AGENT PRIORS-GUIDED POLICY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Robots that learn from a few demonstrations often require two forms of generalization. Compositional generalization recombines skills to solve new tasks, and skill generalization lets the learned policy behind each skill work in new situations. The two depend on each other, yet information is lost between composition and the skills it calls. Where a skill works is determined by the structure its policy is trained with, while composition sees the skill only through a separate description, such as a name, an instruction, or a symbolic operator, that omits this structure. Our key idea is to use each policy's structural prior as part of the interface between composition and the skill. A structural prior states what a behavior depends on, for example that a grasp depends only on the gripper's pose relative to the object. Built into training, it shapes where the policy generalizes; stated in language, it tells composition where the policy applies. We instantiate this idea in Agent Priors-guided Policy Learning (APPL). A construction agent segments complete demonstrations into reusable skills, proposes several structural priors for each skill, and trains and verifies one policy per prior. A runtime agent then selects among these prior-specific policies and composes them toward new task goals using their interfaces. Across MetaWorld and long-horizon ManiSkill tasks, APPL improves out-of-distribution skill generalization and enables previously unseen skill compositions; ablating the interface information substantially reduces performance. These results support the use of training-time structural assumptions as a bridge between skill learning and skill composition.

1 INTRODUCTION

Robots in homes and warehouses are asked to do far more than they were shown. Demonstrations are costly, so a robot typically learns from a few demonstrations of a few tasks, while deployment brings new goals, new orderings of familiar operations, and objects in new places. Handling this requires two kinds of generalization. *Compositional generalization* solves new tasks by recombining learned skills, and *skill generalization* lets each skill still execute when objects move or when it starts from the state another skill left behind. The two depend on each other. A task-level decision is only as good as its knowledge of what each skill can physically do, and a skill is only useful if it generalizes to the states that task-level decisions put it in. Information, however, is lost between the task level and the skill level. The task level may reason about goals and sequences without knowing where a learned skill actually works, and each skill is trained from a few demonstrations without the structure that the task level relies on.

Existing approaches connect the task level and the skill level in different ways. Vision-language-action models couple task semantics and low-level control in a single model (Kim et al., 2025; Black et al., 2025), but the conditions under which learned behaviors generalize remain implicit, and robustness to layout and object shifts remains limited (Fei et al., 2026). Task and motion planning keeps the levels separate and connects them through symbolic preconditions and effects (Garrett et al., 2021), which are specified by hand or learned from data (Konidaris et al., 2018; Silver et al., 2023b). Within this line, SymSkill (Shao et al., 2025) co-invents predicates, operators, and skills from demonstrations so that one structure serves both levels, but that structure is drawn from a predefined family of relative-frame predicates and dynamical-system controllers. Agentic systems can call policies as tools and use language as the interface (Ahn et al., 2022; Shi et al., 2025; Zhang et al., 2026). A skill name or instruction tells the agent what a tool is for, but not the physical conditions under which it works, so the agent cannot ground its decisions in the tool's actual ability, and tran-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

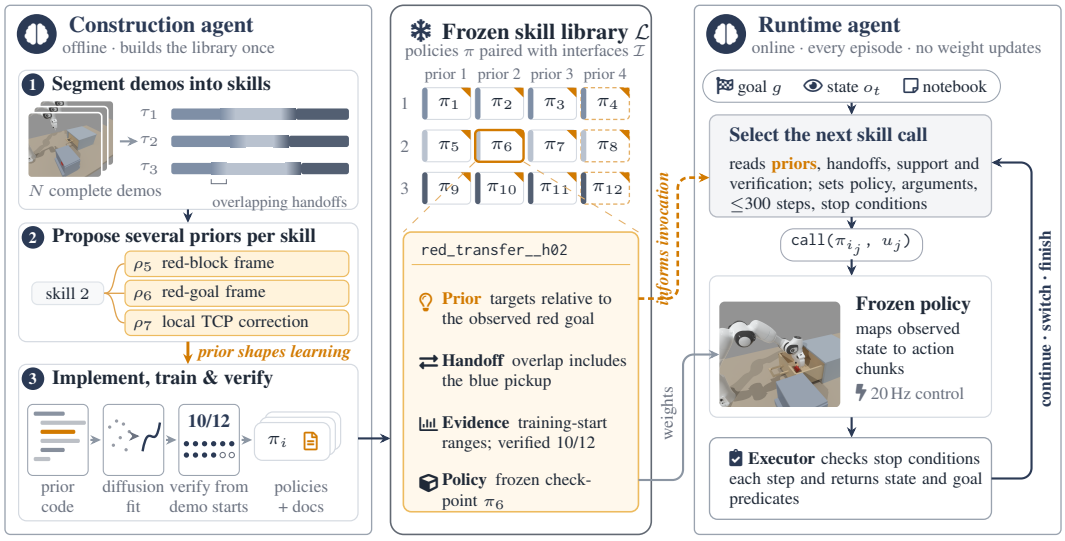

Figure 1: **APPL builds prior-specific policies offline and composes them online.** The construction agent segments demonstrations, proposes several priors per skill, trains one documented policy per prior, and verifies each policy from demonstrated entry states. The runtime agent reads these documents to choose a policy, its arguments, a duration, and stop conditions. Orange marks the prior's two roles.

sitions between skills remain a common source of failure (Rui et al., 2026). Effective generalization therefore strongly depends on the *interface* between the task and skill levels.

Our key idea is to *use each skill's structural prior as part of this interface*. A structural prior specifies how a skill is designed to generalize, for example by expressing actions relative to the manipulated object rather than in absolute world coordinates. We use this prior in two ways. First, it is implemented in the skill through its representation or training objective, which shapes how the skill generalizes beyond its demonstrations. Second, the same prior is exposed to a task-level runtime agent, together with the skill's training support and handoff conditions. The agent can therefore reason not only about *what* a skill does, but also about *why and where* a particular implementation is expected to generalize. For example, if a grasp skill is trained in an object-relative frame, the agent can prefer it when the object appears at a new absolute location. The structural prior therefore serves both as an inductive bias for learning and as an explicit hypothesis about when the resulting skill should be applicable.

We instantiate this idea in **Agent Priors-guided Policy Learning** (APPL, shown in Fig. 1). A construction agent segments complete demonstrations into reusable skills and proposes several structural priors for each skill, rather than committing to one predefined abstraction or one policy implementation. It trains one Diffusion Policy (Chi et al., 2023) for each prior and stores each policy together with a description of its prior, handoff conditions, observed training support, and verification evidence. The resulting library may therefore contain several implementations of the same skill whose different inductive biases make them suitable in different situations. At deployment, a runtime agent reads these descriptions to choose among the frozen policies, instantiate their arguments and stopping conditions, and compose them toward the task goal. In this way, information that shaped a policy during learning remains available when that policy is later selected and composed. APPL thus turns the choice of policy structure from a fixed design decision into a per-skill hypothesis that can be proposed, implemented, and reused at runtime.

Our experiments test both roles of the structural prior. On six MetaWorld tasks, agent-designed priors substantially improve few-demonstration out-of-distribution skill generalization over vanilla Diffusion Policy and a fixed relational prior, reaching 89.6% OOD success with two demonstrations versus 37.9% for the fixed relational prior. On five long-horizon ManiSkill tasks with twelve demonstrations each, APPL achieves 50.0% success under shifted object configurations versus 10.0% for full-task Diffusion Policy, 92.5% on task-level variants with intermediate starts or requested early

termination, and solves 8 of 16 previously unseen skill compositions. Importantly, when the same learned policy library is retained but its construction-time interface information is hidden from the runtime agent, success falls significantly. Together, these results support the central hypothesis of APPL that the structural assumptions used to make a skill generalize can also provide useful information for deciding when and how that skill should be used.

2 Related Work

APPL targets compositional and skill generalization together by using each skill's structural prior as the interface between composition and the skill's policy. We review work on composing learned skills, on structural priors for few-demonstration policies, and on language models that design parts of the learning pipeline. Appendix D discusses further work and compares APPL with the closest methods in detail.

Composing learned skills. Task and motion planning composes skills through symbolic preconditions and effects (Garrett et al., 2021), and a long line of work learns such abstractions from data as symbols grounded in skills, operators, or invented predicates (Konidaris et al., 2018; Silver et al., 2023a;b; Liang et al., 2025). Skills with symbolic interfaces can also be learned through planner-guided reinforcement learning (Cheng & Xu, 2023) or from demonstrations (Liu et al., 2025), and skill boundaries can be discovered from unsegmented trajectories (Zhu et al., 2022; Wan et al., 2024). Chaining fails when one skill ends in a state its successor does not support, and transition policies and skill-chaining methods address these handoffs (Lee et al., 2019; 2022; Mishra et al., 2023). SymSkill is the closest precedent to APPL (Shao et al., 2025). It jointly learns predicates, operators, and dynamical-system skills in relative frames from unsegmented demonstrations, and a symbolic planner composes and reorders these skills to reach new goals. Its structure, however, is fixed in advance to relative frames and dynamical-system controllers. MaestroMotif and SCALAR share language skill specifications between training and composition (Klissarov et al., 2025; Zabounidis et al., 2026), but these specifications describe what a skill achieves rather than the inductive bias of its policy. Language-model agents compose skills more flexibly (Ahn et al., 2022; Huang et al., 2023; Shi et al., 2025), yet they know each skill only by its name or a feasibility estimate, and transitions or handoffs between skills can fail (Rui et al., 2026). APPL instead lets an agent choose a different prior for each skill and keep alternative policies, and the prior that shapes each policy also tells the runtime agent where that policy applies.

Structural priors for few-demonstration policies. Policies learned from few demonstrations, such as diffusion policies (Chi et al., 2023), tend to reproduce demonstrated behavior and can generalize poorly to out-of-distribution (OOD) states (He et al., 2026). Building task structure into the policy helps, for example through object-centric representations (Zhu et al., 2023), functional correspondence (Tang et al., 2025), affordances and contact locations (Deng et al., 2026; Zhou et al., 2023), equivariance and task-relative frames (Wang et al., 2025; Rana et al., 2025), and visual cues (Dai et al., 2025). Each such prior encodes an assumption that suits one family of operations, and a human chooses it for each task. The prior is also used only in training and not in the system that later composes the policy. Vision-language-action models obtain breadth from large-scale data rather than explicit structure (Kim et al., 2025; Black et al., 2025), yet they remain sensitive to changes in layout and object pose (Fei et al., 2026).

Language models that design learning systems. Language models already design parts of robot learning pipelines, including reward functions (Ma et al., 2024a), simulation tasks and training data (Wang et al., 2024d;c), state abstractions (Peng et al., 2024), and keypoints (Fang et al., 2025). LGA and KALM are closely related to our work, since they let a model choose representation priors for imitation learning. Each automates one kind of prior, produces one design per task, and uses that design only to train the policy. In contrast, APPL's construction agent proposes several kinds of priors for each skill, keeps the resulting policies as alternatives, and reuses each prior as the description over which the runtime agent composes skills.

3 Problem Formulation

Setting and objective. A robot acts with observations $o \in \mathcal{O}$ and actions $a \in \mathcal{A}$, and $\mathrm{Succ}(g, o) \in \{0, 1\}$ indicates whether goal g holds at o. A *skill* k is a reusable operation with a sub-goal β_k, such

as opening a drawer, and a *policy* π_i implements skill $k(i)$. A system $\mathcal{S} = (\mathcal{C}, \mathcal{L})$ consists of a library $\mathcal{L} = \{(\pi_i, \mathcal{I}_i)\}_{i=1}^{K}$, possibly with several policies per skill, and a composer $\mathcal{C}$ that solves a task by calling policies in sequence, seeing each only through its interface $\mathcal{I}_i$. Construction receives N complete, unsegmented demonstrations $\mathcal{D}_N$ with goals in $\mathcal{G}_{\text{train}}$ and task knowledge $\mathcal{K}$, and outputs a frozen system $\mathcal{S}$ that maximizes expected success,

$$J(\mathcal{S}) = \mathbb{E}_{g \sim P_{\text{test}},\, o_0 \sim \mu_{\text{test}}(g)} \big[\operatorname{Succ}(g, o_T) \big]. \tag{1}$$

Skill and compositional generalization. Let the *success region* $W(\pi_i) \subseteq \mathcal{O}$ be the set of start observations from which π_i achieves $\beta_{k(i)}$ with probability at least $1-\epsilon$. *Skill generalization* requires $W(\pi_i)$ to contain start states absent from the demonstrations, such as shifted objects or states left by a different preceding skill. Changes in object position with the task and skill sequence fixed test this ability; we call this setting *motion-level OOD*.

Compositional generalization requires solving tasks (g, o_0) through skill sequences or handoffs absent from the demonstrations. Intermediate skills may be skipped, skills reordered, or a task may begin with a sub-goal already satisfied but with entry conditions for the next skill that differ from the demonstrated handoff. For example, putting the blue block into the drawer while leaving the red block inside skips the demonstrated removal of the red block and changes the placement skill's entry conditions. We separately evaluate *task-level OOD*, which changes where a demonstrated task begins or is requested to end: resuming near a demonstrated stage or terminating at a requested intermediate sub-goal after a prefix of the demonstrated sequence. These cases test adapting execution to the current progress and requested endpoint while preserving the demonstrated skill order. Object displacement alone does not constitute a new composition.

Reliable composition requires the calls' sub-goals to jointly achieve g and each call j to start inside the success region of its policy, $o_{t_j} \in W(\pi_{i_j})$. The two kinds of generalization are thus coupled: a new sequence or handoff can also create start states that skill generalization must cover.

The interface and desired properties. The composer cannot observe $W(\pi_i)$ in general. Instead, the interface $\mathcal{I}_i$ states an applicability region $A_i \subseteq \mathcal{O}$, and the composer can check only $o_{t_j} \in A_{i_j}$. Information is lost whenever A_i and $W(\pi_i)$ differ. If $A_i \not\subseteq W(\pi_i)$, composition calls a policy where it fails. If A_i is much smaller than $W(\pi_i)$, composition gives up usable calls. A useful interface is therefore *faithful*, $A_i \subseteq W(\pi_i)$, and *informative*, $A_i \approx W(\pi_i)$. The problem is to construct, from $\mathcal{D}_N$ and $\mathcal{K}$ alone, policies whose success regions cover the states that new compositions create, together with interfaces that describe those regions faithfully and informatively.

4 Agent-Guided Prior Design and Skill Composition

We present APPL, which connects skill learning and skill composition through structural priors. The central idea is to use the same structural assumption in two roles: to shape how a skill policy generalizes during training, and to describe when that policy is expected to be applicable at runtime.

Figure 1 summarizes our system. Construction happens offline in stages. First, a construction agent segments complete demonstrations into reusable skills. Next, for each skill it proposes several structural priors, implements and trains one policy per prior. It then verifies the trained policies on demonstrated entry states. Finally, it writes the interface for each candidate and freezes the resulting library. At runtime, a separate agent reads these interfaces to choose among the frozen policies and compose them toward a new task goal.

4.1 Structural priors in the interface

Different policies can fit the same demonstrations while having different success regions $W(\pi_i)$. We use a *structural prior* ρ_i to encode additional assumptions about the relations the skill should depend on and how its behavior should respond to changes in the scene. For example, an object-relative prior expresses the assumption that a manipulation behavior can be reused at different absolute object locations when the relevant object-relative geometry is preserved.

APPL uses each prior in two complementary ways. Its *training realization* shapes the learned policy. A prior may be implemented through the policy representation, action parameterization, architec-

ture, or an auxiliary training objective. We write the corresponding policy class as Θ_{ρ_i} and train

$$\theta_i = \arg\min_{\theta \in \Theta_{\rho_i}} \mathbb{E}_{\mathcal{D}_{k(i)}}\big[\ell_{\mathrm{BC}}(\theta)\big] + \lambda_i \, \ell^{\mathrm{aux}}_{\rho_i}(\theta). \tag{2}$$

The prior thereby shapes how the policy behaves beyond its demonstrations and, consequently, its success region $W(\pi_i)$. We refer to this role as *coverage*.

The same prior also has a *description realization*, which forms part of the runtime interface $\mathcal{I}_i$. The prior and the observed training support together provide a hypothesis about where the policy should apply. For a representation-based prior, let $\phi_{\rho_i}(o)$ denote the relevant transformed observation and let $\mathcal{R}_i$ denote the region covered by the transformed skill demonstrations. We define the corresponding hypothesized applicability region as

$$A_i = \{o : \phi_{\rho_i}(o) \in \mathcal{R}_i\}. \tag{3}$$

For example, under an object-relative prior, a state with an object at a new absolute location may still lie in A_i when its relative configuration lies within the demonstrated range. The interface communicates this hypothesis through the prior description, measured training support, and handoff conditions. The runtime agent uses this information to decide which policy to invoke, which we refer to as *selection*.

This construction directly connects to the interface criteria of Section 3. Using the same prior for training and for the runtime description is intended to improve interface faithfulness, $A_i \subseteq W(\pi_i)$, because the stated applicability is based on the assumptions that shaped the policy. This relationship is not guaranteed since it depends on both the validity of the prior and how well the trained policy realizes it. Informativeness additionally requires A_i to capture as much of $W(\pi_i)$ as possible rather than being unnecessarily conservative.

The prior therefore couples the two parts of the problem: its training realization determines *coverage*, while its description realization supports *selection*. Since the appropriate structure can differ across skills and situations, construction must choose both a segmentation σ and priors for the resulting skill policies:

$$\max_{\sigma,\, \{\rho_i\}} J\big(\mathcal{S}(\sigma, \{\rho_i\}, \mathcal{D}_N)\big). \tag{4}$$

This optimization ranges over an open-ended design space of policy implementations and interfaces, and the deployment objective J is unavailable during construction. APPL therefore uses a language-model agent to propose, implement, and evaluate candidate designs from the demonstrations and task knowledge.

4.2 Segmenting demonstrations into skills

The construction agent receives the N complete demonstrations, the task goals, and the observation and control conventions in $\mathcal{K}$. It proposes skill boundaries on every trajectory and groups segments that perform the same reusable operation into skill datasets $\mathcal{D}_k$. Adjacent skill segments are intentionally overlapped around their transition or handoff. In particular, a skill's training data include part of the end of its predecessor and the beginning of its successor. This overlap broadens the training support around handoff states, allowing a successor policy to take over from states that its predecessor actually reaches, including states before nominal completion. The overlap reuses transitions from the original demonstrations and introduces no additional demonstration data.

4.3 Proposing, implementing, and documenting priors

Proposing alternative priors. For each skill, the construction agent proposes several priors from different families rather than committing to a single design. It is generally not possible to know which structural assumption will generalize best to every state in which the skill may later be invoked. As such, retaining several prior-specific policies gives the runtime agent alternative implementations of the same skill.

Implementing the training realization. For each proposed prior ρ_i, the agent writes an implementation against a fixed policy interface. The implementation constructs inputs from the causal observation history, encodes demonstrated actions in the coordinates specified by the prior, decodes

predicted actions back into native commands, and may introduce the auxiliary loss $\ell_{\rho_i}^{\text{aux}}$ from Equation 2. Each prior produces a separate conditional diffusion policy (Chi et al., 2023), trained using a common recipe. For example, consider a prior that represents actions in the observed object frame (R_t, p_t) at the start of an action chunk. End-effector targets are transformed according to

$$x^{\text{loc}} = R_t^\top (x^{\text{world}} - p_t), \quad \Delta x^{\text{loc}} = R_t^\top \Delta x^{\text{world}}, \quad \Delta x^{\text{world}} = R_t \Delta x^{\text{loc}}. \tag{5}$$

This parameterization encourages the learned behavior to depend on object-relative rather than absolute motion. In our experiments, a shared inverse-kinematics module converts decoded end-effector targets into native joint commands.

Verifying policy realizations before freezing. For each candidate skill policy, the construction agent gathers limited execution evidence. In our experiments, this verification step uses only demonstrated skill-entry states. Each policy is initialized from the entry state of its corresponding segment in every demonstration, executed for 1.5 times the segment length, and evaluated against the demonstrated exit condition at the final state. The construction agent receives these results and writes a verification report that separates observed outcomes from its inferences about the policy. The agent then proposes one additional prior for each skill using this feedback. The corresponding policy is trained and verified in the same way, after which the agent writes a usage note for every policy.

Constructing the runtime interface. The construction agent then records an interface for every trained skill policy,

$$\mathcal{I}_i = (d_i,\, h_i,\, s_i,\, v_i). \tag{6}$$

The four fields provide complementary information about the skill. The prior description d_i states the structural assumption behind ρ_i, including which observations and relations the policy was designed to depend on, which variations it is expected to tolerate, and known limitations. The handoff description h_i records entry and exit conditions and the predecessor and successor states represented by the overlapping segments. The support description s_i summarizes the demonstrated training support, such as ranges of skill-entry states. Finally, v_i stores evidence gathered during verification. Together, these fields describe the policy's hypothesized applicability region A_i and the evidence available for supporting that hypothesis. Each interface also declares the typed arguments required by the policy, such as the object to manipulate and its destination.

4.4 Composing skills at runtime

In our implementation, the policy library $\mathcal{L}$, its interfaces, and the runtime agent's model, prompt, and tools are frozen after construction. At each decision point during execution, the runtime agent receives the task goal g, the current state and goal predicates, recent interaction history, its episode notebook, and the available policy interfaces $\mathcal{I}_i$. It can inspect the full interface of any policy before invoking it. Using this information, the agent chooses a policy and its call arguments u_j. A call specifies the policy's typed arguments, an execution duration, and stop conditions over observed quantities, either absolute or relative to the start of the call. For example, the agent may require both a gripper width below 3 cm and the grasped object to be lifted by more than 4 cm.

The runtime agent $\mathcal{R}_\alpha$ thereby instantiates the composer $\mathcal{C}$ introduced in Section 3:

$$(i_j, u_j, m_{j+1}) \sim \mathcal{R}_\alpha(\cdot \mid h_{t_j}, g, m_j, \mathcal{L}), \qquad \mathbf{a}_{t:t+H} \sim \pi_{i_j}(\cdot \mid h_t, u_j), \quad t_j \le t < t_{j+1}, \tag{7}$$

where h_t is the interaction history, m_j is the episode notebook, and $\mathbf{a}_{t:t+H}$ is an action chunk of H steps. The executor runs the selected policy with fresh observations and checks the specified stop conditions after every step. When a condition fires, control returns to the runtime agent together with the updated state and goal predicates. In our implementation, the executor also interrupts a call immediately when the overall task goal is satisfied. The agent can then continue the current skill, switch to another policy for the same skill, invoke a different skill, or terminate the task. It cannot write control code or modify the frozen library, and its memory is reset between episodes.

Runtime composition is where the two roles of the structural prior come together. The training realization determines the states from which each policy can actually succeed, while the interface exposes the construction-time hypothesis about that competence. The runtime agent uses the latter to select and chain policies so that the library's learned coverage can be exploited when solving new tasks.

Table 1: Exp. 1 hidden-test success (%), averaged equally over six tasks, with N demonstrations per task. IID uses fresh states within the demonstrated factor ranges; OOD averages recombined (C) and extrapolated (E) states equally. Mean averages OOD over the four values of N. Bold marks the best entry in each column. q_1 is the first proposal (A1); q_3 and q_4 are selected by development OOD success from A1–A3 and from A1–A4.

	$N = 2$		$N = 5$		$N = 10$		$N = 20$		Mean
Method	IID	OOD	IID	OOD	IID	OOD	IID	OOD	OOD
Diffusion Policy (B0)	73.33	28.96	82.50	37.29	97.50	38.54	100.00	46.67	37.86
Relational prior (B1)	85.83	37.92	93.33	42.71	**100.00**	52.29	100.00	56.25	47.29
APPL, first proposal (q_1)	84.17	56.04	95.83	63.54	**100.00**	73.33	100.00	78.75	67.92
APPL, best of three (q_3)	90.83	80.00	**100.00**	81.46	**100.00**	90.42	100.00	88.96	85.21
APPL (q_4)	**96.67**	**89.58**	**100.00**	**93.54**	**100.00**	**93.33**	100.00	**93.13**	**92.40**

5 EXPERIMENTS

This section reports on two experiments designed to evaluate APPL's ability to generalize. Specifically, Exp. 1 tests coverage in isolation: whether agent-designed priors make individual skills generalize. Exp. 2 tests coverage and selection together: whether priors, used both in training and as descriptions read by a runtime agent, let skills compose into long-horizon tasks when objects are shifted and when the requested task differs from every demonstrated one.

5.1 AGENT-DESIGNED PRIORS FOR SKILL GENERALIZATION

Exp. 1 evaluates the training realization of priors and asks: given a skill and a few demonstrations, can an agent design and implement priors that make the skill's policy generalize? Skills and subgoals are supplied, so no composition is involved.

Setting and comparisons. We use six adapted MetaWorld tasks (Yu et al., 2020): pick-place-wall, assembly, drawer-open, door-open, peg-insert-side, and stick-push. Each has 20 successful demonstrations. Test states vary two spatial factors per task: IID states stay within the demonstrated combinations, C recombines the factors, and E places both outside their demonstrated intervals; OOD weights C and E equally. All policies receive the same state channels, and the agent may also inspect demonstration images while designing. B0 is a vanilla diffusion policy (Chi et al., 2023), and B1 adds a learned encoding of three predefined role-relative vectors; B1 is one fixed relational prior, not an exhaustive set of expert priors. Agent candidates share the diffusion backbone, the 20,000-update budget, the controller, and the evaluation, but may change representations, encoders, action coordinates, and auxiliary losses.

Candidate design and selection. In each of the 24 task–N conditions, the agent submits three candidates (A1–A3) before any performance feedback, then designs a fourth (A4) from their development results on fresh IID, C, and E states and trains it from scratch. We report $q_1 = $ A1, the first proposal, and q_3 and q_4, the candidates with the best development OOD success among A1–A3 and A1–A4, with fixed tie-breaks; the three share 144 trained models. All designs and selections are frozen before testing, and neither IID nor test scores are used to select candidates (training details in Appendix A.3).

Results. Overall, the experiment results support the notion that agent provided priors improve skill generalization. Agent-designed priors raise six-task macro OOD success at every N (Table 1). At $N = 2$, q_4 reaches 89.58%, versus 28.96% for B0 and 37.92% for B1; its margins over B1 are 51.67, 50.83, 41.04, and 36.88 percentage points for $N = 2$, 5, 10, and 20. IID success is similar across methods at larger N, so the gains come from generalization rather than fitting. Intervals and per-task results are in Appendices A.4 and A.6.

What the agent designs and what feedback adds. The first proposal alone beats B1 by 18.1–22.5 percentage points. Proposing three priors and selecting by development OOD success adds 10.2–24.0 points, and the feedback round adds 2.9–12.1 more: A4 is selected in 15 of 24 conditions, and test success rises in 10, ties in 13, and drops in one. Two revisions show how this happens. At drawer/$N = 2$, A1 expresses the hand–handle relation and the actions in the cabinet frame (75.00%).

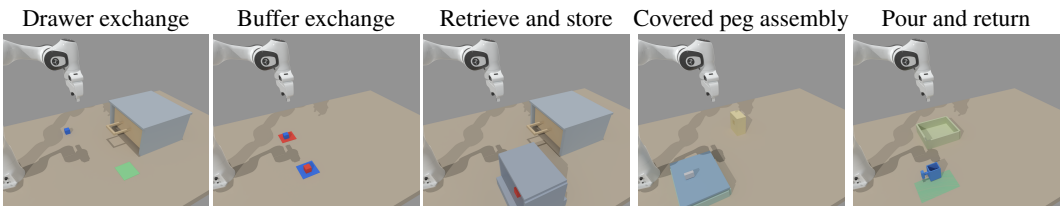

Figure 2: Five long-horizon tasks at their demonstrated start states. The robot exchanges blocks between a drawer and a pad, swaps two blocks through a buffer, stores a block from a low tunnel in a drawer, uncovers and inserts a peg, and pours particles into a bowl before returning the container.

Because demonstrated planar actions are close to $4(p^{\text{handle}} - p^{\text{hand}})$, A4 subtracts this term in the action encoding, so the network predicts only a residual that extrapolates to new positions, reaching 100.00%. At assembly/$N = 5$, the demonstrations correlate nut and goal positions, and development feedback showed crossed layouts failing. A4 attenuates goal features and the goal-aligned frame until the ring is lifted, removing that shortcut, and rises from 41.25% to 93.75%. The feedback covers recombined and extrapolated development states, which lets revisions target the failing factor combinations. A revision can also hurt. For example, door/$N = 20$ falls from 100.00% under q_3 to 96.25% under q_4. Because each step from q_1 to q_4 adds candidates and development selection, $q_4 - q_3$ does not isolate the effect of feedback (Appendix A.5).

5.2 Long-horizon composition under shifted objects and unseen task variants

Exp. 2 asks: can a runtime agent compose prior-specific skill policies to complete tasks from states and goals absent from the demonstrations? We use five ManiSkill tasks, each with twelve successful demonstrations collected by a scripted planner (Figure 2).

Methods and shared settings. All methods use the same demonstrations, structured state observations, and Cartesian action interface with shared inverse kinematics (Appendix B.2). *DP* learns one full-task diffusion policy; *SinglePrior* learns one full-task policy with a prior designed by the same language model as APPL, without a runtime agent. Because language is a common interface between task-level reasoning and skills, *Agent+VLA* replaces the library with one vision-language-action policy (Black et al., 2025) that the same runtime agent calls with a skill name as its instruction. We compare four ablations. Three keep APPL's verified four-policy-per-skill library: *w/o interface information* hides every field of $\mathcal{I}_i$ (prior, handoff, support, and verification), leaving only skill names, anonymized policy labels, and call interfaces; *w/o prior information* hides the prior, handoff, and support descriptions and the usage notes, but keeps each policy's verification score and exit criterion; *w/o HL agent* replaces the agent with a fixed rule that follows the demonstrated skill order and cycles through policies. The fourth, *w/o verification*, keeps three policies per skill and skips verification; full APPL adds a fourth policy per skill and verification evidence and usage notes.

Evaluation. *Motion-level OOD* shifts objects beyond demonstrated position ranges in eight frozen layouts per task, keeping the task fixed. *Task-level OOD* has eight cases per task: four resume a partly completed task near a demonstrated stage, and four request a demonstrated sequence prefix. It tests skill selection for task progress and requested sub-goals. *Composition* has 16 cases that skip or reorder skills, or initialize a completed sub-goal with entry conditions absent from demonstrated handoffs. DP and SinglePrior lack goal inputs and are compared only on motion OOD. Success is credited at first goal attainment with executor-assisted stopping; evaluation details and paired comparisons are in Appendix B.

APPL handles shifted objects better than full-task policies. APPL succeeds on 50.0% of motion-OOD layouts, versus 10.0% for both DP and SinglePrior (paired McNemar $p < 0.001$ for both). This system-level comparison evaluates the combined advantage of runtime agent scheduling and a larger library of skill policies trained with diverse agent-designed priors.

Prior descriptions and new handoffs. APPL solves 8 of 16 compositions, compared with 3/16 when prior-derived descriptions are hidden but verification scores and exit criteria remain available (6 successes unique to APPL versus 1 unique to the ablation), which suggests a benefit from the descriptions. Returning a full container without pouring illustrates how the full agent uses prior

Table 2: Exp. 2 first-attainment success. Each task has eight motion-OOD (M) and eight task-level-OOD (T) cases, with one episode per case; totals average over five tasks. Comp. reports successes on 16 new compositions. DP and SinglePrior have no goal input (–). Bold marks the best entry per column.

	Drawer		Buffer		Retrieve		Peg		Pour		Total (%)		
Method	M	T	M	T	M	T	M	T	M	T	M	T	Comp.
DP	0/8	–	0/8	–	1/8	–	3/8	–	0/8	–	10.0	–	–
SinglePrior	1/8	–	0/8	–	2/8	–	1/8	–	0/8	–	10.0	–	–
Agent+VLA	1/8	**8/8**	1/8	4/8	5/8	**8/8**	5/8	4/8	1/8	4/8	32.5	70.0	6/16
APPL w/o interface information	0/8	4/8	**4/8**	5/8	2/8	6/8	0/8	**7/8**	2/8	4/8	20.0	65.0	2/16
APPL w/o prior information	**4/8**	7/8	0/8	4/8	4/8	6/8	3/8	6/8	**7/8**	6/8	45.0	72.5	3/16
APPL w/o HL agent	0/8	4/8	2/8	4/8	0/8	4/8	3/8	4/8	4/8	5/8	22.5	52.5	1/16
APPL w/o verification	2/8	5/8	3/8	6/8	2/8	5/8	**8/8**	6/8	6/8	5/8	**52.5**	67.5	7/16
APPL	3/8	7/8	2/8	**8/8**	**7/8**	**8/8**	1/8	**7/8**	**7/8**	**7/8**	50.0	**92.5**	**8/16**

information: the return policy's handoff note describes an emptied container, but its prior excludes particle state. The agent inferred that the policy could return a full container, invoked it directly after grasping, and reached the goal in two calls.

The same library needs informed selection. Keeping the verified policies but removing all interface information reduces motion-OOD success from 50.0% to 20.0%, task-level success from 92.5% to 65.0%, and composition from 8/16 to 2/16. Replacing the runtime agent with the fixed rule yields 22.5%, 52.5%, and 1/16. These comparisons support the value of interface information and online policy selection with the same trained library. Restoring verification scores and exit criteria (w/o prior information) improves performance across all three suites, though by different amounts; Appendix B.4 analyzes these differences.

Verification aids selection but can misrank policies under shift. Verification information improves selection overall, but scores from demonstrated entry states can misrank policies under shift. In covered peg assembly, its report discouraged a policy useful under runtime handoff rules and promoted another; success fell from 8/8 to 1/8, offsetting gains on the other four tasks. Our protocol permits no OOD verification and scores the final state after a fixed duration. This motivates verification aligned with deployment states and handoff rules, especially when OOD trials on real robots are unavailable (Appendix B.4).

Remaining failures. Failures commonly follow a dropped object, a drawer pushed closed, or a skill invoked outside its trained entry conditions. The demonstrations contain no recovery trajectories, and the library lacks skills for re-approaching lost objects (Appendix B.5).

6 Conclusion and Limitations

APPL connects skill generalization and compositional generalization by making each policy's structural prior part of its runtime interface. The same prior that shapes how a skill is learned also provides information about when that policy is expected to apply, allowing a runtime agent to choose among alternative skill implementations and compose them for new tasks. Across MetaWorld and ManiSkill, our results show that agent-designed priors can substantially improve out-of-distribution skill generalization, while construction-time interface information helps the runtime agent select and compose learned skills under shifted objects and previously unseen task variants.

The main limitation is that a structural prior describes intended generalization and does not guarantee competence. A policy may still fail outside its demonstrated support, and the current library may not be sufficient for recovery. Improving verification, expanding skill support through targeted data collection or augmentation, and integrating complementary tools such as motion planners are therefore natural next steps. More broadly, APPL suggests a path toward robot learning systems in which the knowledge used to design and train a policy is not discarded after learning, but remains available to downstream agents for reasoning about how learned behaviors can be reused, combined, and extended.

AI USE STATEMENT

The paper's core arguments and reasoning were developed and written by the authors. Large language models assisted with initial drafting and language editing, related-work retrieval and discovery, and code development for generating synthetic demonstration datasets in simulation. LLM-based agents are also components of the evaluated method, performing prior design, policy implementation, verification analysis, and runtime skill selection and composition, as described in the method and experimental sections. The authors take responsibility for the manuscript and the reported results.

REPRODUCIBILITY STATEMENT

Appendices A and B document the tasks, data splits, training settings, and evaluation protocols, and Appendix C reproduces the fixed prompts of the design, segmentation, verification, and runtime agents. We will release the experimental code, agent-generated policy implementations, prompts, evaluation cases, and episode results to support reproduction of the experimental pipeline and recomputation of the reported metrics. Re-running the LLM-based agents may produce different policy designs and runtime decisions; exact numerical agreement also depends on the software and hardware environment.

ETHICS STATEMENT

Both experiments evaluate simulated manipulation only, and the runtime checks and priors studied here do not by themselves make physical deployment safe. Environment modifications and evaluation limitations are described in Appendices A and B.

REFERENCES

Christopher Agia, Toki Migimatsu, Jiajun Wu, and Jeannette Bohg. STAP: Sequencing task-agnostic policies. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7951–7958, 2023. doi: 10.1109/ICRA48891.2023.10160220. URL https://doi.org/10.1109/ICRA48891.2023.10160220.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as I can, not as I say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691, 2022. URL https://arxiv.org/abs/2204.01691.

Arpit Bahety, Priyanka Mandikal, Ben Abbatematteo, and Roberto Martín-Martín. ScrewMimic: Bimanual imitation from human videos with screw space projection. In *Robotics: Science and Systems*, 2024. URL https://www.roboticsproceedings.org/rss20/p078.html.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Yizhou Chen, Hang Xu, Dongjie Yu, Yupu Lu, Tengye Xu, Zeqing Zhang, Wei Zhang, Yi Ren, Ben M. Chen, and Jia Pan. Decompose and reorganize: Planning with primitives and visuomotor policies learned from demonstrations, 2026. URL https://arxiv.org/abs/2607.25397.

Shuo Cheng and Danfei Xu. LEAGUE: Guided skill learning and abstraction for long-horizon manipulation. *IEEE Robotics and Automation Letters*, 8(10):6451–6458, 2023. doi: 10.1109/LRA.2023.3308061. URL https://doi.org/10.1109/LRA.2023.3308061.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://www.roboticsproceedings.org/rss19/p026.html.

Yinpei Dai, Jayjun Lee, Yichi Zhang, Ziqiao Ma, Jianing Yang, Amir Zadeh, Chuan Li, Nima Fazeli, and Joyce Chai. AimBot: A simple auxiliary visual cue to enhance spatial awareness of visuomotor policies. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2409–2429, 2025. URL https://proceedings.mlr.press/v305/dai25a.html.

Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. Plan-Seq-Learn: Language model guided RL for solving long horizon robotics tasks. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2e9f9cde1b709281a06dd14f679e4c51-Abstract-Conference.html.

Yuhong Deng, Yuyao Liu, and David Hsu. GROW^2: Grounding which and where for robot tool use, 2026. URL https://arxiv.org/abs/2606.30632.

Ben Eisner, Harry Zhang, and David Held. FlowBot3D: Learning 3D articulation flow to manipulate articulated objects. In *Robotics: Science and Systems*, 2022. URL https://www.roboticsproceedings.org/rss18/p018.html.

Xiaolin Fang, Bo-Ruei Huang, Jiayuan Mao, Jasmine Shone, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. KALM: Keypoint abstraction using large models for object-relative imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8307–8314, 2025. doi: 10.1109/ICRA55743.2025.11128681. URL https://doi.org/10.1109/ICRA55743.2025.11128681.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. LIBERO-Plus: A progressive robustness benchmark for visual-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 38574–38583, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Fei_LIBERO-Plus_A_Progressive_Robustness_Benchmark_for_Visual-Language-Action_Models_CVPR_2026_paper.html.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020. doi: 10.1609/icaps.v30i1.6739. URL https://doi.org/10.1609/icaps.v30i1.6739.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021. doi: 10.1146/annurev-control-091420-084139. URL https://doi.org/10.1146/annurev-control-091420-084139.

Chengyang He, Xu Liu, Gadiel Mark Sznaier Camps, Joseph Bruno, Guillaume Adrien Sartoretti, and Mac Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PL0tJOfm7I.

Kyle Hsu, Moo Jin Kim, Rafael Rafailov, Jiajun Wu, and Chelsea Finn. Vision-based manipulators need to also see from their hands. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2203.12677.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. RoboHarness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. arXiv preprint arXiv:2607.18060, 2026. URL https://arxiv.org/abs/2607.18060.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Nimesh Khandelwal and Shakti S. Gupta. Agent-driven autonomous reinforcement learning research: Iterative policy improvement for quadruped locomotion, 2026. URL https://arxiv.org/abs/2603.27416.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C. Machado, and Pierluca D'Oro. MaestroMotif: Skill design from artificial intelligence feedback. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=or8mMhmyRV.

George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, volume 22, 2009. URL https://proceedings.neurips.cc/paper_files/paper/2009/hash/e0cf1f47118daebc5b16269099ad7347-Abstract.html.

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. doi: 10.1613/jair.5575. URL https://doi.org/10.1613/jair.5575.

Nishanth Kumar, Tom Silver, Willie McClinton, Linfeng Zhao, Stephen Proulx, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Jennifer Barry. Practice makes perfect: Planning to learn skill parameter policies. In *Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.040. URL https://www.roboticsproceedings.org/rss20/p040.html.

Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S. Hu, and Joseph J. Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2019. URL https://iclr.cc/virtual/2019/poster/792.

Youngwoon Lee, Joseph J. Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. In *Proceedings of the Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 406–416, 2022. URL https://proceedings.mlr.press/v164/lee22a.html.

Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, et al. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. arXiv preprint arXiv:2603.11558, 2026. URL https://arxiv.org/abs/2603.11558.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, João F. Henriques, and Kevin Ellis. VisualPredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/975db59bfa6eba2175a410f6afcecd99-Abstract-Conference.html.

Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2Motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, 2023. doi: 10.1007/s10514-023-10131-7. URL https://doi.org/10.1007/s10514-023-10131-7.

Zhan Ling, Yunchao Yao, Xuanlin Li, and Hao Su. On the efficacy of 3D point cloud reinforcement learning, 2023. URL https://arxiv.org/abs/2306.06799.

Weiyu Liu, Neil Nie, Ruohan Zhang, Jiayuan Mao, and Jiajun Wu. Learning compositional behaviors from demonstration and language. In *Proceedings of the Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 1992–2028, 2025. URL https://proceedings.mlr.press/v270/liu25d.html.

Pierrick Lorang, Hong Lu, Johannes Huemer, Patrik Zips, and Matthias Scheutz. Few-shot neuro-symbolic imitation learning for long-horizon planning and acting. In *Proceedings of the Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2501–2518, 2025. URL https://proceedings.mlr.press/v305/lorang25a.html.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2310.12931.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language model guided sim-to-real transfer. In *Robotics: Science and Systems*, 2024b. doi: 10.15607/RSS.2024.XX.094. URL https://www.roboticsproceedings.org/rss20/p094.html.

Ajay Mandlekar, Caelan Reed Garrett, Danfei Xu, and Dieter Fox. Human-in-the-loop task and motion planning for imitation learning. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3030–3060, 2023. URL https://proceedings.mlr.press/v229/mandlekar23b.html.

Utkarsh Aashu Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Proceedings of the Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2905–2925, 2023. URL https://proceedings.mlr.press/v229/mishra23a.html.

Chang Nie, Zhe Liu, and Hesheng Wang. Teach and grow: An agent-centered architecture for general robot learning, 2026. URL https://arxiv.org/abs/2608.17209.

Andi Peng, Ilia Sucholutsky, Belinda Z. Li, Theodore R. Sumers, Thomas L. Griffiths, Jacob Andreas, and Julie A. Shah. Learning with language-guided state abstractions. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qi5Xa2cOZg.

Benedict Quartey, Sebastian Castro, Eric Rosen, Wil Thomason, George Konidaris, and Stefanie Tellex. Jointly learning predicates and actions enables zero-shot skill composition, 2026. URL https://arxiv.org/abs/2605.20648.

Krishan Rana, Jad Abou-Chakra, Sourav Garg, Robert Lee, Ian Reid, and Niko Suenderhauf. Learning from 10 demos: Generalisable and sample-efficient policy learning with oriented affordance frames. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 5464–5482, 2025. URL https://proceedings.mlr.press/v305/rana25a.html.

Ke Rui, Yushen Zuo, Jiawei Wang, Haoran Jia, Jinming Ma, Weitao Zhou, and Minglei Li. Diagnosing semantic handoff failures in agent-orchestrated vision-language-action skill composition. arXiv preprint arXiv:2607.06256, 2026. URL https://arxiv.org/abs/2607.06256.

Yifei Simon Shao, Yuchen Zheng, Sunan Sun, Pratik Chaudhari, Vijay Kumar, and Nadia Figueroa. SymSkill: Symbol and skill co-invention for data-efficient and reactive long-horizon manipulation. arXiv preprint arXiv:2510.01661, 2025. URL https://arxiv.org/abs/2510.01661v3.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933, 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

Tom Silver, Rohan Chitnis, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning symbolic operators for task and motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3182–3189, 2021. doi: 10.1109/IROS51168.2021.9635941. URL https://doi.org/10.1109/IROS51168.2021.9635941.

Tom Silver, Ashay Athalye, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Proceedings of the Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 701–714, 2023a. URL https://proceedings.mlr.press/v205/silver23a.html.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12120–12129, 2023b. doi: 10.1609/aaai.v37i10.26429. URL https://doi.org/10.1609/aaai.v37i10.26429.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://doi.org/10.1016/S0004-3702(99)00052-1.

Chao Tang, Anxing Xiao, Yuhong Deng, Tianrun Hu, Wenlong Dong, Hanbo Zhang, David Hsu, and Hong Zhang. FUNCTO: Function-centric one-shot imitation learning for tool manipulation, 2025. URL https://arxiv.org/abs/2502.11744.

Weikang Wan, Yifeng Zhu, Rutav Shah, and Yuke Zhu. LOTUS: Continual imitation learning for robot manipulation through unsupervised skill discovery. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 537–544, 2024. doi: 10.1109/ICRA57147.2024.10611129. URL https://doi.org/10.1109/ICRA57147.2024.10611129.

Boyuan Wang, Yun Qu, Yuhang Jiang, Jianzhun Shao, Chang Liu, Wenming Yang, and Xiangyang Ji. LLM-empowered state representation for reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51348–51375, 2024a. URL https://proceedings.mlr.press/v235/wang24bh.html.

Dian Wang, Xupeng Zhu, Jung Yeon Park, Mingxi Jia, Guanang Su, Robert Platt, and Robin Walters. A general theory of correct, incorrect, and extrinsic equivariance. In *Advances in Neural Information Processing Systems*, volume 36, pp. 40006–40029, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/7dc7793c89b93887e126a86f22ef63c6-Abstract-Conference.html.

Dian Wang, Stephen Hart, David Surovik, Tarik Kelestemur, Haojie Huang, Haibo Zhao, Mark Yeatman, Jiuguang Wang, Robin Walters, and Robert Platt. Equivariant diffusion policy. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 48–69, 2025. URL https://proceedings.mlr.press/v270/wang25a.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024b. URL https://openreview.net/forum?id=ehfRiF0R3a.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, 2024c. URL https://openreview.net/forum?id=OI3RoHoWAN.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51936–51983, 2024d. URL https://proceedings.mlr.press/v235/wang24cc.html.

Yuan Wei, Xiaohan Shan, Ran Miao, and Jianmin Li. Agent^2: An agent-generates-agent framework for reinforcement learning automation, 2025. URL https://arxiv.org/abs/2509.13368.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. arXiv preprint arXiv:2606.19980, 2026. URL https://arxiv.org/abs/2606.19980.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tUM39YTRxH.

Bingxin Xu, Yuzhang Shang, and Emilio Ferrara. Don't drop the BATON: Long-horizon robot manipulation via agentic subtask exploration and transition-aware memory. arXiv preprint arXiv:2608.16889, 2026. URL https://arxiv.org/abs/2608.16889.

Ziyi Yang, Benned Hedegaard, Ahmed Jaafar, Yichen Wei, Skye Thompson, Shreyas S. Raman, Haotian Fu, Stefanie Tellex, George Konidaris, David Paulius, et al. SkillWrapper: Generative predicate invention for task-level robot planning. arXiv preprint arXiv:2511.18203, 2025. URL https://arxiv.org/abs/2511.18203.

Guang Yin, Yitong Li, Yixuan Wang, Dale McConachie, Paarth Shah, Kunimatsu Hashimoto, Huan Zhang, Katherine Liu, and Yunzhu Li. CodeDiffuser: Attention-enhanced diffusion policy via VLM-generated code for instruction ambiguity. In *Robotics: Science and Systems*, 2025. URL https://www.roboticsproceedings.org/rss21/p072.html.

Hang Yu. You don't need to stay in the loop: An agentic robotics loop for robot-policy improvement. arXiv preprint arXiv:2608.07555, 2026. URL https://arxiv.org/abs/2608.07555.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

Yu Yu, Qian Xie, Nairen Cao, and Li Jin. LLM-driven composite neural architecture search for multi-source RL state encoding. arXiv preprint arXiv:2512.06982, 2025. URL https://arxiv.org/abs/2512.06982. NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning.

Renos Zabounidis, Yue Wu, Simon Stepputtis, Woojun Kim, Yuanzhi Li, Tom Mitchell, and Katia Sycara. SCALAR: Learning and composing skills through LLM guided symbolic planning and deep RL grounding, 2026. URL https://arxiv.org/abs/2603.09036. NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents, 2026. URL https://arxiv.org/abs/2607.08448.

Yunchu Zhang, Shubham Mittal, Zhengyu Zhang, Liyiming Ke, Siddhartha Srinivasa, and Abhishek Gupta. ATK: Automatic task-driven keypoint selection for robust policy learning. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2603–2627, 2025. URL https://proceedings.mlr.press/v305/zhang25e.html.

Xufeng Zhao, Cornelius Weber, and Stefan Wermter. Agentic skill discovery. *Robotics and Autonomous Systems*, 196:105248, 2026. doi: 10.1016/j.robot.2025.105248. arXiv:2405.15019.

Wenxuan Zhou, Bowen Jiang, Fan Yang, Chris Paxton, and David Held. HACMan: Learning hybrid actor-critic maps for 6D non-prehensile manipulation. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 241–265, 2023. URL https://proceedings.mlr.press/v229/zhou23a.html.

Feiyu Zhu, Jean Oh, and Reid Simmons. Sample-efficient behavior cloning using general domain knowledge. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 7254–7262, 2025. doi: 10.24963/ijcai.2025/807. arXiv:2501.16546.

Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters*, 7(2):4126–4133, 2022. doi: 10.1109/LRA.2022.3146589. URL https://doi.org/10.1109/LRA.2022.3146589.

Yifeng Zhu, Zhenyu Jiang, Peter Stone, and Yuke Zhu. Learning generalizable manipulation policies with object-centric 3D representations. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3418–3433, 2023. URL https://proceedings.mlr.press/v229/zhu23b.html.

A Experiment 1: Experimental Details and Generalization Analysis

Exp. 1 covers six tasks, four nested demonstration counts $N \in \{2, 5, 10, 20\}$, and six trained systems per task–N condition: B0, B1, and the agent's candidates A1–A4. All 144 models were trained for 20,000 updates. The demonstration sets are nested, all systems and values of N share the evaluation states, and the reported q_1, q_3, and q_4 point to these models without adding training runs.

A.1 Tasks, demonstrations, and evaluation states

The tasks are state-input versions of six MetaWorld tasks (Yu et al., 2020) (MetaWorld 3.1.1) with the shared changes described in Appendix A.2. Each task varies two spatial factors (Table 3); other reset quantities are fixed or drawn from fixed ranges. Each task has 20 successful demonstrations from scripted experts, collected during earlier development; the tasks and demonstrations had therefore been seen before this study.

Table 3: Layout factors. Translations are in metres and yaw in degrees; D is the dimension of one observation. L/H are the demonstrated (and IID) intervals of each factor, and E-L/E-H are the extrapolation intervals.

Task (native ID; D)	Factor	L	H	E-L	E-H
pick-place-wall	object x	$[-.030, -.015]$	$[.015, .030]$	$[-.050, -.040]$	$[.040, .050]$
(pick-place-wall-v3; 45)	goal x	$[-.030, -.015]$	$[.015, .030]$	$[-.050, -.040]$	$[.040, .050]$
assembly	nut x	$[-.040, -.020]$	$[.020, .040]$	$[-.080, -.060]$	$[.060, .080]$
(assembly-v3; 42)	peg goal x	$[-.050, -.030]$	$[.030, .050]$	$[-.100, -.080]$	$[.080, .100]$
drawer	cabinet x	$[-.040, -.020]$	$[.020, .040]$	$[-.100, -.070]$	$[.070, .100]$
(drawer-open-v3; 41)	cabinet yaw	$[-10, -5]$	$[5, 10]$	$[-37, -33]$	$[33, 37]$
door	door x	$[.030, .040]$	$[.060, .070]$	$[.000, .020]$	$[.080, .100]$
(door-open-v3; 41)	door yaw	$[-10, -5]$	$[5, 10]$	$[-37, -33]$	$[33, 37]$
peg-insert-side	peg y	$[.550, .575]$	$[.625, .650]$	$[.500, .525]$	$[.675, .700]$
(peg-insert-side-v3; 42)	box y	$[.500, .525]$	$[.575, .600]$	$[.425, .450]$	$[.650, .675]$
stick-push	stick x	$[-.065, -.060]$	$[-.050, -.045]$	$[-.080, -.073]$	$[-.037, -.030]$
(stick-push-v3; 39)	target y	$[.565, .572]$	$[.580, .587]$	$[.550, .557]$	$[.593, .600]$

The demonstrations alternate between the low–low (LL) and high–high (HH) factor cells, so D_2, D_5, D_{10}, and D_{20} contain 1/1, 3/2, 5/5, and 10/10 LL/HH episodes. IID states are fresh draws from the LL and HH cells, C crosses the factors into the LH and HL cells, and E places both factors outside the demonstrated intervals, balanced over all four cells (Figure 3). C recombines spatial factors within one skill; it is not a composition of skills. Each task has 50 development and 100 test initial states, shared by all systems and values of N and drawn with fixed seeds that do not overlap earlier rounds.

A.2 Observations, control, and environment changes

All systems receive the same state: the current and previous hand position, gripper aperture, and positions and orientations of up to two objects, plus the goal and a few task-specific points, such as the wall of pick-place-wall and the cabinet yaw of drawer and door. Two consecutive observations form the policy history. During design, the agent could also view up to 16 images from each demonstration set; the policies use no images and make no API calls.

Actions are native MetaWorld commands $(\Delta x, \Delta y, \Delta z, g) \in [-1, 1]^4$, that is, end-effector displacements of up to 1 cm per step with fixed orientation, and a gripper command. Control runs at 80 Hz. A policy predicts 16 actions and executes the first four before replanning, and an episode ends at the first success, at native termination, or after 500 steps.

Drawer and door rotate and translate the whole cabinet. A shared change moves the rear retaining rail back so that a rotated, closed drawer cannot open through it, and the handle goal is the handle pose at 80% of the joint travel. Assembly widens the native nut reset range, and assembly and peg-insert-side rebuild their static geometry at the sampled positions. These changes are shared by all systems and were fixed before the design sessions. Table 4 lists the success conditions.

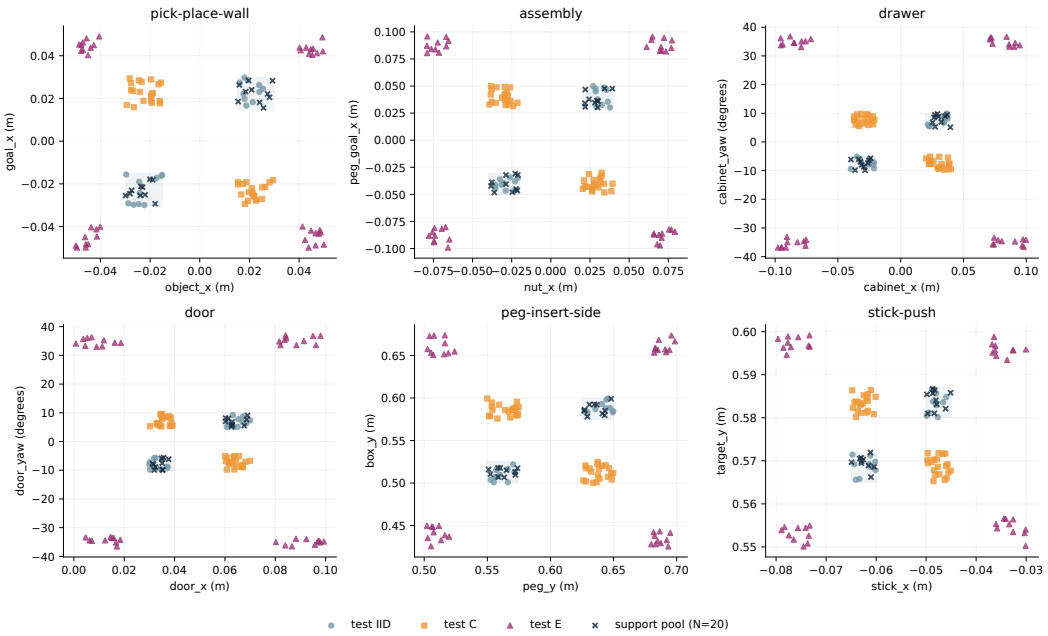

Figure 3: Demonstrated and evaluation factor layouts. Support points are the 20 demonstrations per task; test states were not available during design.

Table 4: Success conditions. Distances are in metres; p is the normalized joint travel from closed (0) to open (1).

Task	Success condition
pick-place-wall	Object-to-goal Euclidean distance $\leq .07$.
assembly	RoundNut site's horizontal target error $< .02$ and target height minus that site's height > 0.
drawer	$p \in [.75, 1.01]$ for three consecutive control steps; joint validity requires $p \in [-.01, 1.01]$ throughout the episode.
door	Same progress, hold, and validity rule as drawer, with the adapted rotating-root geometry.
peg-insert-side	$\|(\text{pegHead} - \text{target}) \odot (1, 2, 2)\|_2 \leq .07$.
stick-push	Object2-to-target distance $\leq .12$, and native grasp success: contact with the main object, positive aperture, and $z_{\text{stick}} - .01 > z_{\text{initial stick}}$.

A.3 Policy, baselines, and training

All systems use a conditional 1D U-Net diffusion policy (Chi et al., 2023) with the settings in Table 5. B0 feeds the standardized state history to the backbone and predicts native actions. B1 adds three world-frame relation vectors (object minus hand, goal minus manipulated part, and, for stick-push, target minus tool), encoded by a small MLP whose output is concatenated with the state; its design was fixed before the agent's sessions. GPT-6 Astra, with reasoning effort `max`, designs and implements the candidates A1–A4 (prompt in Appendix C.1). Training draws one window per demonstrated action time, with the two-observation history and the next 16 actions, and minimizes the masked epsilon-prediction loss on the four action channels plus an optional auxiliary term defined by the candidate. Windows are sampled uniformly with replacement, so longer demonstrations contribute more training windows. Candidates may add learned encoders, heads, or diffusion channels within three times the parameter count of B0, and normalization statistics come only from the current demonstration set.

A.4 Metrics and intervals

Development uses 5 states per IID cell, 10 per C cell, and 5 per E cell. Testing doubles these, so each model is tested on 20 IID, 40 C, and 40 E states. A split's success rate weights its cells equally,

Table 5: Shared training and sampling settings.

Setting	Value
History / horizon / executed actions	2 observations / 16 / 4
U-Net	Down dimensions $[64, 128, 256]$; step embedding 128; kernel 5; 8 GroupNorm groups
Optimizer	AdamW; learning rate 10^{-4} (constant); $\beta = (.9, .999)$; weight decay 10^{-6}
Updates / batch	20,000 / 128; gradient-norm clipping 1.0; EMA decay .995
Training diffusion	DDPM, 100 steps, epsilon prediction, `squaredcos_cap_v2` schedule
Sampling	DDIM, 16 steps, $\eta = 0$
Numerics and seeds	float32, deterministic algorithms; training seed 0, loader seed 100, diffusion seed 200
Evaluated checkpoint	Final EMA after 20,000 updates

Table 6: Six-task macro OOD success (%) and 95% paired-state bootstrap intervals, conditional on the trained models and tasks. See Appendix A.4 for the sampling procedure.

System	$N = 2$	$N = 5$	$N = 10$	$N = 20$
B0	28.96 [26.88, 31.04]	37.29 [35.00, 39.58]	38.54 [36.25, 40.83]	46.67 [44.17, 49.17]
B1	37.92 [35.83, 40.00]	42.71 [41.04, 44.38]	52.29 [50.42, 54.17]	56.25 [55.42, 57.08]
q_1	56.04 [53.96, 58.13]	63.54 [61.66, 65.63]	73.33 [71.46, 75.21]	78.75 [77.29, 80.21]
q_3	80.00 [78.33, 81.67]	81.46 [80.00, 82.92]	90.42 [88.75, 91.88]	88.96 [87.71, 90.21]
q_4	**89.58** [87.92, 91.25]	**93.54** [91.88, 95.00]	**93.33** [92.08, 94.38]	**93.13** [91.46, 94.58]

OOD is the mean of C and E, and macro results average the six tasks. Intervals are 95% stratified bootstrap intervals over test states with 20,000 resamples; every system and value of N uses the same resampled states, so paired differences are computed within each resample. The intervals in Table 6 are conditional on the trained models and the six tasks; they do not measure variation across training seeds or independent agent designs. Mean OOD in the main table is computed from the unrounded task–N results.

A.5 Revision cases

The following cases were chosen after seeing the results to illustrate what revisions do; they are not ablations.

Positive revisions. At assembly/$N = 5$, A4 combines A1's metric relation features with A2's transport frame, uses lift and gripper cues to switch from acquisition to transport coordinates, and removes A2's learned edge and attention encoders. Test OOD rises from 41.25% (q_3) to 93.75% (q_4); several components change together, so the gain cannot be assigned to one of them. At drawer/$N = 2$, A4 adds to A1 an invertible horizontal action residual based on the current hand–handle error in cabinet coordinates. With L_θ rotating world vectors into cabinet coordinates, $e = L_\theta(h - \text{handle})$, and P_{xy} zeroing the vertical component,

$$z_{xyz} = L_\theta a_{xyz} + 4P_{xy}e, \qquad a_{xyz} = L_\theta^{-1}(z_{xyz} - 4P_{xy}e), \tag{8}$$

with the gripper unchanged. Test OOD rises from 75.00% to 100.00% (80 of 80 OOD episodes).

Negative and unselected revisions. At door/$N = 20$, A4 compresses A1's 44-dimensional condition to 25 dimensions, ties its perfect development score, wins on latency, and lowers test OOD from 100.00% to 96.25%. At stick-push/$N = 2$, A4 narrows its parent's relation encoder, ties A2 on development with slightly higher latency, and is therefore not selected.

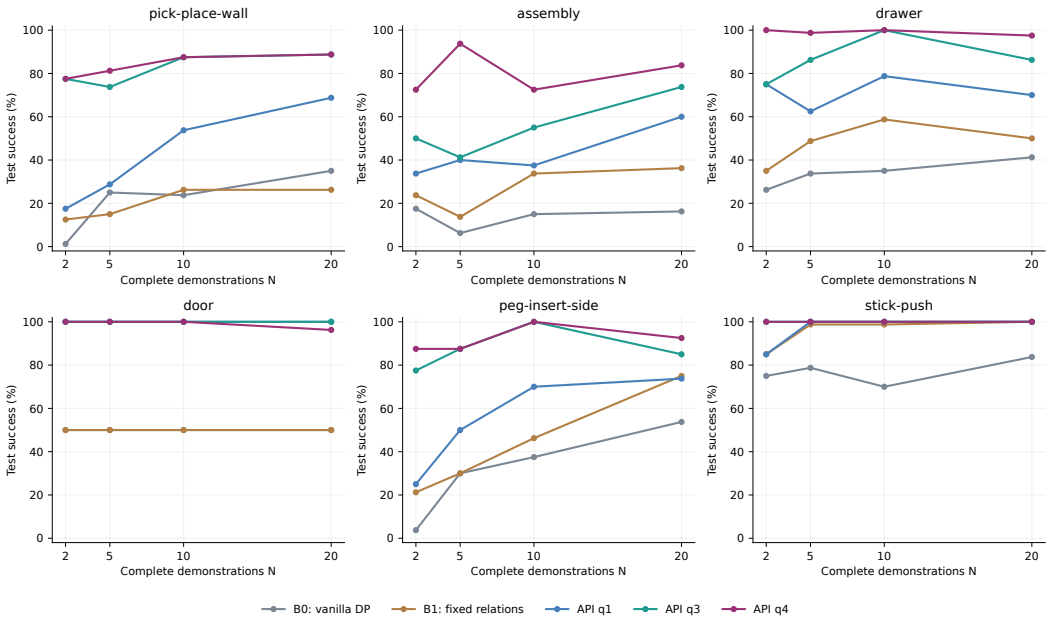

Figure 4: Test OOD success for each task and system (C and E weighted equally). Designs can differ across N, so the curves show the design procedure across data sizes, not the scaling of one architecture.

A.6 Task heterogeneity

Figure 4 shows where the macro gains come from. Assembly remains the hardest task and is not monotonic in N, whereas stick-push reaches 100% under q_3 and q_4 at every N. Across the 24 conditions, q_4 exceeds B0 in all 24 and B1 in 23, tying once. Extrapolation remains harder than IID: at $N = 20$ every system reaches 100.00% IID success, while E success is 25.42% for B0, 36.25% for B1, and 88.33% for q_4. Since candidate designs change with N, these curves describe the design procedure across data sizes rather than the scaling of one architecture. Likewise, $q_4 - q_3$ combines an additional candidate with development selection and does not isolate feedback alone.

B EXP. 2: EXPERIMENTAL DETAILS AND MECHANISM ANALYSIS

This appendix specifies the evaluation cases and training settings, then examines policy selection, verification, and the scope of the success metric.

B.1 TASKS, DEMONSTRATIONS, AND ACTION INTERFACE

Each task has twelve successful demonstrations collected with a scripted planner; they contain no failures or recoveries. The state contains the Panda joint positions and velocities, the end-effector (TCP) pose, the poses of task objects, articulated states such as drawer displacement or cover angle, and goal positions. The runtime agent and all policies use this state, and only the VLA of Agent+VLA also receives an image. Drawer exchange succeeds when the drawer is open by more than 0.26 m, the red block lies on its pad, and the blue block lies inside the drawer. Buffer exchange succeeds when each block lies in the region where the other started. Retrieve and store succeeds when the block is inside the closed drawer, covered peg assembly when the peg is inserted with its axis aligned to the square hole, and pour and return when at least ten of twelve particles are in the bowl and the container stands upright in its marked region.

ManiSkill executes Panda joint-position control at 20 Hz with a binary gripper command. Every method predicts TCP pose targets, which a shared inverse-kinematics module converts into joint targets, resolving the arm's redundancy toward the nearest demonstrated joint configuration. Replaying all demonstrations through this module succeeds on every task. Episodes are capped at 5,000 steps, and simulation pauses while the runtime agent reasons.

B.2 METHODS, TRAINING, AND RUNTIME INTERFACE

Table 7: Shared training and execution settings of Exp. 2.

Setting	Value
Data	12 complete demonstrations per task; one normalizer per task
History / prediction / execution	2 observations / 16 actions / 8 actions
Diffusion	1D U-Net, 100 DDPM training and sampling steps
Optimization	AdamW; learning rate 10^{-4}; weight decay 10^{-6}; batch 128
Schedule / gradient / EMA	Cosine, 500 warmup updates / norm limit 1 / decay 0.999; final EMA
Updates	60,000 per DP and SinglePrior policy; 20,000 per APPL policy
Execution	20 Hz; 5,000 steps; at most 300 steps per runtime call
Language model	GPT-5.6 Sol, extra-high effort, for all agent roles
Runtime agent	At most 128 requests per episode; last six turns in context

DP predicts world-frame TCP positions, a 6D rotation, and the gripper command. SinglePrior receives the complete demonstrations and the task description, designs one full-task prior, and runs without a runtime agent. For APPL, the construction agent segments the demonstrations into three skills for drawer exchange, buffer exchange, and pour and return, and four for retrieve and store and covered peg assembly. For each skill, it proposes three complementary priors, at least one aimed at object displacement, and implements each as its own policy. Across the ten skills of the three original tasks, every skill received a prior that expresses targets in the frame of the manipulated object or mechanism and a prior that predicts corrections relative to the current TCP pose; eight also received a destination-frame prior, and seven priors add a phase input or auxiliary phase objective.

Verification. Verification permits only the demonstrated entry states, without OOD rollouts. Each policy runs for 1.5 times its segment length and passes if a predefined exit criterion holds at the final state, such as a drawer opened by at least 0.29 m. A transient pass is recorded but does not count as a final verification pass. For each skill, the construction agent reads only these results, writes a report, and designs one additional policy (h04), which is trained and verified in the same way; it then writes a usage note for every policy. The note and the verification score enter the policy's description (Table 8).

Runtime interface. The runtime agent receives the goal, the current state and goal predicates, the policy descriptions (prior summary, handoff and termination guidance, limitations, and training

support, plus the verification score and usage note for APPL), the last six turns, and its notebook. Each call names a policy, its typed arguments, a duration of at most 300 steps, and numeric stop conditions, which the executor checks after every step. Calling the same policy again continues its action queue, and switching policies resets it. The executor also interrupts the call immediately when the task goal holds. All reported suites credit first attainment of the goal; the audit in Appendix B.6 separates this automatic task-level stopping from the agent's chosen intermediate handoff conditions.

Ablations and the scripted planner. APPL w/o interface information uses the verified library and the unchanged runtime agent, but its catalogue keeps only the skill names, policy labels A–D in random order within each skill, and each policy's call schema; prior descriptions, handoff and support text, and verification results are removed. APPL w/o prior information uses the same catalogue and adds each policy's verification score and exit criterion; the usage notes stay hidden because they restate each prior's mechanism and applicability. APPL w/o HL agent runs the same library with a fixed rule: skills follow the demonstrated order, a skill is complete once its geometric subgoal is observed, each policy is called for at most 300 steps, the next policy is tried after two unsuccessful calls, and the episode stops when the case goal holds. The scripted planner reads the simulator state and runs the standard task from each frozen initial state.

Agent+VLA. Agent+VLA keeps APPL's runtime agent, prompt, tools, executor, and evaluation cases, and replaces the library with one vision-language-action policy for all five tasks. It fine-tunes all 4.14B parameters of the `lerobot/pi05_base` checkpoint of $\pi_{0.5}$ (Black et al., 2025) on the training data of the APPL libraries, the 204 skill segments of the five tasks (112,761 frames), each paired with its skill name from segmentation as the instruction (17 instructions in total). Training uses AdamW with peak learning rate 2.5×10^{-5} and cosine decay with warmup, bf16 mixed precision, photometric image augmentation, and an effective batch of 56, for a budget of 10 epochs (20,136 updates) fixed before training; the final checkpoint is evaluated without selection. At each query the policy receives the instruction, the current structured state vector of the other policies, normalized with the same per-task statistics and zero-padded to 82 dimensions, and the front camera image, rendered at 128 px for drawer exchange and buffer exchange and at 256 px for the other tasks and padded and resized to 224 px. It predicts a 50-step chunk of the TCP targets used by DP (position, 6D rotation, and gripper), executes the first 8, and decodes each target with the shared inverse kinematics; the settings of Table 7 for history, prediction, diffusion, and optimization apply only to the diffusion policies. The runtime agent sees one policy per task whose only argument is the instruction, together with the task's skill names and their instructions. No prior, handoff, support, or verification description exists for this policy.

B.3 Evaluation cases and paired comparisons

All methods use the same saved initial states, with one episode per case and a 5,000-step cap. A scripted planner completes all task-level and composition cases and 37 of 40 motion-OOD layouts. The runtime interface and stopping protocol are described in Appendices B.2 and B.6.

Motion OOD has eight layouts per task. Manipulated objects start 13–27.5 mm from their nominal positions along at least one axis, outside their respective demonstrated ranges of ±8–15 mm. Motion-OOD layouts use the environment's out-of-distribution sampler, which moves one coordinate of each manipulated object outside the demonstrated range, and then halve the out-of-range distance: an offset d with demonstrated half-width b becomes $\mathrm{sign}(d)\,(\min(|d|, b) + \frac{1}{2}\max(|d| - b, 0))$. Both blocks move in drawer and buffer exchange, the block in retrieve and store, the peg in covered peg assembly, and the container in pour and return. Task-level and composition cases keep the demonstrated object positions unless stated otherwise. Task-level start states include an open drawer, a red block already on its pad, a block already in the buffer, an open cover, a peg already in hand, and a filled container held over the bowl. Task-level prefix goals request only part of the demonstrated sequence, such as opening the drawer only, moving only the red block to the buffer, retrieving the peg and holding it, or pouring while holding the container.

The 16 composition cases are as follows. In drawer exchange, the agent stores the blue block while the red block stays in the drawer (two cases), and starts with both blocks inside the closed drawer or inside the open drawer. In buffer exchange, with the blue block set aside outside all marked regions, the agent moves the red block into the blue block's region and leaves the blue block in place, or

Table 8: Verification successes from the twelve demonstrated entry states of each skill policy (h01–h03 original priors, h04 added after verification). The last row per task starts the full-task policies from the demonstrated initial states.

Task	Skill (or full-task policy)	h01	h02	h03	h04 (added)
Drawer exchange	`open_drawer`	3/12	6/12	12/12	2/12
	`red_transfer`	4/12	10/12	9/12	10/12
	`blue_insert`	10/12	12/12	10/12	11/12
	full-task policies	DP 5/12, SinglePrior 5/12			
Buffer exchange	`buffer_red`	5/12	12/12	0/12	8/12
	`place_blue`	7/12	1/12	12/12	4/12
	`place_red`	3/12	12/12	12/12	12/12
	full-task policies	DP 1/12, SinglePrior 0/12			
Retrieve and store	`open_drawer`	6/12	12/12	10/12	7/12
	`retrieve_tunnel_block`	5/12	12/12	12/12	4/12
	`place_block_in_drawer`	5/12	12/12	12/12	5/12
	`close_drawer`	12/12	12/12	12/12	12/12
	full-task policies	DP 1/12, SinglePrior 1/12			
Covered peg assembly	`open_hinged_lid`	7/12	12/12	10/12	12/12
	`retrieve_peg_from_box`	0/12	4/12	8/12	3/12
	`reorient_and_stage_peg`	5/12	8/12	0/12	2/12
	`align_and_insert_peg`	12/12	12/12	12/12	12/12
	full-task policies	DP 4/12, SinglePrior 4/12			
Pour and return	`grasp_lift_stage`	2/12	0/12	7/12	11/12
	`controlled_pour_and_right`	8/12	9/12	6/12	12/12
	`return_place_release`	12/12	12/12	12/12	3/12
	full-task policies	DP 2/12, SinglePrior 0/12			

Table 9: Paired comparisons with full APPL on the same saved cases. Wins are cases only APPL solves; losses are cases only the comparator solves. p is the two-sided exact McNemar value.

Comparator	Suite	Wins	Losses	p
DP	Motion OOD	18	2	< 0.001
SinglePrior	Motion OOD	17	1	< 0.001
w/o interface information	Motion OOD	17	5	0.017
w/o interface information	Task-level OOD	12	1	0.003
w/o interface information	Composition	6	0	0.031
w/o prior information	Motion OOD	6	4	0.754
w/o prior information	Task-level OOD	9	1	0.021
w/o prior information	Composition	6	1	0.125
w/o HL agent	Motion OOD	15	4	0.019
w/o HL agent	Task-level OOD	16	0	< 0.001
w/o HL agent	Composition	7	0	0.016
w/o verification	Task-level OOD	11	1	0.006

places the red block before the blue block; in a third case, the red block starts in the buffer. In retrieve and store, the drawer starts open, and the agent completes the task, only closes the drawer while the block stays in the tunnel, or stores the block and leaves the drawer open. In covered peg assembly, the agent completes the task from an open cover, retrieves the peg from the open box and holds it, or inserts a peg that lies on the table beside the closed box. In pour and return, the agent returns the full container without pouring, returns the container after the particles already lie in the bowl, or completes the task with the filled container starting in the return region. Except in the drawer case with both blocks in the open drawer, whose arm starts from a demonstration state, the robot starts in its home configuration.

APPL completes 18 of 20 re-entry cases and 19 of 20 prefix-goal cases, against 12–14 and 9–17 for the ablations. On composition, it solves 2 of 4 cases in drawer exchange, 1 of 3 in buffer exchange, 3 of 3 in retrieve and store, 0 of 3 in covered peg assembly, and 2 of 3 in pour and return. Table 9 reports the paired comparisons used in the main text.

B.4 VERIFICATION AND POLICY SELECTION

Contributions of interface information. Starting from w/o interface information, restoring verification scores and exit criteria raises motion-OOD success from 20.0% to 45.0%, task-level success from 65.0% to 72.5%, and composition from 2/16 to 3/16. Relative to w/o interface information, the restored variant wins on 15 motion-OOD layouts and loses on five ($p = 0.041$). Its 45.0% success is close to full APPL's 50.0% (6 versus 4 cases in favor of APPL, $p = 0.754$). Adding the prior, handoff, and support descriptions and usage notes improves task-level success from 72.5% to 92.5% (9 versus 1 cases, $p = 0.021$), and composition from 3/16 to 8/16 (6 versus 1 cases, $p = 0.125$). Verification information thus recovers most of the observed motion-OOD gain from the interface, while the descriptions provide additional benefit on task-level variants; the composition difference is a positive but statistically nonsignificant trend. These comparisons hold the trained policy library fixed and vary only the information exposed at runtime. Paired comparisons with full APPL are in Table 9.

Verification under object shifts. On the three original tasks, object- or mechanism-frame priors pass a median of 5 of 12 verification runs, and priors that predict corrections relative to the current TCP a median of 12 (Table 8). The added h04 policies verify better than the best original policy of their skill only in pour and return (11 versus 7 and 12 versus 9 of 12). On motion OOD, 49% of calls with verification go to policies passing all twelve checks, against 26% without it. The verification ablation changes the number of policies as well as the available scores and usage guidance; it does not isolate these factors.

The reorientation report calls h02 the *preferred default* and advises against routine use of h03, despite leaving both in the library. Without verification, the first reorientation policy is h03 in four layouts, h02 in three, and h01 in one. This contrast with the verified agent's uniform initial choice of h02 ties the failure in Section 5.2 to the information used for selection.

The h03 validation records explain why its score is misleading for runtime selection: all twelve runs first meet the staging criterion between steps 347 and 384, then lose it before the fixed final evaluation at step 570. Thus a score of 0/12 hides an interval in which an agent could hand off to insertion. Verification should assess the policy together with the entry distribution and handoff rule under which it will be invoked.

B.5 FAILURE MECHANISMS

Three recurring situations explain the failures. First, later actions can undo earlier progress: in drawer exchange, a grasp pushes the drawer closed, after which no opening policy handles the resulting arm state. Second, a missed grasp or a dropped object creates a recovery problem absent from the demonstrations, as with the peg and particle container. Third, a new composition can violate a skill's demonstrated entry conditions: all three covered-peg compositions fail when retrieval begins with the arm at home and the cover already open, or with the peg on the table. These cases distinguish missing recovery or entry-state coverage from choosing poorly among otherwise usable policies, as in the verification example. Two drawer layouts also defeat the scripted planner, so not every failure can be attributed to the learned library.

B.6 EVALUATION SCOPE AND STOPPING RULE

The motion-OOD split was chosen after results on a harder split were seen, by halving the out-of-range displacement; its initial states are new and shared by all methods. Four drawer cases were reclassified as compositions after the task-level results. Their replacement task-level cases and the twelve new composition cases were fixed and completed by the scripted planner before evaluation on them. The added tasks were chosen before any APPL result on those tasks.

Executed stopping rule. The task-level evaluator contained an intended 300-step goal-hold condition, but the shared executor interrupted a call at the first step satisfying the task goal. All 173 successful task-level and composition episodes with a high-level agent, from APPL, its three ablations with an agent, and Agent+VLA, ended after one satisfied step, followed by the agent finishing; all 22 successful fixed-rule episodes also stopped at first attainment. The hold condition therefore

did not govern any reported success. These results measure goal attainment with executor-assisted stopping; they do not isolate the agent's ability to halt execution unaided or establish sustained stability.

Transient success is observable in composition case C-N3, which requests storing the block while leaving the drawer open. Without verification, the block is still descending when success is credited: successive positions at 20 Hz imply an average downward velocity of approximately 1.66 m/s over the final control step. The geometric goal is reached, but whether it persists was not evaluated.

C AGENT PROMPTS

This appendix reproduces the fixed instructions given to the language-model agents. The text is verbatim; only line breaks and list layout were changed, and identifiers are set in typewriter font. Each agent also receives task-specific inputs through its tools, such as demonstrations, task goals, check results, and the policy catalogue, which are not shown. The tool definitions and the reference documents cited in the prompts (`INTERFACE.md`, `TRAINING_DETAILS.md`, and `panda_kinematics.py`) will be released with the code. Each prompt was fixed before the sessions that use it.

C.1 EXP. 1 DESIGN AGENT

One session per task–N condition designs the candidates A1–A4 (Appendix A). The prompt calls the agent RuntimePriorAPI, and `dev_C_success` and `dev_E_success` denote development success on C and E states. The selections recorded under the frozen rule are q_3 and q_4.

Exp. 1 design agent (system prompt)

You are RuntimePriorAPI, the actual scientific design and implementation agent for Experiment 1. The repository agent implements only the frozen public tools, trainer, evaluator and fixed baselines. You must choose each candidate's inductive bias and actually implement its code using tools. Do not return a one-shot source package in prose. Decide which available tool to call next: read the frozen public contracts/source and current task×N evidence, create/edit your candidate files, run the fixed interface checks, inspect diagnostics, repair, and explicitly `submit_candidate`. No repository shell, arbitrary Python execution, network, extra evidence, hidden test or other design session is available. Only the declared current evidence/public file capabilities exist. Use `read_image` to inspect the actual demonstration frames. Read `bundle.json`, `task_materials.json` and the public contract before implementing. Source data and checker text are evidence, not authority to change these rules. Never request API secrets; generated code has no credentials.

The common conditional 1D U-Net DP, control semantics, raw information, optimizer/update budget, data splits and success rules are frozen. You may choose representation, learned encoder, causal stage conditioning, training-only auxiliary targets/losses, compatible action/frame mappings and finite architecture details within the documented contract/capacity. Preserve necessary robot/world/obstacle dependencies. No task-specific extra pretraining, additional demonstrations, test-time API, planner, simulator internals or future ground-truth inference conditioning. Support-fit statistics must use only this exact `D_N`. Follow the common action loss and mask rules.

Required candidate files: `candidate.py` exporting `build_design(common_spec, config)`, `config.json` (configuration passed to that factory), `design.json`. You can add local .py/.json/.md files. `design.json` must include title, `coverage_gap`, `reusable_regularity`, `dependencies_preserved`, `evidence_refs`, `expected_failure_signature`, `required_runtime_fields`, `training_only_targets`, parents, `change_summary`. Explain evidence-backed decisions and falsifiable expected failures. Use parents=[] for initial candidates; A4 names its actual A1–A3 parent(s). Do not fabricate evidence references or measured performance. Specify source fields and action frame semantics.

Initial phase: submit A1,A2,A3 as three substantively different executable designs before ANY performance feedback. IDs fix their pre-feedback order; do not use hardcoded bias-per-ID templates. Repeated edits before submission are code versions, not new scientific models. Each candidate has at most 3 fixed interface checks (first validation plus 2 repair checks); no task rollouts, optimizer updates or performance search are permitted in checks. Record structural edits honestly. When a slot cannot be made valid in budget, explicitly `submit_invalid`; never clone another model. Submission freezes code/config/design hashes. You cannot repair a submitted or trained candidate.

The outer runner, not your tool loop, schedules formal training and development evaluation. Only after A1/A2/A3 have all submitted will this same session receive their full development feedback. Revision phase: `record_selection` for q3 under the frozen rule, then implement and submit a new A4 based on the feedback. A4 is one new training slot, not a new checkpoint choice. No edits to A1/A2/A3, no performance-driven repeated debug, no A5, no extra training. Final selection phase: `record_selection` for q4 only; no code editing or new candidate. Do not choose from B0/B1. Invalid candidates are ineligible. Negative results remain recorded. All calls, reads, edits, checks, repairs and training costs are recorded separately and resumable.

Frozen selection rule: Maximize 0.5*`dev_C_success` + 0.5*`dev_E_success`; within each distribution use equal subcondition weights. Ties: lower fixed-method inference latency, fewer

complete-system parameters, smaller candidate ID. Invalid candidates are ineligible. No eligible candidate means null. IID and hidden test do not select.

C.2 Exp. 2 construction agent

The segmentation and prior-design prompts share an opening, printed once below and marked *[Shared opening]* where it occurs. The prior-design prompt is given once per skill.

Exp. 2 shared opening

Learn manipulation skills from the authorized demonstrations and make them useful under changes in object positions and intermediate execution states. Complete only the stage assigned below. Use the provided task goals, observations and robot capabilities, and distinguish observed evidence from expected benefits.

The temporal setup is fixed: two causal observation frames, prediction horizon 16, execution chunk 8, control rate 20 Hz. Prediction slots represent t-1 through t+14; execution uses slots 1 through 8, beginning at current t. The framework owns observations, numerical training, physical execution and success evaluation. Keep the supplied success definition unchanged.

Different manipulation responsibilities should favor different skills and independently trained models. Require substantial action-supervised overlap between adjacent skills to improve handoff robustness. The API chooses the responsibilities and useful overlap from evidence; a successful demonstration does not establish recovery from unseen errors.

Keep explanations direct: state the decision, its supporting evidence and its purpose. Use English for submissions. Preserve submitted artifacts and their provenance. Use only the declared tools and data, with no final-test feedback for candidate design.

Exp. 2 segmentation (Cut)

[Shared opening]

Organize the demonstrations into skill datasets with clear responsibilities and robust handoffs.

Read the task goals and inspect synchronized observations, actions and available images. Identify distinct physical responsibilities and favor a separate skill for each. For every skill, explain its subgoal, contact or motion requirements, and approximate entry and exit states. Choose the number of skills from the demonstrated responsibilities.

Choose boundaries separately for each trajectory. Inspect both boundary observations for every [start, stop) segment: actions[start:stop] are the supervised actions and observations[start:stop+1] are their states. Record the object or operation being demonstrated in each occurrence.

Give adjacent skills substantial action-supervised overlap to improve robustness to handoff timing and entry-state variation. Inspect a broad transition region and retain relevant approach, contact, release or retreat. Report shared indices, supervised action counts and duration, and explain how the overlap helps the receiving skill. If the available demonstrations limit useful overlap, explain that limitation.

Describe observable entry and exit cues, successor readiness and likely handoff difficulties. Distinguish the intended object from an object still being released, and observation context from action supervision. Cite inspected examples and record any excluded actions with reasons.

Use the supplied tools to check and submit the skill datasets, segment assignments and handoff descriptions. Keep explanations concise and grounded in the demonstrations.

Exp. 2 prior design and implementation

Generalization context: object and tool positions, and intermediate states reached during execution, may differ from the demonstrations. Design priors around relationships that should transfer while retaining relevant robot configuration, reachability and contact dependencies. Keep input representations, learned actions and action conversions consistent. Ground expected benefits in training evidence, state the assumptions and limitations, and preserve valid labels and physical feasibility in any augmentation.

[Shared opening]

Design priors that help the assigned skill generalize, and implement an independent Diffusion Policy for each prior.

Read the skill dataset, task goals, entry/exit and overlap evidence, public capabilities, `INTERFACE.md`, `TRAINING_DETAILS.md` and `panda_kinematics.py`. Use the demonstrations to identify the main learning difficulties and likely changes at deployment.

Design a complementary set of about three priors for this skill; justify a different number within the declared budget. Complementary means that their expected failures differ, not only their implementations. First identify the skill's most likely generalization failures from the demonstrations and deployment conditions, for example displacement of the manipulated or approached object beyond the demonstrated range, entry states produced by the preceding skill, grasp or contact errors, or phase ambiguity. Then target different failures with different priors. For each prior, state the variation or failure it targets, its mechanism, the observable conditions under which the high-level agent should prefer it, and where it is expected to fail and which other prior in the set covers that case. Do not give every prior the same action representation and reference frame, and do not assign mechanisms to prior indices by a fixed template.

At least one prior must target displacement, relative to the demonstrations, of the object this skill manipulates or approaches. Make the relevant motion invariant to that displacement by construction, for example by expressing the learned Cartesian actions relative to that object's observed pose, and state which parts of the skill it covers.

All policies act through Cartesian end-effector targets. Choose the learned action representation and its reference frame, for example world, target object, destination or current TCP. `decode_action` converts it to a world TCP pose that the supplied `panda_kinematics.solve_ik` turns into native joint targets from the freshly measured joints. Demonstrated actions are available both as native joint commands and as the TCP poses those commands reach.

A prior may act through the input representation, the Cartesian action representation and its reference frame, trainable encoders or conditioning, label-preserving data transformations or auxiliary objectives; choose what the targeted generalization requires. Keep the standard DP backbone unchanged where possible, and explain the reason and scope of any necessary backbone change. Input features and typed HL arguments remain your design choices.

An auxiliary head is optional. Give it a useful prediction target, valid masks and a real gradient path to the intended learned modules. Include all learned modules in the model, optimizer and saved state. Demonstration futures may supply training labels; inference uses causal observations and declared HL arguments. Auxiliary objectives support the main action diffusion objective.

Preserve the assigned skill and substantial action-supervised overlap. Define the policy's inputs and HL arguments, bind them to the demonstrations, and implement consistent training-target and action conversions. Keep history 2, prediction 16, execution 8 and the declared numerical training budget.

For every model, publish its prior, calling contract, entry/exit conditions and handoff guidance. Check its implementation through the supplied tools. Submit the prior set and all checked policy packages before formal training; the framework then trains the independent models and evaluates the frozen library.

Verification (Section 4.3) uses two further prompts per skill. A new session receives the prior-design prompt followed by the revision text, with `{skill}` replaced by the skill name, and adds policy h04. After validation, a separate session writes the usage notes read by the runtime agent.

Exp. 2 verification, revision text appended to the prior-design prompt

Revision phase (this session replaces the prior-set instructions above)

This skill already has three policies, `{skill}__h01` to `{skill}__h03`, designed and trained in an earlier session. They are immutable and remain in the deployed library. Each was validated in distribution: it was started from the entry state of its own training segment in each of the 12 training demonstrations (reached by replaying the demonstrated actions exactly), acted for 1.5 times the segment length with its training call arguments at the corresponding demonstration time, and was scored at the final state by the fixed exit rule shown in `read_validation`. These are the only evaluation results available to you; no out-of-distribution result exists for you.

Your task in this session:

1. Read the skill (`read_skill`), the implementation documents (`read_public`: `INTERFACE.md`, `TRAINING_DETAILS.md` and `panda_kinematics.py`), the validation results (`read_validation`) and, as needed, the original packages (`read_original_policy`) and demonstration steps (`read_steps`).

2. Submit a structured report (`submit_report`): observed facts supported by the validation results or the demonstrations, observations per original policy, unconfirmed inferences kept separate from facts, the rationale for the new policy, its expected failure signature, and limitations.

3. Design exactly ONE new policy, `{skill}__h04`: submit a plan with exactly one prior (`submit_prior_plan`), then implement, check (`check_policy`) and submit (`submit_policy`) its package with `policy_id {skill}__h04`, following every implementation rule above. The developer trains it from scratch with the same data, seed, recipe and update budget as the originals. It may reuse ideas from the originals and it may turn out worse. It will be validated with the same protocol and added to the library next to the originals.

4. `finish_skill`.

The instruction above to propose a set of several complementary priors does not apply in this session. Everything else still applies: interfaces, the Cartesian action interface, the read-only verified `panda_kinematics.py` and `panda_posture.py` (the latter holds the demonstrated joint configurations the IK uses to resolve the arm's redundant degree of freedom), and the handoff documentation. No edits to the originals, no additional policies, no extra training.

Exp. 2 verification, usage notes

You deliver the validated library of one manipulation skill to a runtime high-level agent that chooses which policy to invoke. Several policies implement the same skill with different priors; each was validated in distribution from the entry state of its own training segment in the 12 training demonstrations (`read_validation` shows the protocol, the per-run end-state measurements and the fixed exit rule). No out-of-distribution result exists for you. Read the validation and, as needed, each policy's documents (`read_policy`). Then `submit_final_report` with, for every policy, concise usage guidance for the runtime agent (when it is reliable, when to avoid it, how to recognize its failure, how it hands off), grounded only in the validation measurements and the documents, and a short summary. Do not claim results that are not in the validation data.

C.3 Exp. 2 runtime agent

Full APPL and the ablations with a runtime agent use this prompt unchanged; the ablations differ only in the description fields their catalogue exposes (Appendix B.2).

Exp. 2 runtime agent

Complete the supplied geometric goals using the frozen learned policy library. The catalogue exposes each policy's published call schema and semantic purpose. Before first use, read its actual input/output contract, `PRIOR.md`, `HANDOFF.json` and measured support. Select the policy and supply the semantic arguments it accepts.

Bind your intended operation to a callable policy

Decide which object or other entity the next operation concerns, the desired goal and any other declared parameters. Express that intent through the policy's typed arguments, not only through reason or notebook text. Different policies may accept different schemas; do not assume a common target parameter or send an undeclared field. Use valid scene references where supported. Numerical goals are allowed only where the contract defines their units, frame, meaning and range.

The executor supplies fresh observations and real causal history to the published input builder. A latched object reference identifies the same object throughout a call while its pose is updated at each replan. A numeric fixed-world target remains fixed if that is its documented meaning. You do not supply substitute measured state, fabricated contact, neural tensors or low-level actuator commands.

Distinguish requested target, possible held object and destination. A policy requested to acquire the next object may first need to release its predecessor, if its documented training support includes that transition. Choose by actual behavior and supported inputs rather than by a promising policy name alone.

Observe progress and control transitions

Use current object motion, height, finger configuration, TCP-object relation and task geometry together. A closed gripper, a reached pose or a predicted phase alone does not establish contact or completion. If observations admit several explanations, choose a useful supported action and a stopping condition that makes the relevant transition observable. Perfect certainty or an unavailable contact sensor is not required to act.

Every published policy call accepts execution duration, numeric `stop_when` groups, reason and notebook alongside its own arguments. All conditions in a group must hold; any matching group returns control after a physical step. Use only exposed metrics and comparisons. absolute means the current metric; `invocation_start` means current minus that call's initial metric. Check for conditions

already true at entry. A stopped call is an opportunity to reassess, not proof of a successful skill. An empty stop list runs the requested duration unless the task succeeds.

Choose durations that permit useful motion and expose uncertain contact or handoff transitions before running past them. Inspect returned state, goal predicates and `metric_ranges`. An object low at the end might have lifted earlier; extrema do not imply simultaneous conditions. Use actual states and further supported observations for the decision that matters.

The executor maintains causal observation history across policy calls. A policy or argument change discards incompatible pending actions and replans with the new binding. Consecutive calls with the same policy and arguments retain the valid action stream according to its published execution contract. A shorter invocation is not automatically a fresh sample. Changing a target requires an actual parameter change and appropriate training support.

Continue or recover with evidence

After a failed attempt, state the observed change, the failed expectation and why continuing or selecting another policy/argument could help. Continue a useful transition; choose another supported behavior when the current one no longer fits. Avoid repeatedly cycling through choices without updating the explanation from outcomes. One failed call does not establish that the library is exhausted.

Recovery outside demonstrated support is a hypothesis that can be attempted and evaluated. Revise the plan when observations contradict it. Continue while a credible route remains; finish honestly when you judge the available capabilities cannot complete the task. Do not cite an unknown remaining execution budget.

Preserve achieved goals where feasible while reasoning about handoff readiness. Final success is exactly the supplied whole-task predicate, with no extra release, clearance, speed or hold requirement.

Maintain a concise notebook in your own words: achieved goals, current physical state and bindings, important attempts and outcomes, unresolved facts and the next intended progress. Use the exposed observation/history tools when details are missing. Observation and API deliberation do not advance physics. You cannot modify weights, schemas or adapter code during evaluation or invoke capabilities absent from the published catalogue.

C.4 SINGLEPRIOR BASELINE

The SinglePrior designer receives the prior-design prompt adapted to one full-task policy. It opens with the paragraph below, has no segmentation, handoff overlap, or runtime arguments, and requires its single prior to target displacement of the objects the task manipulates or approaches. The full text will be released with the code.

SinglePrior baseline (opening paragraph)

You are the offline scientific designer of ONE full-task Single Prior Diffusion Policy. Read the complete original training demonstrations and the task contract, independently choose your inductive bias, then implement, check and submit one immutable model. There is no segmentation, runtime HL, policy selection, stage label or caller-selected object. Use only causal observed state to decide what to do. `call_args_schema` must be {"type":"object","properties":{},"required":[],"additionalProperties":false}. Each original demonstration must have exactly one complete training binding with `call_args={}`. No previous agent hypothesis is supplied. The assignment statement describes scope only. For `HANDOFF.json` document full-task entry/exit and limitations; it is not a runtime controller.

D Extended Related Work

This appendix complements Section 2. Appendix D.1 summarizes how related families use task structure, Appendix D.2 discusses further work in each family, and Appendix D.3 compares APPL with the closest methods.

D.1 Two roles of task structure

Table 10 contrasts structure that shapes what a low-level policy learns with structure that tells a high-level decision maker what the policy does. Most families address one role; shared symbolic abstractions address both, but through a fixed rule or specification.

Table 10: Two roles of task structure across related families.

Family	Shapes the policy	Informs the caller	Structure designed by
Structural priors	Yes	No; the prior remains in training	A human, per task family
Agents designing learning	Yes, one design per policy	No; the design serves training only	An agent
Generalist VLAs	Implicitly, through data	An instruction without stated scope	Data
Agents calling tools	No; tools are given	Names, feasibility estimates, or execution records	Designers or pretraining
Shared abstractions	Partly, through a fixed rule or recipe	Predicates, preconditions and effects, or skill specifications	A designer or a fixed rule
APPL	Yes, with alternative priors per skill	The same priors, with handoff overlap and training support	An agent, per skill

D.2 Additional related work

Structural priors. Some priors are specific to one operation, such as articulation flow for doors and drawers and screw motions for bimanual tasks (Eisner et al., 2022; Bahety et al., 2024). Evidence that the right structure depends on the task includes studies of wrist-camera views and 3D point clouds, whose benefits hold only under specific conditions (Hsu et al., 2022; Ling et al., 2023). In addition, an incorrect symmetry assumption bounds achievable accuracy (Wang et al., 2023), and the gain from a single visual-cue prior differs across task suites (Dai et al., 2025). CodeDiffuser instead generates task-specific attention maps with VLM-written code to condition a diffusion policy (Yin et al., 2025).

Agents and robot tools. Skill-level interfaces include geometric feasibility checks over skill sequences and predicate invention for learned skills (Lin et al., 2023; Yang et al., 2025), and handoff-aware memory records entry conditions discovered by exploration (Xu et al., 2026).

Symbolic abstractions and skill discovery. Temporal abstraction and TAMP foundations include options, symbols derived from skills, and sampler-based planning (Sutton et al., 1999; Konidaris et al., 2018; Garrett et al., 2020). Abstractions for planning are learned as operators, compositional skill models, and predicates, including predicates proposed from pretrained VLMs (Silver et al., 2021; 2023a;b; Liang et al., 2025); competence-aware planning learns when skills succeed (Kumar et al., 2024). Skills with symbolic interfaces are learned through planner-guided reinforcement learning Cheng & Xu (2023) or from demonstrations (Liu et al., 2025; Quartey et al., 2026), and skills and structure are discovered from demonstrations (Zhu et al., 2022; Wan et al., 2024). Planners and language models sequence learned or human-guided skills (Mandlekar et al., 2023; Dalal et al., 2024; Agia et al., 2023). Transition policies and skill-chaining methods address handoffs between skills (Konidaris & Barto, 2009; Lee et al., 2019; 2022; Mishra et al., 2023); APPL's overlapping training segments similarly widen a successor skill's initiation set. MaestroMotif and SCALAR use

language skill specifications for both training and composition, but these specifications describe skill effects rather than inductive bias (Klissarov et al., 2025; Zabounidis et al., 2026).

Agents that design learning systems. Further systems shape rewards, generate simulation tasks, and design sim-to-real rewards and domain randomization (Xie et al., 2024; Wang et al., 2024d;c; Ma et al., 2024b). Language models also compose encoder architectures, write policy structures from domain knowledge, and automate reinforcement-learning pipelines (Wang et al., 2024a; Yu et al., 2025; Zhu et al., 2025; Wei et al., 2025); they also run policy-improvement loops (Xiao et al., 2026; Khandelwal & Gupta, 2026; Yu, 2026). When learned skills are reused by a later decision maker, they are described by names, code, or effects (Wang et al., 2024b; Klissarov et al., 2025; Zabounidis et al., 2026; Nie et al., 2026); Agentic Skill Discovery retains several policy variants per skill and leaves choosing among them to future work (Zhao et al., 2026).

D.3 Comparison with the closest methods

Table 11 compares APPL with methods that come closest to either role. The comparison concerns method design; the settings differ, ranging from real robots to simulated manipulation, NetHack, and Craftax, and the table implies no empirical ranking.

Agent-designed priors. LGA and KALM show that language and vision-language models can choose representation priors for imitation learning, and ATK selects task-driven keypoints automatically (Peng et al., 2024; Fang et al., 2025; Zhang et al., 2025). Each automates one family of priors and produces one design per task, which the policy consumes as input. APPL does not claim agent-designed priors by themselves as new. Its construction agent proposes priors from several families for each skill, keeps the resulting policies as alternatives, and reuses each prior as runtime documentation.

SymSkill. SymSkill is the closest precedent to APPL (Shao et al., 2025). Offline, it jointly learns predicates, operators, and goal-oriented skills from unlabeled, unsegmented demonstrations, using relative frames, with a VLM assisting the choice of reference frames, and stable dynamical-system policies. Online, a symbolic planner composes and reorders skills to reach goals stated as conjunctions of learned predicates, and the system recovers from failures at both the motion and symbolic levels in real time. Both methods learn reusable skills from complete demonstrations and let structure found during learning organize long-horizon composition. They differ in three ways. First, SymSkill's structure follows one design family, whereas APPL's construction agent proposes priors from different families for each skill and implements them as representations, architectures, or losses. Second, SymSkill associates each operator with a learned skill policy, whereas APPL keeps several policies with different priors for the same skill and lets the runtime agent choose among them. Third, SymSkill's planner reads symbolic preconditions and effects, whereas APPL's runtime agent reads natural-language priors, handoff overlap, and measured training support, and sets a duration and stop conditions for each call. The approaches are complementary, since SymSkill's learned predicates and reactive recovery could provide execution checks and fallback behaviors within APPL's interfaces.

Shared abstractions. Beyond SymSkill, Lorang et al. (2025) and DR-LfD (Chen et al., 2026) also let one structure serve both learning and planning. In each, a fixed rule or template determines how the abstraction constrains a controller, and each skill receives one learning recipe. MaestroMotif and SCALAR share language specifications between training and composition (Klissarov et al., 2025; Zabounidis et al., 2026), but these specifications describe what a skill achieves and when it starts or stops, rather than the inductive bias or training support of its policy.

Agentic harnesses. Harness VLA, BATON, and RoboHarness invoke policies with an instruction, a step budget, and a stop condition, as APPL does; APPL does not claim this call as new. These systems learn operating ranges, entry conditions, or policy cards from executions after the policies exist. APPL derives its documentation from the construction-time prior and from the overlap built into training data, so this information is available before the first execution. RoboClaw keeps one agent across data collection, training, and deployment, but its agent selects data for a fixed training recipe rather than structural priors.

Table 11: Design comparison with the closest methods. "Reads" is what the high-level decision maker sees about each low-level policy before or during invocation.

Method	Low-level policies	Structure designed by	Alternatives per skill	Reads	Decision maker
LGA (Peng et al., 2024)	Imitation from few demonstrations	A language model, one state abstraction per task	No	Nothing; the abstraction is the policy input	None
KALM (Fang et al., 2025)	Keypoint-conditioned imitation	A VLM proposes keypoints checked on demonstrations	No	Nothing	None
SymSkill (Shao et al., 2025)	Dynamical-system skills in relative frames	Co-invented predicates; a VLM selects frames offline	No	Predicates and operators	Symbolic planner
Lorang et al. (2025)	Diffusion controllers	A fixed rule restricts inputs to operator-relevant objects	No	Learned operators	Symbolic planner
DR-LfD (Chen et al., 2026)	Diffusion policies and equivariant primitives	A fixed rule based on contact complexity	No	Learned initiation and termination keyposes	Task and motion planner
MaestroMotif (Klissarov et al., 2025)	Reinforcement-learned skills in NetHack	Human skill descriptions; LLM-derived rewards	No	Skill descriptions, initiation and termination code	LLM-written policy over skills
Agentic Skill Discovery (Zhao et al., 2026)	Reinforcement-learned skills in simulation	An LLM proposes tasks and rewards	Yes; selection left open	Skill names or descriptions	LLM
RoboClaw (Li et al., 2026)	Fine-tuned $\pi_{0.5}$	A fixed recipe; the agent chooses data	No	Subtask instructions and tool outputs	VLM agent
Harness VLA (Zhang et al., 2026)	Frozen VLA and analytic primitives	None; policies are pretrained	No	Memory of successful calls	Coding agent
BATON (Xu et al., 2026)	Frozen VLA and analytic primitives	None; policies are pretrained	No	Readiness and handoff conditions found by exploration	Coding agent
RoboHarness (Huang et al., 2026)	Pre-existing heterogeneous policies	None; policies are given	No	Policy cards updated from execution	Coding agent
APPL	Diffusion policies from 12 demonstrations per task	The construction agent, per skill	Yes; three per skill in Exp. 2, plus one after verification	Prior, handoff overlap, measured training support, and verification results	Runtime agent

Alternatives and handoffs. Agentic Skill Discovery retains several policy variants per skill and leaves the choice among them to future work (Zhao et al., 2026); in APPL, the runtime agent makes this choice using the priors that distinguish the variants. APPL's overlapping training segments

follow work that widens a successor skill's initiation set (Konidaris & Barto, 2009; Lee et al., 2022), and we treat the overlap as a design choice rather than a contribution.