)*, meaning door being open, as a precondition (illustrated as ${}^{o_{\text{ref}}}\psi_{o_{\text{int}}}$ in Fig.1). We then manually specify the goal predicates as {*CloseSingleDoor-RelPose(Door, Cabinet)*, *PnPCabToCounter-RelPose(Cheese, Counter)*}. With this setup, the Franka robot successfully plans the operator sequence: open the door, pick and place the cheese, and finally close the door. It completes the task by chaining together six skills and recovering from symbolic errors multiple times. Video of the experiment can be found on the project website.

C. Learning From Play In Real-World

We demonstrate our method can learn from play data in the real world. We set up a scene with block and banana (`thing_type`), red plate (`drawer_type`), white plate (`container_type`), dishrack (`cabinet_type`), lid

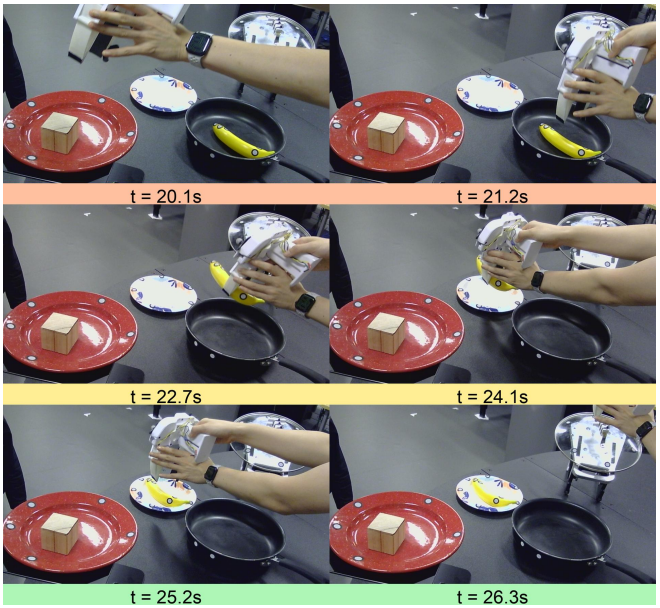

Fig. 5: Real-world data collection pipeline. We use a motion capture system to record object interactions in the workspace. Here we show one motion episode with a sequence of timestamped images; the manipulated object (o_{int}) is a banana. Frames with orange, yellow, and green banners denote the `premotion`, `motion`, and post-motion segments, respectively.

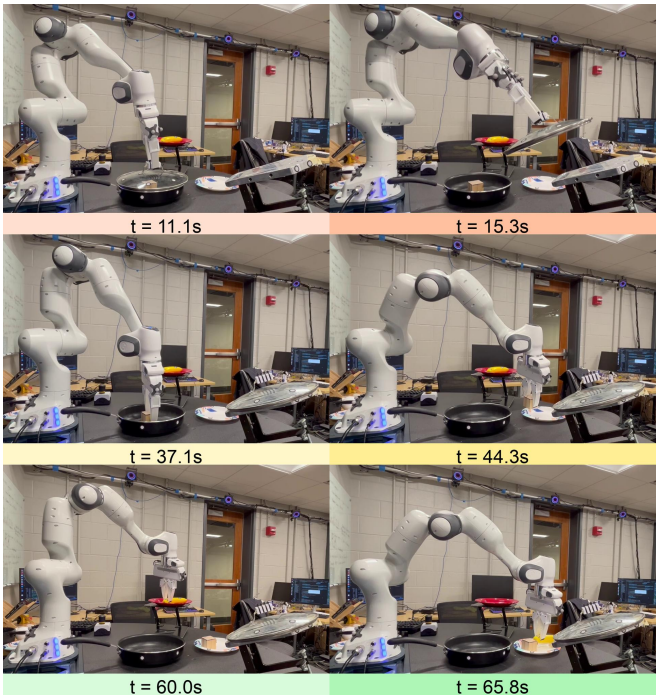

Fig. 6: Real-world execution of `SymSkill`. Each color denotes a distinct operator: lighter shades correspond to pick operators, while darker shades correspond to place operators. The symbolic goal is manually specified as {`RelPose(block, plate)`, `RelPose(banana, plate)`}.

(lid_type), and pan (cookware_type). During play data collection, the demonstrator uses a UMI gripper [30] to perform sequences of manipulation tasks such as closing the pan with a lid or placing the banana on a plate. We obtain the pose of objects and the gripper from a motion capture system and record the video data from a webcam. Fig. 5 shows the data collection process. Fig. 3 shows selecting reference frame process and the VLM prompt. We find that with minimal prompt engineering, VLM can correctly identify the reference frame o_{ref}, leading to correct learned predicates. Table III summarizes the learned operators from approximately 5 minutes of unsegmented real-world play. We find that our method learns semantically meaningful and logical operators from unsegmented data, such as recognizing that picking items from the pan requires first removing the lid to place it on the dishrack. An example is shown in Fig. 6.

We note that an interesting precondition in Op9 requires an item to be in the white plate before the robot can pick up another item from the red plate. We find this is indeed the case in all of the demonstrations. SymSkill therefore infers 'Thing-in-container' as a precondition. Although counterintuitive, this reflects actual household conventions that are rarely captured by generic LLM/VLM priors with little data, highlighting our method's sample efficiency. Consequently, at test time, if the goal is to pick from the red plate and the white plate is empty, the planner first inserts a preparatory step to place an item in the white plate, and only then proceeds with picking an item from the red plate. We also demonstrate reacting to human external disturbance and recovering from failure in a OpenSingleDoor task. The video of the experiment is on the project website.

VII. Conclusion and Future Work

We presented SymSkill, a symbol–skill co-invention framework that jointly learns relative-pose predicates for planning and DS-based skills for execution. Our results in simulation and on real robots show that SymSkill is significantly more sample-efficient and faster to learn than existing baselines, while enabling robust long-horizon manipulation. As future work, we plan to extend our framework to learn directly from egocentric video and to scale toward mobile manipulation scenarios, further broadening its applicability to real-world generalist robots.

Acknowledgment: We thank Bowen Li, Nishanth Kumar, Tom Silver and Rachel Holladay for the helpful discussions at various stages of the project. We thank Peng Qiu for helping out with setting up the simulator.

References

[1] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *The International Journal of Robotics Research*, p. 02783649241273668, 2023.

[2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[3] 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*, vol. 4, no. 1, pp. 265–293, 2021.

[4] N. Shah, J. Nagpal, P. Verma, and S. Srivastava, "From reals to logic and back: Inventing symbolic vocabularies, actions, and models for planning from raw data," *arXiv preprint arXiv:2402.11871*, 2024.

[5] L. Keller, D. Tanneberg, and J. Peters, "Neuro-symbolic imitation learning: Discovering symbolic abstractions for skill learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[6] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum, "Predicate invention for bilevel planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 12 120–12 129.

[7] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," in *Robotics: Science and Systems*, 2024.

[8] K. Kronander and A. Billard, "Passive interaction control with dynamical systems," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 106–113, 2015.

[9] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, May 2012.

[10] N. Figueroa and A. Billard, "A physically-consistent bayesian non-parametric mixture model for dynamical system learning." in *CoRL*, 2018, pp. 927–946.

[11] A. Billard, S. Mirrazavi, and N. Figueroa, *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. The MIT Press, 2022.

[12] S. Sun and N. Figueroa, "Se(3) linear parameter varying dynamical systems for globally asymptotically stable end-effector control," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 5152–5159.

[13] T. Li, S. Sun, S. S. Aditya, and N. Figueroa, "Elastic motion policy: An adaptive dynamical system for robust and efficient one-shot imitation learning," in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025.

[14] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the international conference on automated planning and scheduling*, vol. 30, 2020, pp. 440–448.

[15] J. Mendez-Mendez, L. P. Kaelbling, and T. Lozano-Perez, "Embodied lifelong learning for task and motion planning," in *Proceedings of the 7th Conference on Robot Learning (CoRL-23)*, 2023.

[16] S. Cheng, C. R. Garrett, A. Mandlekar, and D. Xu, "Nod-tamp: Generalizable long-horizon planning with neural object descriptors," in *8th Annual Conference on Robot Learning*, 2024.

[17] Z. Xue, S. Deng, Z. Chen, Y. Wang, Z. Yuan, and H. Xu, "DemoGen: Synthetic Demonstration Generation for Data-Efficient Visuomotor Policy Learning," in *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025.

[18] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox, "Mimicgen: A data generation system for scalable robot learning using human demonstrations," in *7th Annual Conference on Robot Learning*, 2023.

[19] C. Garrett, A. Mandlekar, B. Wen, and D. Fox, "Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment," in *8th Annual Conference on Robot Learning*, 2024.

[20] W. Wan, Y. Zhu, R. Shah, and Y. Zhu, "Lotus: Continual imitation learning for robot manipulation through unsupervised skill discovery," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 537–544.

[21] C. Wang, L. Fan, J. Sun, R. Zhang, L. Fei-Fei, D. Xu, Y. Zhu, and A. Anandkumar, "Mimicplay: Long-horizon imitation learning by watching human play," in *7th Annual Conference on Robot Learning*, 2023.

[22] L. P. Kaelbling and T. Lozano-Pérez, "Learning composable models of parameterized skills," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 886–893.

[23] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[24] W. Liu, N. Nie, R. Zhang, J. Mao, and J. Wu, "Blade: Learning compositional behaviors from demonstration and language," in *CoRL*, 2024.

[25] B. Li, T. Silver, S. Scherer, and A. Gray, "Bilevel Learning for Bilevel Planning," in *Proceedings of the Robotics: Science and Systems (RSS)*, 2025.

[26] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann, "Neural descriptor fields: Se (3)-equivariant object representations for manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6394–6400.

[27] A. Athalye, N. Kumar, T. Silver, Y. Liang, T. Lozano-Pérez, and L. P. Kaelbling, "Predicate invention from pixels via pretrained vision-language models," *arXiv preprint arXiv:2501.00296*, 2024.

[28] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," *arXiv preprint arXiv:2507.06261*, 2025.

[29] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4166–4173.

[30] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song, "Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots," in *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Appendix I
Online Algorithm

We show the pseudo-code of our online algorithm.

Require: Current State $\mathbf{x}_0$, Goal atoms s_g, Learned operators Ω (each operator $\alpha = \langle$ pre, maintain, eff, *skill* $\rangle$), all objects $\mathcal{O}$

1: Initialize replan_count $\leftarrow 0, failmem \leftarrow \{\}$ ▷ failure memory
2: CurrentPlan$(\alpha_1, \ldots, \alpha_n) \leftarrow$ SymbolicPlanner(s_0, s_g, Ω)
3: **if** s_g unreachable or replan_count ≥ 20 **then**
4: **return** Failure
5: **end if**
6: $\alpha_{exe} \leftarrow$ CurrentPlan$[i]$
7: $\alpha_{prev} \leftarrow$ CurrentPlan$[i-1]$
8: **if** $\alpha_{exe} \in failmem$ **then**
9: Goal Pose $\leftarrow$ sample(eff(α_{exe})) ▷ Sample in ${}^{o_{\text{int}}}\psi_{ee}$ or ${}^{o_{\text{ref}}}\psi_{o_{\text{int}}}$
10: **else**
11: Goal Pose $\leftarrow 0$
12: **end if**
13: **while** maintain$(\alpha_{exe}) \in$ abstract$(\mathbf{x})$ **do**
14: $< f, g > \leftarrow skill(\alpha_{exe})$
15: $f' \leftarrow \mathbf{M}(\mathcal{O}_{-o_{\text{int}}})f$ ▷ Obstacle Avoidance
16: **if** $f' < \epsilon$ **then** ▷ Current skill ending
17: **if** eff$(\alpha_{exe} \notin$ abstract$(\mathbf{x}))$ **then**
18: Add (α_{exe}) to $failmem$ ▷ Current skill failure
19: Go To Line: 2 ▷ Current skill success
20: **end if**
21: $i \leftarrow i + 1$
22: Go to Line: 6
23: **end if**
24: **end while**
25: Add (α_{prev}) to $failmem$ ▷ Failure of maintain predicates
26: Go To Line: 2

Appendix II
Extended Analysis of [5]

a) Difficulty 1: Assumption of Optimal Demonstrations.: NSIL requires that candidate predicate sets yield plans whose length matches the demonstration length. In practice, demonstrations in RoboCasa often contain suboptimal behaviors, such as multiple approaches before grasping. For instance, in `OpenSingleDoor` or `PnPCounterToCab`, demonstrations sometimes include repeated approaches, leading to longer trajectories than the optimal plan. As a result, valid predicate sets are rejected because the demonstration is not optimal. We relaxed this by introducing a penalty for mismatched lengths rather than strict rejection, but the issue remains fundamental.

b) Difficulty 2: Ambiguity from Distractor Objects.: Including irrelevant objects significantly increases ambiguity. For a simple pick-and-place task with one distractor, low-speed analysis generated 13 candidate predicates. Beam search often selected fragile ones, e.g., RelPose(DoorHandle, Object), because incorrect relationships (object to door) has the same cost as semantically meaningful ones (object to cabinet). Such predicates fail to generalize if the distractor moves. Our method mitigates this by using VLM-based semantic grounding to select reference objects, bypassing spurious associations.

c) Difficulty 3: Noisy Non-Prehensile Interactions.: For articulated objects, grasping is frequently non-prehensile (e.g., pushing a door handle rather than firmly holding it). This makes RelPose(Gripper, Handle) a noisy predicate and artificially inflates the demonstration symbolic sequence length. In our trials with four demonstrations of door opening, the optimization incorrectly favored gripper-to-cabinet relations, which do not reflect the actual manipulation. This highlights NSIL's difficulty in capturing non-prehensile skills robustly.

d) Summary.: Overall, NSIL's reliance on optimal demonstrations, its vulnerability to distractors, and its fragility in non-prehensile settings limit its robustness in realistic environments such as RoboCasa. By contrast, our framework leverages semantic grounding (via VLMs) for predicate discovery and SE(3) DS policies with stability guarantees for skill execution, making it more robust under limited demonstrations and realistic variations.

Appendix III
DP with Data Augmentation

We used the trained SE(3) LPV-DS policies to generate 100 rollouts for each skill as additional training data for the diffusion policy (DP). Initial end-effector poses were sampled from an SE(3) Gaussian fit to the initial poses in the training set. Example training data, augmented data, and a rollout generated by the trained DP are shown in Figure 7.

We used trained SE(3) LPV-DS policies to generate rollouts as additional training data for diffusion policy (DP). Initial end-effector poses were sampled from an SE(3) Gaussian fit to the initial poses in the training set. All training data, augmented data, and a rollout from the trained DP are shown in Figure 7.

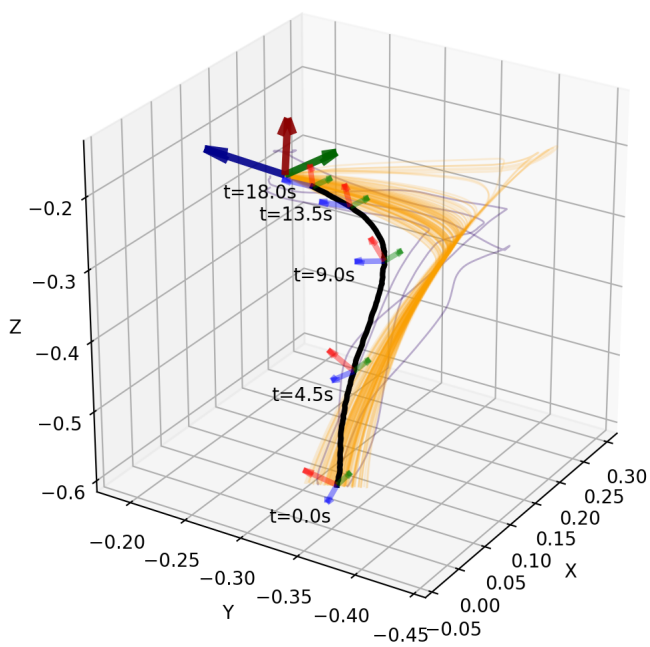

Fig. 7: DP rollout for the `premotion` segment of *OpenSingleDoor*. Purple trajectories are 4 training demonstrations used for `SymSkill`, yellow trajectories are augmented data generated by SE(3) LPV-DS, and the black curve is a rollout from the trained DP. The poses (red, green, blue axes) indicate end-effector orientation at several timestamps; the larger pose denotes the averaged final orientation across all demonstrations.

We evaluated DP with data augmentation in `RoboCasa`.

Although the rollout in Figure 7 appears successful, we observed degraded orientation performance in simulation, leading to a zero success rate. We hypothesize this is due to the robot's kinematic constraints, which push the end-effector into regions outside the training distribution. DP succeeds on relatively simple tasks (e.g., *CloseDrawer*) that do not require precise orientation control, but fails on tasks such as *OpenSingleDoor* and *PnPCounterToCab*, where the policy consistently approaches the o_{int} but cannot achieve the grasp orientation required for success.