OBJECT COUNT GENERALIZATION IS AN INTERFACE PROBLEM: WARPING A FROZEN SKILL INSTEAD OF RETRAINING IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning policies for robot manipulation degrade when the number of objects N in the scene grows beyond the training range. This is a covariate shift that also changes the dimension of the state. We propose WAVE (Warping Actions and Views of an Expert), which extends a pretrained single-object skill to many objects without updating its weights. WAVE has three parts. First, a selection policy picks the one object the robot works on next. Second, a warping policy reads a local view of the scene around that object, whose size does not depend on N. It emits a multiplier and an offset for every quantity the skill reads and every action it outputs. Third, the skill acts under this affine warp. It is shown a scene that holds a single object, while the warp steers it around the objects it is not shown and edits the actions that are sent to the robot. Both the selection and warping policies are trained with PPO on the multi-object reward and use the skill as a black box, so WAVE is task-general and skill-agnostic. It requires only that the skill was trained on the same embodiment and that every object has a goal pose. We also analyze the probability that the composition places all N objects within a budget of skill invocations that grows linearly in N. We evaluate WAVE on cube arrangement tasks with a gantry robot and with a Franka Panda arm. Trained with at most nine cubes, it generalizes zero-shot to several times its training count, on the gantry to scenes of up to thirty cubes. Published methods that learn multi-object manipulation show count generalization up to six objects. Fine-tuning the skill, learning from the same local view without it, and an entity transformer, even when distilled from the pretrained skill, all collapse within their training counts. Code, videos and full results are available at [anonymized].

1 INTRODUCTION

Reinforcement learning (RL) learns a policy that maximizes the expected cumulative reward of an agent interacting with an environment (Sutton & Barto, 2018). Policies trained this way for robot manipulation solve the scene they were trained in and often degrade when that scene changes (Kirk et al., 2023). This work addresses a covariate shift (Shimodaira, 2000), a change of the state distribution under unchanged dynamics. In our case the number of objects in the scene changes, so a policy trained with a few objects is deployed with many. Such a shift also changes the dimension of the state, and appears for example when a robot that has learned to manipulate a few plates is asked to fill a dishwasher. This work asks what can be learned around a frozen pretrained skill to enable zero-shot generalization to object counts unseen during training.

Two families of prior work address this shift. Set and relational architectures let a policy act at any object count (Zaheer et al., 2017; Zambaldi et al., 2019). Robot policies built this way learn the manipulation behavior itself (Li et al., 2020; Haramati et al., 2024; Lanier et al., 2019; Li et al., 2022). That is, a learned policy issues the robot command at every control step, and no primitive, planner or model supplies the motion. Such policies report whole-task success of at most 0.75 at six objects (Li et al., 2020) and at most 0.36 at eight (Park et al., 2025b; Li et al., 2025) on their respective benchmarks. Further, the theory of size extrapolation shows that networks which pool over every entity can fail to generalize to sets larger than those seen in training (Xu et al., 2021; Veličković et al., 2025; Haramati et al., 2024). Works that keep a trained skill frozen and learn a

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.

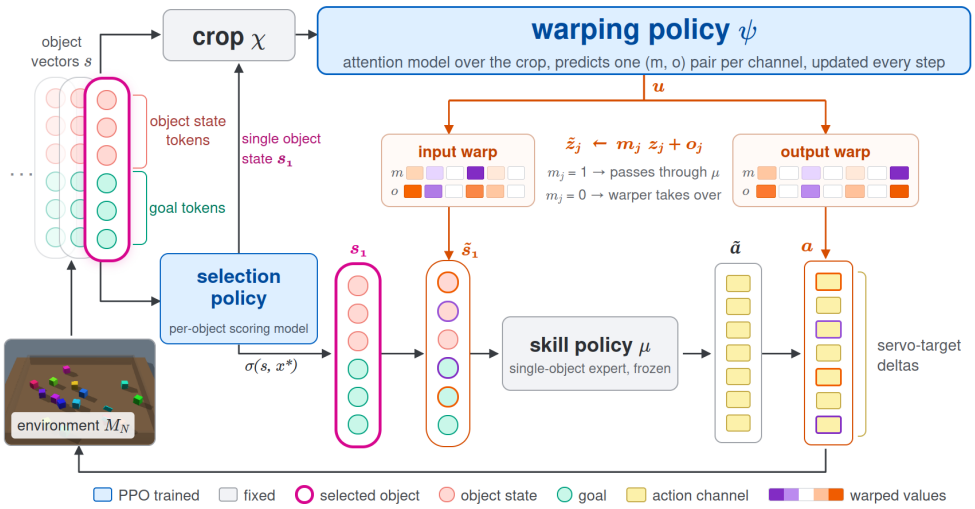

Figure 1: Overview of WAVE. The selection policy picks one object x^* that is not yet placed. The presentation σ shows the frozen skill μ only x^* and its goal. The warping policy ψ reads the crop χ around x^* and edits every input and action channel of μ with a multiplier and an offset. Once x^* is placed, the selection policy picks the next object. Only the two policies are trained.

correction around it (Silver et al., 2018; Johannink et al., 2019; Yuan et al., 2025; Wagenmaker et al., 2025; Hirschowitz & Ramos, 2026; Zhang et al., 2023; Alqithami, 2026; Park & Lee, 2026) absorb dynamics shifts, sensor changes and task mismatch, but none edits both the input and the output interface of a frozen skill to generalize across the object count.

We propose WAVE (Warping Actions and Views of an Expert), which learns a warping policy around a frozen skill. Because the skill can already place one object, training only has to learn the corrections needed for many. Figure 1 gives an overview, and our contributions are as follows.

- We propose WAVE, which composes a frozen single-object skill with a learned warping policy. The policy applies an affine edit to every input and output channel of the skill and is trained with PPO. It uses the skill as a black box, so it applies to any goal-conditioned skill trained on the same robot and to any task that gives every object a goal pose. We characterize which shifts such a warp can absorb (Section 3.5).
- We train WAVE on six cube arrangement tasks, on a gantry robot and on a Franka Panda arm with a gripper and with a pusher, with at most nine objects and evaluate it zero-shot on up to thirty. On the gantry, this is two to five times the largest count to which published methods that learn the manipulation itself generalize.
- We compare our method against nine baselines, which fine-tune, distill, jump-start or command subgoals to the same skill, or train an entity transformer (Haramati et al., 2024) or a policy on WAVE's own crop from scratch. None generalizes to more than five cubes, even when given a scripted rule that only offers objects whose goal is currently fulfillable, which WAVE does not use. We also ablate the parts of WAVE's architecture.

2 Related Work

A large body of work builds policies that are defined for any object count by making the network permutation invariant over a set of object entities (Zaheer et al., 2017; Zambaldi et al., 2019). Entity-centric RL (ECRL) (Haramati et al., 2024) learns per-object latent particles from pixels and trains an Entity Interaction Transformer with three objects that generalizes to six objects. EC-Diffuser (Qi et al., 2025) replaces RL by behavior cloning with an entity-centric diffusion Transformer. Li et al. (2020) stack blocks with an attention-based graph network trained by a curriculum that adds blocks, and their reported success declines beyond the counts the curriculum visited. Relational, curriculum and slot-based variants (Mambelli et al., 2022; Lanier et al., 2019; Li et al., 2022; Locatello et al.,

2020; Zadaianchuk et al., 2021) and the successors of ECRL (Qi et al., 2025; Haramati et al., 2026) report whole-task success at up to six objects. Li et al. (2020) show up to nine blocks as a figure without values, Haramati et al. (2024) show a twelve-object sort qualitatively, and the stacking variant of that line is evaluated at two to four objects (Haramati et al., 2026). Janisch et al. (2020) reach thirty blocks in a symbolic block world without physics simulation. Slot models expose the count through their number of slots (Zimmermann et al., 2023; Krimmel et al., 2024). Theoretical work on size extrapolation shows that networks which pool over every entity can fail on larger sets (Yehudai et al., 2021; Veličković et al., 2025), and that they can generalize when the local structure they see does not change with the count (Yehudai et al., 2021; Ouellette et al., 2023). Large-scale RL learns stacking at a fixed small count (Lampe et al., 2024; Macaluso et al., 2026). On the offline cube benchmarks, which train one policy per count, the best published baseline completes the stacking goal in 0.23 of episodes with two cubes and in none with three or four (Park et al., 2025a). The hierarchical methods evaluated at eight cubes reach at most 0.36 averaged over five goals, from success on the rearrangement goals and almost none on the two stacking goals (Park et al., 2025b; Li et al., 2025).

In task and motion planning, count generalization is easy, because a discrete search over a known symbolic model scales to any number of objects and motion primitives supply the manipulation (Garrett et al., 2020; Chitnis et al., 2022). The same holds when a network selects scripted primitives (Zeng et al., 2020; Lin et al., 2022). Neither applies here, since we have no model and no primitives, so the learned manipulation itself must generalize to more objects. Methods that pass arguments to a frozen executor (Riedmiller et al., 2023) assume that the executor works for every argument.

A single-object skill can also be reused without freezing it. Kickstarting distills a teacher policy into a new student while the student learns the task reward (Schmitt et al., 2018), and jump-start RL rolls a guide policy for the first part of every episode and hands the rest to an exploration policy, moving the switch point earlier as the exploration policy improves (Uchendu et al., 2023). Both leave the multi-object behavior to a network trained on the new task.

Another line keeps a trained policy frozen and learns an edit of its interface. Most works improve the policy on its own task and distribution, by an action residual (Silver et al., 2018; Johannink et al., 2019; Yuan et al., 2025; Ankile et al., 2025), steered diffusion noise (Wagenmaker et al., 2025), inserted latents or edited activations (Yu et al., 2026; Das et al., 2026; Miao et al., 2026), or a remapped observation (Liu et al., 2026). The few that target generalization absorb dynamics shifts (Hirschowitz & Ramos, 2026; Zhang et al., 2023), a changed sensor layout (Alqithami, 2026) or a task mismatch around a shared backbone (Park & Lee, 2026), never a change in the number of objects. Closest to our mechanism, Zhang et al. (2023) learn diagonal projections of the perceived state and commanded action of a frozen pivoting skill and generalize across object properties that change the dynamics. They choose the projection once per episode from a supervised object descriptor. In our setting, other objects and the skill's own placements change the scene during the episode, so the warp is chosen anew at every step. Appendix I compares these methods with ours.

3 METHODOLOGY

3.1 Intuition. WAVE takes a skill μ that can move a single object to its goal and uses it, without changing its weights, to arrange many objects. A selection policy picks one object at a time, and μ is shown a scene in which this object is the only one. The other objects are still there, so a warping policy reads the neighborhood of the object and bends the inputs and actions of μ where needed, for example to steer the robot around a stack or to let go of a cube on top of another (Figure 1). Where nothing is in the way, μ acts unchanged.

Warping the skill is necessary because, with many objects, the behavior of μ itself has to change. Choosing the next object or sending μ to intermediate subgoals, as in hierarchical RL, leaves this behavior unchanged. For example, a subgoal can bring a cube above a stack, but it does not change how μ releases the cube there. A policy that adds a correction to the action of μ does change the behavior. However, μ still sees the unedited scene and moves along paths that collide with objects it is not shown, so the correction has to cancel part of μ's action at every step. The warp edits both what μ sees and what it outputs. Editing the input makes μ itself compute a corrected action, and editing the output produces behavior μ never learned, such as the release onto a stack.

WAVE generalizes to higher object counts for two reasons. First, μ can already place a single object, so it is a strong exploration prior that makes multi-object tasks learnable and training only learns the corrections needed with many objects. Second, nothing μ or the warping policy reads grows with the number of objects. Thirty cubes produce the same kind of input as nine, whereas a policy that reads all objects receives a new, out-of-distribution input with every added object.

3.2 Problem and composition. Let $M_1 = (\mathcal{S}_1, \mathcal{A}, P_1, R_1, \mathcal{G})$ be a goal-conditioned manipulation MDP whose state holds one robot and one movable object, whose goal $g \in \mathcal{G}$ is a pose for that object, and let $\mu : \mathcal{S}_1 \times \mathcal{G} \to \mathcal{A}$ be a deterministic policy trained on it. Let $M_N = (\mathcal{S}_N, \mathcal{A}, P_N, R_N, \mathcal{G}^N)$ be the task in which the same robot must bring N objects to the goal poses $g_1, \ldots, g_N$, and the objects collide with, rest on and occlude each other. As illustrated in Figure 1, we compose μ into a policy π_N for M_N that acts on one selected object x^* at a time and computes

$$\pi_N(s) = W_{\text{out}}\big(\mu(W_{\text{in}}(\sigma(s, x^*); u)); u\big), \qquad u = \psi(\chi(s, x^*)), \tag{1}$$

with s the state of M_N, $\sigma(s, x^*) \in \mathcal{S}_1 \times \mathcal{G}$ the presentation of the selected object, $\chi(s, x^*)$ the crop, a local view of the scene with a fixed width, ψ the learned warping policy, and u the warp parameters it emits. W_{in} and W_{out} are the warping functions that apply u as per-channel affine edits of μ's input and of its action. The presentation $\sigma(s, x^*)$ is the state and goal μ would see in M_1 if x^* were the only object, with nothing about the other objects, expressed relative to the robot so that the presented scene stays inside the distribution μ was trained on. The warping policy runs at the same rate as μ. At every control step, ψ computes a new warp from the current crop before μ acts. The selection policy decides again whenever a cube is placed or a placed cube is knocked off its goal (Section 3.3). An attempt is the period in which the selected object stays the same.

WAVE makes three assumptions about μ. μ is a goal conditioned policy, it was trained on the same robot, so that its output is a command for the robot at hand and the output side needs no rewiring, and its input layout $\mathcal{S}_1 \times \mathcal{G}$ is known, so that the presentation can be rendered into it. No weights, gradients, activations or knowledge of what μ does are used.

3.3 Selection policy. The selection policy scores every object that is not at its goal. For each candidate x_i it reads the position of x_i relative to the gripper and to its goal, the robot state, and the neighborhood of x_i. A shared MLP maps these features to two scalars per object, a priority f_i and a feasibility $\hat{p}_i$ (the probability that x_i can be successfully placed). The score is then computed as

$$v_i = \beta \tanh f_i - \alpha \left(\tilde{d}_i + {}^{\ell_i}/_{\hat{p}_i} \right) + \theta_i, \qquad x^* = \arg\max_i \, v_i, \tag{2}$$

where $\tilde{d}_i$ is the gripper to object distance, ℓ_i is an estimated length of one attempt at x_i (distance of x_i to its goal plus a small constant), and α a fixed travel price. Since ${}^1/_{\hat{p}_i}$ is the expected number of attempts x_i takes, the bracket is the expected travel that placing x_i costs. The priority head starts at 0 and the feasibility head starts at 1, so the untrained rule is shortest gripper-to-object-to-goal path first, from which the selection policy deviates only where another order yields a higher return. The priority is bounded by the constant β, so the learned priority can reorder candidates of similar expected travel but cannot overturn a travel difference larger than $2\beta/\alpha$. The feasibility head is trained on the realized outcomes of past attempts, so an object that has a low feasibility (e.g., because it is blocked by a neighbor) has a high expected travel and is less likely to be selected, but it is not removed from the pool. Order constraints are learned, not specified. For example, a cube can only be stacked once the cube below its slot is placed. The learned holding bonus θ_i is nonzero only for the current x^*, so that the selection policy keeps it until it is placed.

3.4 The warping policy. The crop $\chi(s, x^*)$ contains the position and goal of x^*, the robot state and the action of μ on the unwarped presentation $\sigma(s, x^*)$. In addition, we compute a set of geometric features for every other object that is not x^*. These features hold its position relative to x^*, to the goal of x^* and to the robot, and whether it is placed. A per-pair network maps each object feature vector to a fixed number of bounded scores, and an elementwise minimum over these scores reduces them to a single neighborhood encoding. We use the minimum because the closest obstacle is the one an attempt is most likely to hit. The minimum is also independent of the count, since a neighbor five centimeters away yields the same score in a scene of three objects and in a scene of thirty. A mean or a softmax would dilute as N grows. Objects farther than a fixed range from x^* are masked, so the input value range of the per-pair network is equally independent of the object count.

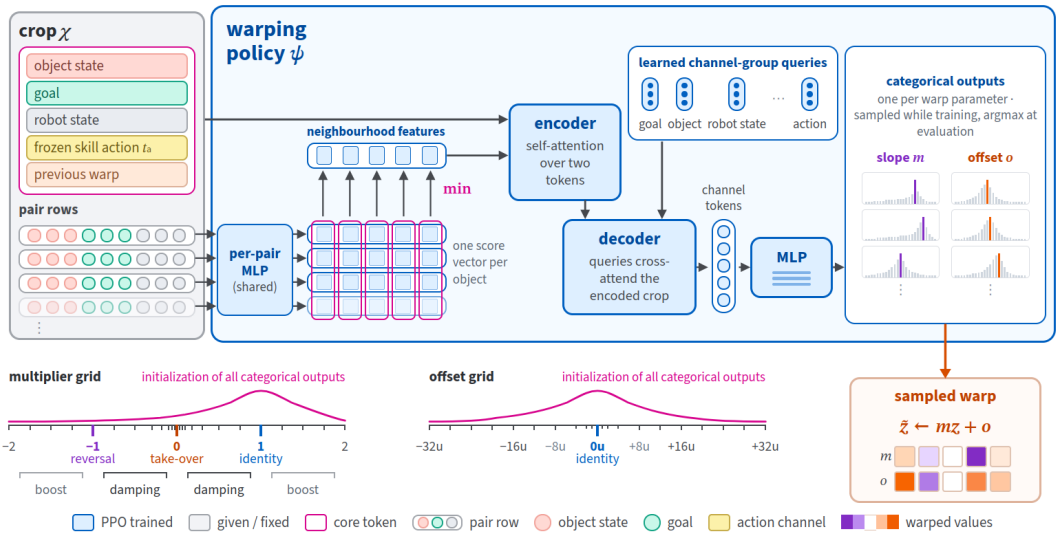

Figure 2: The warping policy ψ and its initialization. The lower axis shows the multiplier grid, dense between take over ($m = 0$) and identity ($m = 1$), with the initial mass on the identity bin.

A channel is one scalar of μ's interface, a single coordinate of the state it reads or of the action it emits. The channels that carry one physical quantity, the presented goal position or the action for instance, form a channel group. The warping policy emits the warp parameters u of equation 1, one pair (m_j, o_j) per channel j. W_{in} and W_{out} apply that pair to the channel's value z_j as $\tilde{z}_j \leftarrow m_j z_j + o_j$. At $m_j = 1$ and $o_j = 0$ the channel passes through unchanged, at $m_j = 0$ its input is removed and o_j becomes its value, so the warping policy takes over the channel and commands it directly. As shown in Figure 2, ψ encodes the reduced pair scores and the features of x^*, its goal and the robot as the tokens of a small transformer, decodes one query per channel group and emits one categorical distribution per warp parameter over a fixed grid of values. The multiplier grid is dense between take over and identity, the offset grid is a signed ladder in multiples of a step size per robot (1.5 cm on the gantry), and one further bin per channel repeats the value applied at the previous step, so the warping policy can choose to keep its previous edit. Every distribution starts with most of its mass on the identity value, so training starts from the unwarped skill and departs from it only where that yields a higher task reward.

The warping policy and the selection policy are trained jointly with PPO (Schulman et al., 2017) on the return of M_N. The reward R_N (Appendix A) pays a small reward at every control step proportional to the fraction of objects currently placed, a bonus when all of them are, and a one time bonus the first time an object is placed. The reward depends only on which objects are at their goals. It contains no term for a route, a grasp or a release and is the same for every task in Section 4.

3.5 Absorbable shifts. A warp changes the coordinates of μ's input and of its action, which makes it a feedback transformation of the dynamics μ acts on (Brunovský, 1970; Jakubczyk & Respondek, 1980). Let M_1' be the single-object task at deployment, with μ's interface unchanged and only its dynamics P_1' altered. We call the shift from M_1 to M_1' *absorbed* when the warp turns every trajectory of M_1' into one of μ's own trajectories in M_1,

$$W_{\text{in}} P_1'(s_1, W_{\text{out}} a) = P_1(W_{\text{in}} s_1, a). \tag{3}$$

W_{in} is the input warp and W_{out} is the output warp. An attempt under an absorbed shift then runs like an episode of μ alone in M_1, only in warped coordinates. We call this a solo episode. Its success rate is therefore μ's own, and this holds without measuring anything about μ (Appendix H).

Which shifts a warp can absorb depends on what the warp reads. A warp that applies the same edit at every step already absorbs any recalibration of μ's sensors and actuators, since it inverts the recalibration channel by channel, and any translation of the scene parallel to the floor. A warp that varies with the state, as ψ does through the crop, absorbs far more. It can retarget μ at every step, which produces detours, hovering, retreating and holding, and it can take over any channel of μ and

command it directly. Proposition 2 in Appendix H guarantees that each of these shifts is absorbed. Beyond this guaranteed class, the warping policy is free to learn any edit that raises the return, for example to compensate for small differences in the dimensions of the robot. More objects do not change what μ receives. σ shows μ only the selected object, and since σ is expressed relative to the robot, a larger arena only shifts positions. What changes with more objects is the crop, because the scene around x^* becomes denser. The warping policy reads this crop and turns the interactions with the other objects into a change of coordinates for μ. Through these coordinate changes, the warping policy can add behaviors to μ that it never learned, such as curved trajectories to avoid collisions, safe release of a cube on top of another, or nudging a neighbor out of the way. A transformation matrix in place of the per-channel edit would extend this to a robot with a different number of joints, which we leave to future work.

A warp cannot absorb shifts that are not observable from the crop, or shifts that it cannot invert. A shift is *unobservable* if two deployments that require different warps produce the same crop, so that a single fixed edit has to serve both. This would occur for example when two cubes of equal size differ in friction, which the crop does not carry. A shift is *not invertible* if canceling it would require moving the state in a direction μ's action cannot reach, so that no multiplier or offset reproduces the needed correction. This would occur for example at a kinematic singularity, with the arm stretched straight, where no joint motion moves the gripper further out whatever gain the warp applies.

3.6 Bounded completion guarantee. Whether WAVE completes a task of N objects within a number of skill invocations linear in N is decided by one measurable quantity, the net progress per attempt, $\rho = (1 - \delta_{\text{cov}})\, p_e - \lambda$. Here p_e is the success probability of an attempt, over the objects the selection policy actually picks, λ the expected number of placed objects an attempt unplaces, and δ_{cov} the fraction of crops outside the training distribution, which we pessimistically assume to be all failures. ρ lower bounds the expected net gain in placed objects per attempt. If $\rho > 0$ then for every factor $c > 1/\rho$

$$\Pr[\,\text{task not complete within } cN \text{ attempts}\,] \le \exp\left(-\frac{2(\rho c - 1)^2 N}{c(1 + \lambda_{\max})^2}\right), \tag{4}$$

with $\lambda_{\max}$ the largest number of objects one attempt can unplace (Theorem 1 in Appendix H, from Azuma's inequality). Unlike value-based bounds (Haramati et al., 2024), no term grows with the number of added objects, since the count enters only through the number of attempts and the constants (Appendix H), which we measure per count (Table 9). On cube stacking ($h = 3$) ρ is positive at every count we evaluate but shrinks from $+0.036$ at seven cubes to $+0.011$ at twenty and $+0.008$ at thirty, where success falls to 0.48, with δ_{cov} at most 0.0014. The net progress is lost to knockdowns, not to crops outside the training set. The constants therefore show from which count completion slows. Between seven and thirty cubes, the attempt budget $c > 1/\rho$ that the guarantee requires grows from about 28 to 125 per object. On the pyramid, ρ changes sign between seven and ten cubes, from $+0.005$ to -0.013 and down to -0.035 at thirty with δ_{cov} again at most 0.0006, and success collapses right after, from 0.90 at ten to 0.35 at twenty cubes, so ρ decreases as success declines.

4 Experiments

We evaluate WAVE on six cube arrangement tasks, stacks of height $h = 3$ and $h = 6$, a line, a square, a pyramid and a wall (Figure 5). A gantry robot with a parallel-jaw gripper runs every task with one to thirty cubes (Section 4.1). A Franka Panda arm runs five of them with a parallel-jaw gripper or a sphere-tip pusher, up to fifteen, twenty or thirty cubes (Section 4.2). Each policy trains on mixed counts of one to N_{train}, with N_{train} between six and nine depending on the task and robot (Appendix A). An episode succeeds when every cube simultaneously rests within a small tolerance of its goal pose on its intended support at the end of the episode, and we call the fraction of successful episodes *whole-task success*. For every object count evaluation, we use initial layouts which were not part of the training pool. Every evaluation uses a deterministic policy, 20 s per cube and three seeds. The robot, the skill, the tasks, all hyperparameters, and the full result tables are given in Appendices A and B.

4.1 Count generalization on the gantry. Trained on one to seven cubes, WAVE (Figure 3) stays at or above 0.92 on cube stacking ($h = 3$) up to twenty cubes and falls to 0.48 at thirty. The line,

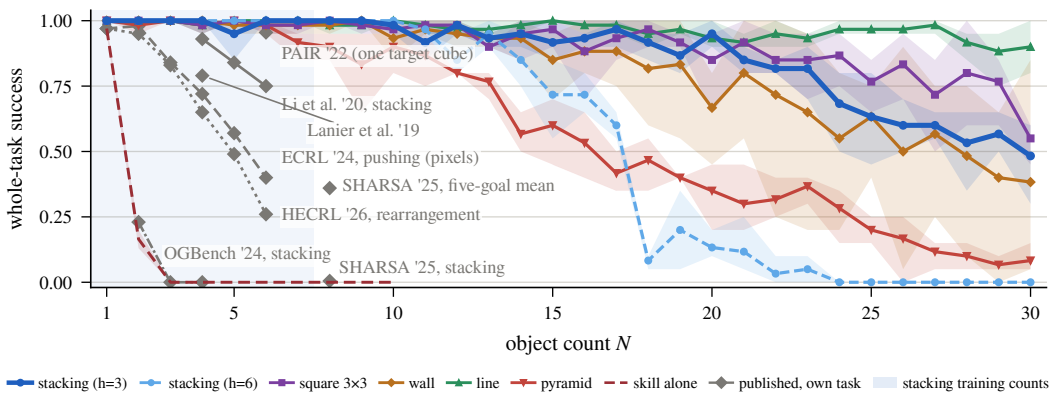

Figure 3: Whole task success against the number of cubes N for WAVE on the six gantry tasks (color), and for published methods on their own tasks (gray, Appendix E). The shaded band is the training range of cube stacking and the dashed red curve the skill alone baseline on cube stacking ($h = 3$). WAVE and skill alone are averaged over seeds with bands for the range.

square, pyramid and wall complete between 0.80 (pyramid) and 0.97 (line and square) at twice their training count and between 0.08 (pyramid) and 0.90 (line) at thirty (Table 2).

Published methods that learn the manipulation itself reach at most 0.75 at six objects (Li et al., 2020) and 0.36 at eight (Park et al., 2025b; Li et al., 2025), while every WAVE task is at or above 0.98 at six. WAVE stays above the 0.75 of Li et al. (2020) up to twenty-three cubes on cube stacking, the task closest to theirs, a nearly fourfold extension, and up to thirty on the line, a fivefold one. These comparisons cross tasks, simulators, embodiments and in places success criteria (Appendix E), since no controlled comparison exists at these counts.

Cube stacking with $h = 6$ tests placement precision. The training scenes hold at most seven cubes, one full stack and the base cube of a second, so no training episode places a cube onto the second stack. WAVE still builds the second stack up to five cubes high and completes 0.97 or more of the episodes up to eleven cubes. Success falls to 0.85 at twelve cubes, where the second stack needs its sixth cube, and is zero from twenty-four on, lost to toppled stacks rather than missed placements.

At high counts, the failures come from knockdowns of placed cubes. On cube stacking ($h = 3$), on average, 26.8 of 30 cubes are placed at the same time at the peak, but knockdowns of placed cubes grow from 7.5 per episode at ten cubes to 126 at thirty, shrinking the net progress per attempt (Section 3.6) until episodes run out of time (0.36 of them at thirty). The frozen skill alone (the warp fixed at the identity) succeeds in 0.97 of the one-cube and 0.17 of the two-cube scenes and in none with three, since on its way to the next cube it knocks the placed ones off their goals.

4.2 Count generalization on the Panda arm. We transfer the same method and reward to a Franka Panda arm with 4 cm cubes in two embodiments, each with its own frozen skill (Appendix A). A parallel-jaw gripper handles cube stacking ($h = 3$), the pyramid and the wall. A sphere-tip pusher, whose skill pushes one cube to a planar position and yaw goal, handles the line and the square. Figure 4 shows the first fifteen counts and Table 3 the full range of each task.

With the pusher, WAVE completes 0.95 of the line episodes at nine cubes, its largest training count, 0.78 at fifteen and 0.62 at twenty, and failed episodes end only a cube or two short (14.2 of fifteen cubes placed on average on these episodes). The square is harder, 0.60 at nine and 0.10 at fifteen, since its interior slots leave the pusher no free approach side and late cubes are pushed in across placed ones until the episode runs out of time.

With the gripper, success already falls within the training range. Cube stacking completes 0.75 at six, level with the best published result (Li et al., 2020), and 0.37 at ten, while the pyramid and the wall complete 0.22 and 0.05 at six. Both failure modes of Section 4.1 appear early. At six cubes the wall ends 0.76 of its episodes by timeout, and on cube stacking the fraction of placed cubes that stay placed falls from 0.95 at six cubes to 0.66 at fifteen. We attribute this mainly to the large body of

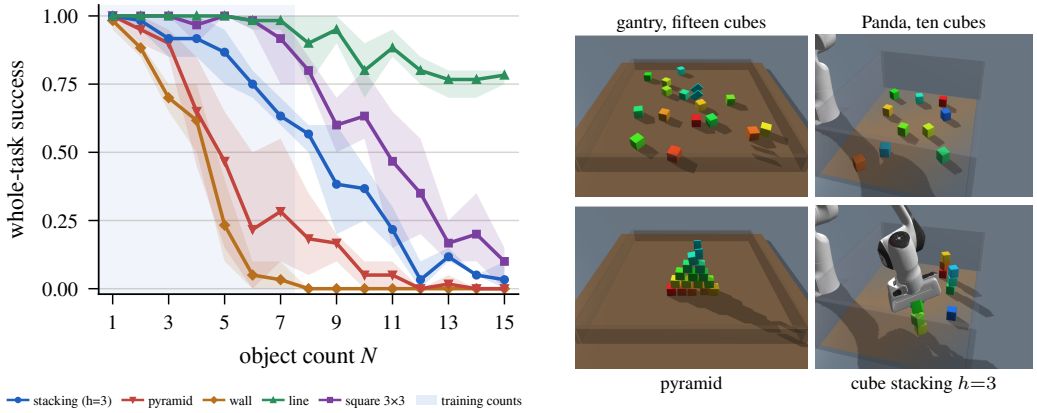

Figure 4: Left, whole-task success against the object count on the Panda arm, the first fifteen counts of every task. Right, a start scene (top) and a completed episode (bottom) on each robot, a fifteen-cube pyramid on the gantry and ten-cube stacking ($h = 3$) on the Panda. All tasks are rendered in Figures 5 and 6.

Table 1: Whole task success of WAVE and six of the nine baselines on the gantry, the mean over three seeds. The first three columns give how each method uses μ, which objects besides x^* it sees, and how it picks the next object. Every seed of every baseline is at 0.00 at six and at thirty cubes. Bold marks the highest mean per count. Seed ranges, every count and the other three baselines are in Table 4.

				line, N					cube stacking ($h = 3$), N				
	μ	other objects	selection	1	2	3	6	30	1	2	3	6	30
WAVE	frozen	crop	learned	**1.00**	**1.00**	**1.00**	**1.00**	**0.90**	**1.00**	**1.00**	**1.00**	**1.00**	**0.48**
skill alone	frozen	none	scripted	0.79	0.35	0.00	0.00	0.00	0.97	0.17	0.00	0.00	0.00
TD7 on the crop	–	crop	scripted	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fine-tuning μ on the crop, $\omega = 1$	fine-tuned	crop	scripted	0.96	0.43	0.03	0.00	0.00	0.98	0.64	0.03	0.00	0.00
HRL subgoals	frozen	crop	learned	0.54	0.18	0.00	0.00	0.00	0.83	0.53	0.10	0.00	0.00
EIT (Haramati et al., 2024)	–	all	–	0.20	0.00	0.00	0.00	0.00	0.53	0.00	0.00	0.00	0.00
EIT distilled from μ	distilled	all	–	0.82	0.56	0.37	0.00	0.00	0.94	0.20	0.00	0.00	0.00

the arm, which knocks neighbors off their goals when cubes are placed next to each other. Longer training, larger networks or a richer crop may close this gap and are left for future work.

We also deployed a cube stacking policy trained in simulation on a physical Panda arm, with cube poses tracked from AprilTag fiducials by four calibrated RGB cameras. It placed all five cubes of a two-tower task in 8 of 10 runs, and a policy fine-tuned for five-cube towers built a five-high tower (Appendix C).

4.3 Baselines. We compare WAVE on the line and on cube stacking ($h = 3$) with nine baselines, trained on the same gantry, task, reward and training counts, and with the same μ wherever one is used. Table 1 reports six of them and Appendix D all nine. Apart from HRL subgoals, which keeps WAVE's selection policy, the baselines that act on one object at a time use a scripted nearest-first rule that offers only objects whose goal is currently fulfillable, an easier task than the one WAVE learns from the reward. No baseline succeeds with more than five cubes.

TD7 on WAVE's crop without μ completes no episode at any count, and the Entity Interaction Transformer (EIT) of Haramati et al. (2024) learns only the one-cube task, 0.53 on cube stacking and 0.20 on the line. Neither has a skill to explore with, so both rarely place a cube and rarely see a reward. Distilling μ into the EIT makes it the strongest baseline on the line, 0.37 at three cubes, but it still fails within its training counts, since it learns every object interaction in one network whose input grows with each added cube, unlike the fixed-size inputs of WAVE's two policies.

Fine-tuning μ on its own input, fine-tuning it on the crop without an anchor and JSRL (Uchendu et al., 2023) succeed in at most 0.08 of the one-cube episodes. Without the crop, the same observation earns different returns depending on unseen neighbors. With the crop, the actor follows a critic that is still inaccurate on the multi-object return. Both fine-tuning variants also change actions that were already correct with one cube. Anchoring the actor to μ (Fujimoto & Gu, 2021) with $\omega = 1$ keeps the skill and improves on it at two cubes, 0.64 against 0.17 on cube stacking, but falls to 0.03 at three. The anchor penalizes every departure from μ equally. A warp can instead stay at the identity on some channels and steps and fully take over others ($m_j = 0$).

HRL subgoals, which moves only the goal μ pursues, completes 0.53 of the two-cube stacking episodes and at most 0.10 from three cubes on, since a subgoal changes where μ goes but not how it releases a cube onto a stack. Warping only μ's input channels, which also edits what μ sees of the robot and of the object, nearly matches the full warp, 0.81 against 0.82 mean success (Section 4.4).

4.4 Ablations. We ablate WAVE on cube stacking ($h = 3$), the square and the pyramid. First, we replace the selection rule of equation 2 with a random choice or a nearest-first choice over the objects whose goal is currently fulfillable, a filter the learned selection policy does not use. Second, we let the warp edit only the input or only the output channels of μ. Editing only the input is close to methods that remap observations (Alqithami, 2026; Liu et al., 2026), and editing only the output is close to residual and action distribution methods (Silver et al., 2018; Johannink et al., 2019; Hirschowitz & Ramos, 2026). Third, we let the warp emit only offsets or only multipliers. Random selection completes no episode with five or more cubes on cube stacking and the pyramid, or with nine or more on the square. Without any scripted gate, the learned selection policy matches the nearest-first rule on stacking, trails it on the square from twenty cubes on and is ahead on the pyramid, whose goals depend on each other. Warping only the input channels nearly matches the full warp, warping only the output channels collapses by ten cubes, and warps with only multipliers or only offsets fall in between, multipliers above offsets. Curves, analysis and the constants of equation 4 are in Appendix F.

To measure when in an episode each side of the warp is needed, we stop a running episode at every time step, copy the simulator, and continue the copies with either the input or the output warp switched off. We then measure how much longer the current phase of the manipulation (approach, grasp, carry, release or depart) takes compared to copies where the warp stays on. Switching off the output warp delays mainly the release, since the frozen skill never releases a cube on its own. Switching off the input warp slows down every phase, by 0.95–1.56 s, because it steers the skill throughout the episode. Appendix G gives the details and shows an example episode.

5 Conclusion and Limitations

We introduce WAVE, which extends a frozen single-object skill to multi-object arrangement tasks. It shows the skill one object at a time, learns a warping policy that edits every channel of the skill's interface, and learns a selection policy that finds from outcomes which object to attempt next. The skill is a black box, no learned component observes the object count, and all six tasks use the same reward and hyperparameters. Trained with at most nine cubes, WAVE generalizes on the gantry to two to five times the largest count reached by published methods that learn the manipulation itself, whereas fine-tuning the skill, a policy on the same local view and an entity transformer, even when distilled from the skill, collapse within their training counts. WAVE requires training one warping policy and one selection policy per task.

We evaluated WAVE in simulation on three embodiments and demonstrated the real-world transfer of a multi-object stacking policy on a physical Panda arm (Appendix C). The Panda gripper tasks degrade within their training range (Section 4.2). The method applies only to tasks that provide a goal pose for every object. All scenes consist of identical cubes, and since object size, shape and dynamics lie outside the guaranteed class of Proposition 2 in Appendix H, generalization over object geometry is untested. The skill moves one object at a time, so tasks that need the coordinated motion of several objects, or in which objects affect each other without touching, are out of scope.

AI USE STATEMENT

In this work, we used generative AI tools for related work surveys, implementations, conduction of experiments and drafting parts of this paper as well as some final edits. We have not used generative AI tools for summarization and reading of related papers, the ideation of our approach, the generation of plots or for the majority of the written contents of this paper. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

WAVE is basic research on making manipulation policies generalize across object counts, and every result in this paper except one demonstration on a physical Panda arm is simulated. The risks it touches are the generic ones of more capable robot manipulation, displaced manual handling labor and the possibility of building manipulation into systems used for harmful ends, and none is specific to this method or amplified by it. On the positive side, reusing a trained skill instead of retraining from scratch lowers the compute cost of extending a robot's competence.

REPRODUCIBILITY STATEMENT

Appendix A specifies the robot, the simulation, the frozen skill and its training, the six tasks and their goal predicate, the reward with all its constants, every hyperparameter of the warping and selection policies and the evaluation protocol. Appendix B reports the full per count results with seed ranges, Appendix F describes how the constants of equation 4 are measured, and Appendix H states every assumption of Proposition 1 and Theorem 1 and gives the complete proof. Code, videos and full results are available at [anonymized]

REFERENCES

Saad Alqithami. DC-Ada: Reward-only decentralized sensor adaptation for heterogeneous multi-robot teams. *IEEE Access*, 14:74241–74260, 2026. doi: 10.1109/ACCESS.2026.3687163.

Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement – residual RL for precise assembly. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2025. doi: 10.1109/ICRA55743.2025.11127442.

Pavol Brunovský. A classification of linear controllable systems. *Kybernetika*, 6(3):173–188, 1970.

Hien Bui, Yufeiyang Gao, Haoran Yang, Eric Cui, Siddhant Mody, Brian Acosta, Thomas Stephen Felix, Bibit Bianchini, and Michael Posa. Push anything: Single- and multi-object pushing from first sight with contact-implicit MPC. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

Satyajeet Das, Darren Chiu, Shashank Hegde, and Gaurav S. Sukhatme. Steering multirobot behavior via closed-loop affine activation editing. *arXiv preprint arXiv:2606.11489*, 2026. arXiv:2606.11489.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Scott Fujimoto, Wei-Di Chang, Edward J. Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For SALE: State-action representation learning for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Alexandre Galashov, Siddhant M. Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M. Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in KL-regularized RL. In *International Conference on Learning Representations (ICLR)*, 2019.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 440–448, 2020.

Seyed Kamyar Seyed Ghasemipour, Satoshi Kataoka, Byron David, Daniel Freeman, Shixiang Shane Gu, and Igor Mordatch. Blocks assemble! learning to assemble with large-scale structured reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7435–7469, 2022.

Raj Ghugare, Roger Creus Castanyer, Catherine Ji, Kathryn Wantlin, Jin Schofield, Karthik Narasimhan, and Benjamin Eysenbach. BuilderBench: The building blocks of intelligent agents. *arXiv preprint arXiv:2510.06288*, 2025.

Daniel Graves, Jun Jin, and Jun Luo. LISPR: An options framework for policy reuse with reinforcement learning. *arXiv preprint arXiv:2012.14942*, 2020. arXiv:2012.14942.

Dan Haramati, Tal Daniel, and Aviv Tamar. Entity-centric reinforcement learning for object manipulation from pixels. In *International Conference on Learning Representations (ICLR)*, 2024.

Dan Haramati, Carl Qi, Tal Daniel, Amy Zhang, Aviv Tamar, and George Konidaris. Hierarchical entity-centric reinforcement learning with factored subgoal diffusion. In *International Conference on Learning Representations (ICLR)*, 2026.

Ethan Hirschowitz and Fabio Ramos. Warp RL: Reshaping base policy distributions for dynamics adaptation. *arXiv preprint arXiv:2606.31043*, 2026. arXiv:2606.31043.

Bronisław Jakubczyk and Witold Respondek. On linearization of control systems. *Bulletin de l'Académie Polonaise des Sciences, Série des Sciences Mathématiques*, 28:517–522, 1980.

Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Symbolic relational deep reinforcement learning based on graph neural networks and autoregressive policy decomposition. *arXiv preprint arXiv:2009.12462*, 2020. arXiv:2009.12462.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, 2019. doi: 10.1109/ICRA.2019.8794127.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Markus Krimmel, Jan Achterhold, and Joerg Stueckler. Attention normalization impacts cardinality generalization in slot attention. *Transactions on Machine Learning Research*, 2024, 2024. URL `https://openreview.net/forum?id=llQXLfbGOq`.

Thomas Lampe, Abbas Abdolmaleki, Sarah Bechtle, Sandy H. Huang, Jost Tobias Springenberg, Michael Bloesch, Oliver Groth, Roland Hafner, Tim Hertweck, Michael Neunert, Markus Wulfmeier, Jingwei Zhang, Francesco Nori, Nicolas Heess, and Martin Riedmiller. Mastering stacking of diverse shapes with large-scale iterative reinforcement learning on real robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7772–7779, 2024. doi: 10.1109/ICRA57147.2024.10610297.

John B. Lanier, Stephen McAleer, and Pierre Baldi. Curiosity-driven multi-criteria hindsight experience replay. *arXiv preprint arXiv:1906.03710*, 2019. arXiv:1906.03710.

Qiyang Li, Seohong Park, and Sergey Levine. Decoupled Q-chunking. *arXiv preprint arXiv:2512.10926*, 2025.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4051–4058, 2020. doi: 10.1109/ICRA40945.2020.9197468.

Yunfei Li, Tian Gao, Jiaqi Yang, Huazhe Xu, and Yi Wu. Phasic self-imitative reduction for sparse-reward goal-conditioned reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12765–12781, 2022.

Yixin Lin, Austin S. Wang, Eric Undersander, and Akshara Rai. Efficient and interpretable robot manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, 7(2):2740–2747, 2022. doi: 10.1109/LRA.2022.3143518.

Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. VLS: Steering pretrained robot policies via vision-language models. *arXiv preprint arXiv:2602.03973*, 2026. arXiv:2602.03973.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Annabella Macaluso, Haochen Zhang, Ishaan Masilamony, Yingshan Chang, and Yonatan Bisk. Inductive generalization for robotic manipulation. *arXiv preprint arXiv:2606.20999*, 2026. arXiv:2606.20999.

Davide Mambelli, Frederik Träuble, Stefan Bauer, Bernhard Schölkopf, and Francesco Locatello. Compositional multi-object reinforcement learning with linear relation networks. *arXiv preprint arXiv:2201.13388*, 2022. arXiv:2201.13388.

Miranda Muqing Miao, Subin Kim, Brandon Yang, and Lyle Ungar. Contrastive conceptor activation steering (COAST): Unlocking vision-language-action models through hidden states. *arXiv preprint arXiv:2605.17144*, 2026. arXiv:2605.17144.

Simon Ouellette, Rolf Pfister, and Hansueli Jud. Counting and algorithmic generalization with transformers. *arXiv preprint arXiv:2310.08661*, 2023.

Bumgeun Park and Donghwan Lee. Adaptive policy backbone via shared network. In *International Conference on Machine Learning (ICML)*, 2026.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025a.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes RL scalable. In *Advances in Neural Information Processing Systems*, volume 38, 2025b.

Carl Qi, Dan Haramati, Tal Daniel, Aviv Tamar, and Amy Zhang. EC-Diffuser: Multi-object manipulation via entity-centric behavior generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Guannan Qu, Adam Wierman, and Na Li. Scalable reinforcement learning for multi-agent networked systems. *Operations Research*, 2022.

Balaraman Ravindran and Andrew G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1011–1018, 2003. URL `http://ijcai.org/Proceedings/03/Papers/145.pdf`.

Martin Riedmiller, Andrea Gesmundo, Tim Hertweck, and Roland Hafner. Less is more – the dispatcher/executor principle for multi-task reinforcement learning. *arXiv preprint arXiv:2312.09120*, 2023. arXiv:2312.09120.

Luana Ruiz, Luiz F. O. Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1702–1712, 2020.

Simon Schmitt, Jonathan J. Hudson, Augustin Zídek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Küttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018. arXiv:1803.03835.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018. arXiv:1812.06298.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

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.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

Dhruva Tirumala, Alexandre Galashov, Hyeonwoo Noh, Leonard Hasenclever, Razvan Pascanu, Jonathan Schwarz, Guillaume Desjardins, Wojciech Marian Czarnecki, Arun Ahuja, Yee Whye Teh, and Nicolas Heess. Behavior priors for efficient reinforcement learning. *Journal of Machine Learning Research*, 23(221):1–68, 2022. URL `https://jmlr.org/papers/v23/20-1038.html`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Marc Toussaint, Cornelius V Braun, Armand Jordana, Sayantan Auddy, Eckart Cobo-Briesewitz, Denis Shcherba, Tilman Burghoff, and Justin Carpentier. Combined constrained sampling and reinforcement learning for robotic manipulation. *arXiv preprint arXiv:2602.08557*, 2026.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 34556–34583, 2023.

Petar Veličković, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. Softmax is not enough (for sharp size generalisation). In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61190–61211, 2025.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2021.

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, 2021.

Dongjie Yu, Kun Lei, Zhennan Jiang, Jia Pan, and Huazhe Xu. Beyond action residuals: Real-world robot policy steering via bottleneck latent reinforcement learning. *arXiv preprint arXiv:2605.19919*, 2026. arXiv:2605.19919.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=e5jGTEiJMT.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations. In *International Conference on Learning Representations (ICLR)*, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, pp. 3391–3401, 2017.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations (ICLR)*, 2019.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Ayzaan Wahid, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning (CoRL)*, volume 155 of *Proceedings of Machine Learning Research*, pp. 726–747, 2020.

Xiang Zhang, Siddarth Jain, Baichuan Huang, Masayoshi Tomizuka, and Diego Romeres. Learning generalizable pivoting skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Roland S. Zimmermann, Sjoerd van Steenkiste, Mehdi S. M. Sajjadi, Thomas Kipf, and Klaus Greff. Sensitivity of slot-based object-centric models to their number of slots. *arXiv preprint arXiv:2305.18890*, 2023. arXiv:2305.18890.

A Experimental setup

Robot and simulation. The robot is a three axis gantry with a three joint wrist and a parallel-jaw gripper (8 joints, 7 actuators). It is commanded at 20 Hz with a 7-D action (three translations, three wrist rotations, one grip command). The scene is simulated in MuJoCo (Todorov et al., 2012) on the GPU with 2 ms physics substeps and 2048 parallel environments. Objects are cubes of 16 cm edge length and 0.2 kg in a 2.7 m $\times$ 2.7 m workspace. The Panda experiments of Section 4.2 use a simulated 7-DoF Franka Panda arm commanded by joint velocities, with cubes of 4 cm edge length, in two embodiments, one with a parallel-jaw gripper and a binary grip channel (8-D action) and one with a rigid sphere-tip in place of the jaws (7-D action) that pushes cubes across the table and never lifts them.

The skill μ. μ moves one cube from any resting pose to any goal pose, including the goal orientation. We train it with TD7 (Fujimoto et al., 2023) in a single-object scene with constrained sampling and reinforcement learning (CSRL) (Toussaint et al., 2026). Its input is the 65-D state vector of joint and cube positions, joint and cube velocities, the servo errors of the 7 actuators, the positions of four cube corners, the goal-relative rotation and the corner to goal offsets, and its action is the 7-D command above. μ is frozen in every experiment except the fine-tuning baseline. On its own task it succeeds in 0.97 of 64 evaluation episodes at the checkpoint we freeze. Each Panda embodiment has its own skill, trained the same way in its own single-object scene. The gripper skill places one

cube as the gantry skill does, and the pusher skill brings one cube to a position and yaw goal in the plane, where the cube's fourfold symmetry identifies yaws a quarter turn apart.

Tasks. The six tasks share one goal predicate and differ in the goal layout. The pyramid, square, line and the wall grow with N, and the stacks are repeated at their target height. Every task is trained and evaluated in the 2.7 m $\times$ 2.7 m workspace, except that the line is evaluated in a workspace widened to fit thirty cubes. A cube counts as placed when it is within 3 cm of its goal position (on the Panda, whose cubes are 4 cm, within 1.75 cm with the gripper and 1.25 cm with the pusher) and rests on its intended support (the floor or the cube beneath it). Initial layouts are drawn from a pool of sampled resting configurations, and every count N is built as a scene of exactly N cubes. An episode lasts $20N$ seconds.

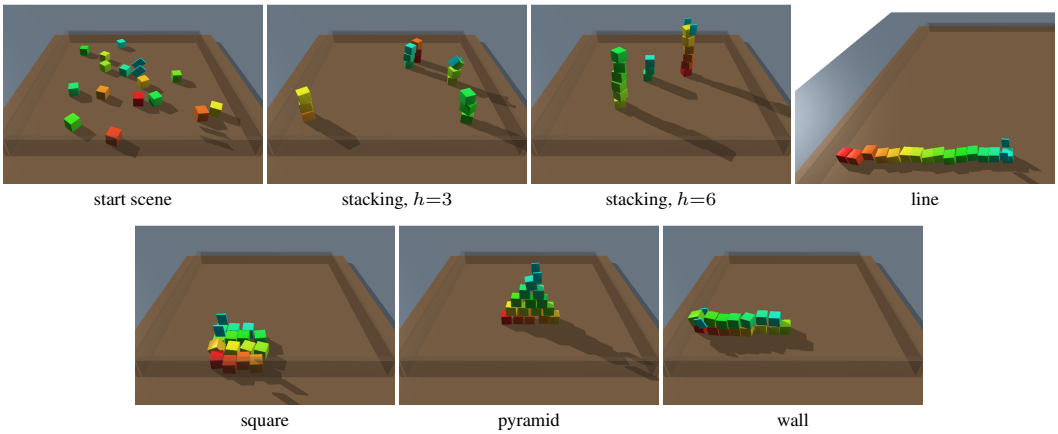

Figure 5: Renderings of the start scene and the final frames of all six gantry tasks.

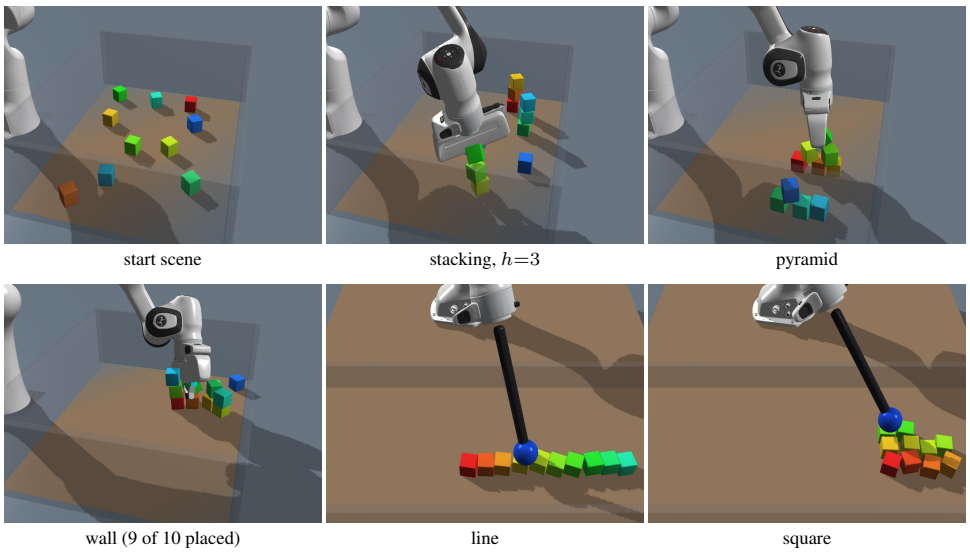

Figure 6: Renderings of the start scene and the final frames of all five Panda tasks.

Training. For every task we train one warping policy and one selection policy with PPO on scenes whose object count is drawn uniformly from 1 to N_{train}, with $N_{\text{train}} = 7$ for cube stacking with either height, 6 for the line and the pyramid, 9 for the square and 8 for the wall on the gantry, and 7 for cube stacking, 6 for the pyramid, 8 for the wall and 9 for the line and the square on the Panda. A gantry run uses 5.0×10^8 environment steps (2048 environments $\times$ 96 steps $\times$ 2543 iterations) on one A100 GPU within 10 h, and every Panda run uses 7.9×10^8 (4000 iterations) environment steps within 18 h. The warping policy uses a two layer transformer encoder with width 128 and four heads, followed by a two layer transformer decoder with one query per channel group that emits one

categorical distribution per warp parameter. The warp edits 55 channels of μ with one multiplier and one offset each, i.e., 110 parameters. The 47 input channels are the presented goal position and orientation (3 + 3), the object position and orientation (3 + 3), the object linear and angular velocity (3 + 3), the robot joint positions and velocities (9 + 9), the object to goal offset (3) and the servo errors (8). The 8 output channels are the 7 motion commands and the grip command. The remaining inputs of μ, such as the cube corners, are computed from the warped channels. The channel counts fit the Panda gripper, the largest of our robots, and on the gantry the 4 surplus channels stay at the identity. Multipliers take values on the 25 point grid of Figure 2, and offsets take values on a 21 point grid of multiples of a 1.5 cm step, up to 32 steps. Every parameter distribution is initialized with 80% of its mass on the identity value. PPO uses a learning rate of 10^{-3} for the warping policy and 3×10^{-4} for the selection policy, both annealed by a cosine schedule to a tenth, a clip ratio of 0.2, entropy coefficients of 5×10^{-4} and 3×10^{-3}, GAE with $\lambda_{\mathrm{GAE}} = 0.95$, and four epochs over minibatches of 8192 samples. The selection policy scores are drawn through a softmax during training. We train three seeds for every evaluation.

Reward. The task reward is sparse and requires long successful manipulation trajectories before it pays a reward. Placed cubes can be knocked over again, so keeping an arrangement must be worth as much as extending it. Otherwise, a fast placement that knocks down other cubes would earn more return than a slower one that keeps them in place. Episode lengths grow with N, so the balance between these terms must not shift with the count. R_N is built from the goal completion predicate alone. Every control step pays a reward of $w(1-\gamma)/2$ times the fraction of currently placed cubes, completion pays $w\gamma$ once, and the first placement of each cube pays bw/N once, with $w = 4$ and $b = 0.65$. This per step reward makes holding a partial arrangement valuable, so a knockdown lowers the return without a dedicated penalty term. The completion bonus keeps the actual objective dominant in the return. The first-placement bonus makes the earliest success of every cube visible to the policy gradient, and its division by N fixes its total mass across counts. The discount is set per episode to $\gamma = 1 - 3/D$, with D the number of control steps in the episode, so the terminal bonus is discounted by e^{-3} and the ratio of per step reward to bonus is the same at every N. The same constants are used for every task in Section 4.

Evaluation. Initial layouts come from a held-out pool whose seed differs from the training pool. For every count, $N \in \{1, \ldots, 30\}$ on the gantry and the Panda pusher square, up to 20 on the pusher line and up to 15 on the Panda gripper tasks, we build scenes of exactly N cubes and run the first episode of each of 20 scenes with the deterministic (argmax) warping and selection policies and the training clock of 20 s per cube. Alongside whole-task success we record, per count, the mean peak number of simultaneously placed cubes, the retention (placed cubes at the end divided by the peak), the number of placed to unplaced transitions, the fraction of the episode elapsed at the last new placement and the fraction of episodes that end by timeout.

B Full results

Table 2 reports the whole-task success of WAVE on the gantry per task and count and Table 3 on the Panda arm, as the mean and the range over seeds.

C Real-world Panda experiments

We deployed a cube stacking policy trained in simulation on a physical Panda arm. The policies are state-based, so we estimate the poses of the $4 \times 4 \times 4$ cm cubes from AprilTag fiducials tracked by four calibrated RGB cameras.

We evaluate the policy on ten consecutive rollouts of building a three-cube and a two-cube tower from five cubes (Table 5, Figure 9). The two base cubes start in their goal slots, so three cubes have to be picked and placed. The initial layout differs between runs. In 8 of the 10 runs all five cubes reached their goals (95% Wilson interval 49–94%), after 410 ± 248 policy steps (92 ± 54 s) on average. In 5 of the 10 runs the policy also released the last cube and both towers stood at the end. The gap between the two rates comes from the final release, which the policy never learned. In training, an episode ends as a success as soon as every cube is within tolerance, even while the last cube is still in the gripper. On the robot the rollout continues, and without a remaining objective the policy keeps the cube near its goal (runs 6, 7 and 10) or lets go only after a long delay (29 s in

Table 2: Whole task success of WAVE on the gantry per task and object count, the first episode of each of 20 scenes per count and 20 s per cube, mean with the minimum and maximum over three seeds in parentheses. Cube stacking builds stacks of height h. The line is evaluated in a workspace widened to fit thirty cubes, and the wall is two courses high and grows longer with the count.

N	stacking, $h = 3$	stacking, $h = 6$	line	square	pyramid	wall
1	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)
3	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)
5	0.95 (0.85–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	0.98 (0.95–1.00)
7	1.00 (1.00–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	0.98 (0.95–1.00)	0.92 (0.85–0.95)	0.98 (0.95–1.00)
10	0.98 (0.95–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)	0.97 (0.95–1.00)	0.90 (0.80–0.95)	0.93 (0.85–1.00)
12	0.98 (0.95–1.00)	0.85 (0.80–0.95)	0.97 (0.95–1.00)	0.98 (0.95–1.00)	0.80 (0.80–0.80)	0.95 (0.90–1.00)
15	0.92 (0.85–1.00)	0.72 (0.55–0.85)	1.00 (1.00–1.00)	0.97 (0.90–1.00)	0.60 (0.45–0.70)	0.85 (0.70–0.95)
20	0.95 (0.90–1.00)	0.13 (0.10–0.20)	0.93 (0.90–1.00)	0.85 (0.65–0.95)	0.35 (0.20–0.45)	0.67 (0.45–0.80)
25	0.63 (0.45–0.80)	0.00 (0.00–0.00)	0.97 (0.95–1.00)	0.77 (0.65–0.85)	0.20 (0.15–0.25)	0.63 (0.40–0.75)
28	0.53 (0.35–0.65)	0.00 (0.00–0.00)	0.92 (0.85–0.95)	0.80 (0.60–1.00)	0.10 (0.05–0.15)	0.48 (0.10–0.75)
30	0.48 (0.30–0.60)	0.00 (0.00–0.00)	0.90 (0.80–1.00)	0.55 (0.45–0.60)	0.08 (0.05–0.15)	0.38 (0.05–0.85)

Table 3: Whole task success of WAVE on the Panda arm per task and object count, the same protocol as Table 2. Cube stacking ($h = 3$), the pyramid and the wall use the parallel-jaw gripper and the line and the square the sphere-tip pusher. A dash marks a count the robot was not evaluated on.

	gripper			pusher	
N	stacking, $h = 3$	pyramid	wall	line	square
1	1.00 (1.00–1.00)	1.00 (1.00–1.00)	0.98 (0.95–1.00)	1.00 (1.00–1.00)	1.00 (1.00–1.00)
3	0.92 (0.85–1.00)	0.90 (0.85–0.95)	0.70 (0.65–0.75)	1.00 (1.00–1.00)	1.00 (1.00–1.00)
5	0.87 (0.75–0.95)	0.47 (0.15–0.65)	0.23 (0.10–0.40)	1.00 (1.00–1.00)	1.00 (1.00–1.00)
7	0.63 (0.60–0.70)	0.28 (0.10–0.55)	0.03 (0.00–0.10)	0.98 (0.95–1.00)	0.92 (0.75–1.00)
9	0.38 (0.20–0.60)	0.17 (0.10–0.20)	0.00 (0.00–0.00)	0.95 (0.90–1.00)	0.60 (0.50–0.70)
12	0.03 (0.00–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.80 (0.75–0.85)	0.35 (0.20–0.55)
15	0.03 (0.00–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.78 (0.75–0.80)	0.10 (0.00–0.15)
20	–	–	–	0.62 (0.50–0.70)	0.00 (0.00–0.00)
25	–	–	–	–	0.00 (0.00–0.00)
30	–	–	–	–	0.00

run 2, about two minutes in run 3). Requiring the release for success during training would remove this failure mode, but we leave this for future work. The two failed runs were stopped at a joint limit with four of five cubes placed (run 5) and at the time limit (run 8). Grasps often needed several attempts. Of the 78 close commands, 68% held a cube, and the policy recovered from failed grasps by trying again. Runs 4 and 9 needed the minimum of three grasps. Runs 6, 7 and 10 needed 9–11. The policy releases a cube a median of 13 mm above its goal slot, a behavior learned in simulation, where the jaws open slowly. Together with gripper stiction on the real system, this was the main reason cubes toppled.

We also deployed a policy fine-tuned for five-cube towers. It built a five-high tower on the real robot (Figure 8), but we did not evaluate this task systematically.

State estimation added a further source of error. We applied no temporal filtering, so tracking jitter, occluded fiducials and camera calibration errors entered the observation of the policy directly. The real gripper also has velocity limits and more stiction than the simulated one. A motion-capture system with temporal filtering and domain randomization of gripper friction and stiction would address both sources of error.

D BASELINES

Every baseline of Section 4.3 runs on the same gantry and task as WAVE, with the same reward unless noted, the same mixed training counts (one to six cubes on the line, one to seven on cube stacking) and 5×10^8 environment steps over 2048 parallel scenes, and every baseline that uses a

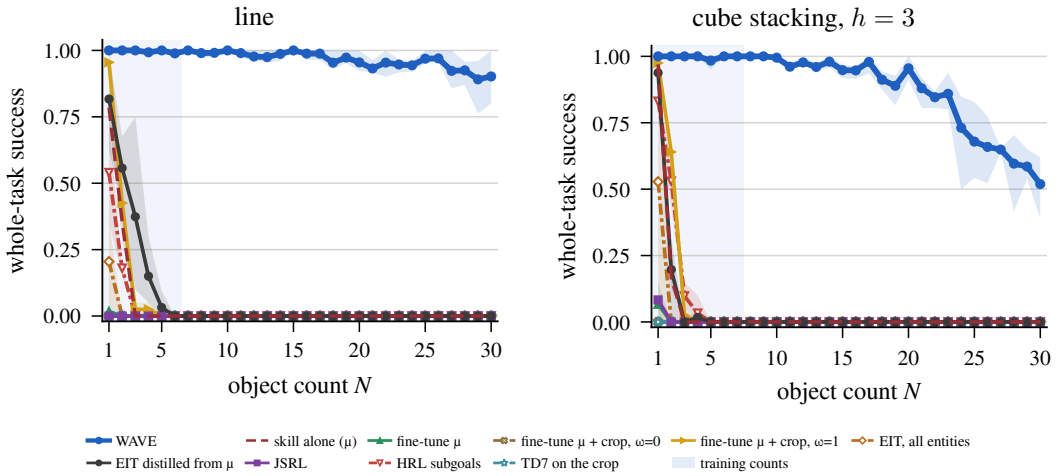

Figure 7: Whole task success for WAVE and all nine baselines of Section 4.3, left on the line and right on cube stacking. The curves are the mean over three seeds and the bands the seed range.

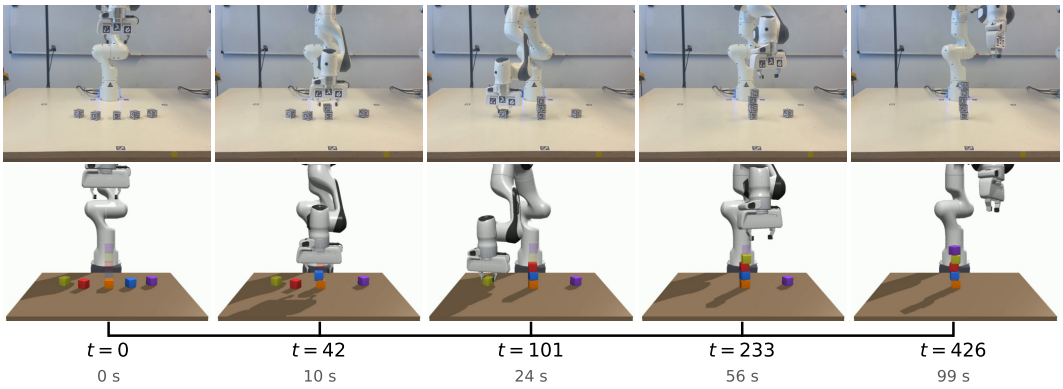

Figure 8: Five-high tower built on the real robot.

skill uses the same checkpoint of μ. They are evaluated like WAVE from one to thirty cubes over three seeds.

We supply a scripted selection policy that chooses only objects with goals that are currently fulfillable. For example, in cube stacking only cubes in the bottom layer or on an already placed cube can be selected. The selection then chooses the fulfillable object nearest to the gripper. WAVE does not use a scripted selection policy, so the baselines train on an easier task. All baselines use this selection policy, except for the EIT and HRL baselines.

Table 4 compares WAVE with the baselines of Section 4.3 on the line and on cube stacking ($h = 3$), with the mean and the range over all seeds. Table 1 in Section 4.3 shows six of these baselines. Plain fine-tuning of μ, fine-tuning on the crop without the anchor ($\omega = 0$) and JSRL are reported only here, in Table 4 and Figure 7.

Skill alone (μ) runs the frozen skill with no training on the selected object. *Fine-tuning* μ keeps the same selection but continues to train μ's own actor and critic on the multi-object reward. *EIT* is the Entity Interaction Transformer of Haramati et al. (2024) with every object and goal as a token, trained with TD3 and hindsight relabeling, with the same hyperparameters as reported in their paper. *EIT distilled from* μ first imitates the actions of μ, whose rollouts also fill its replay buffer. It is then trained like *EIT*, with an added loss that keeps the actor close to μ. In *JSRL* (Uchendu et al., 2023) μ is used as a guide policy that acts the first 90% of every training episode and a TD7 exploration policy initialized from μ's weights acts the rest of the episode. The control portion is transferred gradually to the exploration policy, until at the end of training only the exploration policy acts on the

Table 4: Whole task success per object count for WAVE and the baselines of Section 4.3 (Figure 7), the mean over three seeds with the minimum and maximum over the seeds in parentheses below it. Every method trains on one to six cubes on the line and one to seven on cube stacking. Bold marks the highest mean per task and count.

N	1	2	3	4	5	6	10	30
line								
WAVE	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**0.98** (0.95–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**0.90** (0.80–1.00)
skill alone (μ)	0.79 (0.75–0.83)	0.35 (0.29–0.43)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
fine-tuning μ	0.02 (0.00–0.05)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
on the crop, $\omega = 0$	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
on the crop, $\omega = 1$	0.96 (0.93–0.98)	0.43 (0.40–0.45)	0.03 (0.00–0.05)	0.03 (0.00–0.05)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
EIT (Haramati et al., 2024)	0.20 (0.05–0.36)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
EIT distilled from μ	0.82 (0.62–0.99)	0.56 (0.43–0.68)	0.37 (0.10–0.75)	0.15 (0.05–0.30)	0.03 (0.00–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
JSRL (Uchendu et al., 2023)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
HRL subgoals	0.54 (0.35–0.73)	0.18 (0.15–0.20)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
TD7 on the crop	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
cube stacking, $h = 3$								
WAVE	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**0.95** (0.85–1.00)	**1.00** (1.00–1.00)	**0.98** (0.95–1.00)	**0.48** (0.30–0.60)
skill alone (μ)	0.97 (0.95–0.99)	0.17 (0.13–0.20)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
fine-tuning μ	0.07 (0.00–0.15)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
on the crop, $\omega = 0$	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
on the crop, $\omega = 1$	0.98 (0.95–1.00)	0.64 (0.61–0.67)	0.03 (0.00–0.05)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
EIT (Haramati et al., 2024)	0.53 (0.00–0.90)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
EIT distilled from μ	0.94 (0.85–0.99)	0.20 (0.00–0.32)	0.00 (0.00–0.00)	0.02 (0.00–0.05)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
JSRL (Uchendu et al., 2023)	0.08 (0.05–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
HRL subgoals	0.83 (0.77–0.91)	0.53 (0.43–0.59)	0.10 (0.05–0.15)	0.03 (0.00–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
TD7 on the crop	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)

robot. *HRL subgoals* is a hierarchy with the frozen μ as its low level, which keeps WAVE's selection policy and its crop χ, and in place of the warp a TD7 manager that reads the crop and proposes goals for μ to pursue. *TD7 on the crop* trains a TD7 policy from random initialization on the same crop χ as WAVE, with the scripted selection and without μ.

Fine-tuning μ on the crop gives the fine-tuned μ the view of the other objects that plain fine-tuning lacks. The crop χ is the same as for the warping policy (Section 3.4). This crop is passed to the first layer of μ's actor as additional input, through new weights initialized at zero, so that the actor outputs exactly $\mu(\sigma)$ at the start of training. The critic is extended in the same way and reads (σ, χ, a). For the first 10% of training only the critic is trained and the actor stays fixed, so that the critic has learned the multi-object reward before it updates the actor. The actor then minimizes the TD7 loss with an anchor to the frozen skill, as in TD3+BC (Fujimoto & Gu, 2021),

$$\mathcal{L}(\theta) = -\frac{Q(\sigma, \chi, \pi_\theta(\sigma, \chi))}{\overline{|Q|}} + \omega \left\| \pi_\theta(\sigma, \chi) - \mu(\sigma) \right\|^2,$$

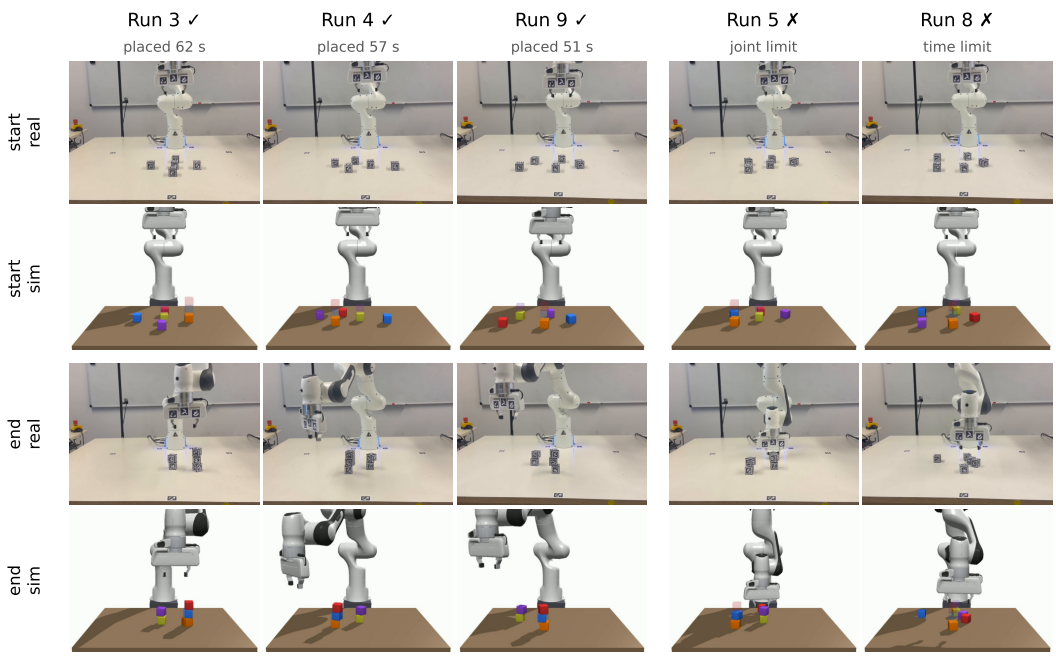

Figure 9: Selected real rollouts of building a three-cube and a two-cube tower (columns) with the same rollouts replayed in simulation from the logged joint states, gripper widths and tracked cube poses. The rows show the start and end on the real robot and in the replay. Runs 3, 4 and 9 succeeded. Run 5 stopped at a joint limit and run 8 at the time limit.

	success		steps to (time [s])					
run	placed	released	placed	released	grasps	closes	cubes	stop
1	✓	✓	373 (82)	472 (102)	5	5	5/5	
2	✓	✓	235 (53)	402 (88)	4	5	5/5	
3	✓	✓	276 (62)	873 (184)	5	5	5/5	
4	✓	✓	254 (57)	259 (58)	3	4	5/5	
5	–	–	–	–	2	8	4/5	joint limit
6	✓	–	300 (66)	–	9	11	5/5	
7	✓	–	844 (194)	–	9	16	5/5	
8	–	–	–	–	2	6	2/5	time limit
9	✓	✓	234 (51)	252 (55)	3	3	5/5	
10	✓	–	764 (175)	–	11	15	4/5	
rate	8/10	5/10						
95% CI	49–94%	24–76%						
mean			410 ± 248	452 ± 254	5.3	7.8	4.5/5	

Table 5: Ten consecutive real-robot rollouts of the task of building a three-cube and a two-cube tower from five cubes (two base cubes start in their slots, three have to be moved). *Placed* means all five cubes are within tolerance of their goals. *Released* additionally requires that the last cube was let go and the structure stands at the end of the run. Steps are policy actions at 5 Hz, with the time in seconds in parentheses and means over the successful runs. *Grasps* counts grasps that held a cube, *closes* counts gripper close commands and *cubes* counts cubes within tolerance at the end.

where $\overline{|Q|}$ is the mean absolute value of Q over the batch, which makes ω independent of the scale of the reward. With $\omega = 0$ the actor is trained on the reward alone. With $\omega = 1$, staying close to the action of μ is weighted about as much as improving the normalized value. This baseline keeps the scripted selection, and like μ inside WAVE it chooses an action at every control step.

E Overview of published methods and results

Table 6: Training conditions and best reported results of the published methods cited in Section 2 that learn the manipulation itself, in a physics simulator or on a real robot, with a goal pose for every object. All values are taken from their reports. The gray series of Figure 3 are the rows that report whole-task success per count.

Method	*Trains on*	*Task*	*Counts (train → test)*	*Best reported*
Li et al. (2020)	MuJoCo Fetch, privileged state, attention graph network	stacking every block	curriculum 1–6 → 9	0.93 at 4, 0.84 at 5, 0.75 at 6, separate policies per count
ECRL (Haramati et al., 2024)	IsaacGym, two RGB views to latent particles, end effector position deltas on a pusher	planar pushing, 2-D goals	3 → 1–6	0.84 at 3, 0.40 at 6, and a 12 object sort is shown qualitatively only
HECRL (Haramati et al., 2026)	MuJoCo adapted from OGBench, state and image variants	rearrangement, stacking	3 → 1–6 (stacking 2–4)	0.83 at 3 and 0.26 at 6 on rearrangement from state, 0.64 and 0.10 from images
EC-Diffuser (Qi et al., 2025)	IsaacGym, two RGB views to latent particles, behavior cloning	pushing cubes and T blocks	3 → 6	0.24 at 6, 0.83 at 3 by pushing cubes
SMORL (Zadaianchuk et al., 2021)	MuJoCo, RGB to object slots	pushing and rearranging pucks	per count, 1–2 from pixels and 2–4 from state	— (transfer only downward, 2 to 1)
PAIR (Li et al., 2022)	PyBullet, privileged state, discretized actions	six-cube stacking	6 → 6	0.955 at 6, where success counts one target cube among six
Lanier et al. (2019)	MuJoCo Fetch, privileged state, binary or incremental reward	stacking	per count, 2–4	0.98 at 3 and 0.79 at 4 with the incremental reward, 0.95 and 0.00 at 4 with the binary one, all whole-task success
LRN (Mambelli et al., 2022)	CausalWorld TriFinger, privileged state	pushing and lifting one cube among distractors	3 cubes → 1–10 cubes	— (distractors, not goal objects)
OGBench (Park et al., 2025a)	offline datasets, state, UR5e, one policy per count	five goals per count, pick and place, cycles and one stack from two cubes on	per count, 1–4	0.68, 0.40, 0.04 and 0.00 at 1 to 4 over the five goals; on the stack goal 0.23 at 2 and 0.00 at 3 and 4, best of six baselines
SHARSA (Park et al., 2025b)	OGBench cube octuple, state, up to 10^9 transitions, hierarchical	five goals on eight cubes, three rearrangements and two stacks	8 → 8	0.36 at 8 over the five goals with 10^8 transitions and 0.21 with 10^9; 0.00 and 0.01 on the two stacking goals with 10^8, 0.00 on both with 10^9
DQC (Li et al., 2025)	same benchmark and data as SHARSA, flat action chunking	as SHARSA	8 → 8	0.34 at 8 over the goal types, no subtask breakdown
Lampe et al. (2024)	ten real Sawyer arms, three workspace cameras with force and torque sensing	stacking diverse shapes	3 in scene, a two object stack, fixed	0.96 at the fixed count

Tables 6 and 7 list, for every count generalization method cited in Section 2 and every gray series of Figure 3, what it trains on, the counts it trains and tests with, and the best result it reports at its highest evaluated count. A dash marks a quantity the paper does not report in a comparable form.

Every result at a high object count uses privileged state. The only pixel-based method, ECRL, pushes on a plane without gripper actuation, and its successor HECRL, which grasps and stacks, completes 0.10 at six objects from images. Several of the best numbers hold under a criterion less strict than whole-task success, namely one target cube of six for PAIR and the fraction of goals filled for IL-GNN. Methods that learn the manipulation itself, in a physics simulator or on a real robot and with a goal pose for every object, publish whole-task success at up to eight objects. The offline methods on cube octuple reach that count with 0.21 to 0.36 over the five goals and at most 0.01 on the stacking

Table 7: The highest object counts we know of for methods that relax at least one condition of Table 6, with the same columns. All values are taken from their reports.

Method	*Trains on*	*Task*	*Counts (train → test)*	*Best reported*
Janisch et al. (2020)	symbolic block world, relational graph, no physics simulation	goal-conditioned reconfiguration	5 → 2–30	0.78 at 20 and 0.54 at 30
IL-GNN (Lin et al., 2022)	PyBullet and a real Franka, privileged object graph, imitation over a predefined pick-and-place primitive	stacking, rearrangement	3–4 → 9 with data	0.75 averaged over counts up to 40, no per count value above 9, manipulation supplied by the primitive
Blocks Assemble! (Ghasemipour et al., 2022)	MuJoCo, privileged state, virtual grippers that move blocks without grasping	magnetic block blueprints	2–16 → 12–16 held out	1.00 at 6, 0.99 at 12, 0.91 at 16, success up to a rigid motion of the structure
Transporter (Zeng et al., 2020)	PyBullet, RGB-D, scripted two pose pick and place	palletizing 18 boxes into a zone	18 → 18	0.98 at 18 with 1000 demonstrations, scored as the fraction of box volume inside the zone, no per box pose
QT-Opt (Kalashnikov et al., 2018)	real arms, RGB, closed-loop grasping	clearing a bin	4–10 → 28	0.76 over the first 30 grasps of a 28 object bin and 0.96 with replacement, no goal pose
Push Anything (Bui et al., 2026)	real arm, contact implicit MPC, controller not learned	pushing to per-object goal poses	1–4	0.98 at 2 and 0.79 at 4 within 2 cm and 0.1 rad

goals. Every higher count we know of relaxes one of these conditions (Table 7). Blocks Assemble! moves blocks with virtual grippers and scores a blueprint up to a rigid motion. Transporter and IL-GNN execute a scripted primitive between network-chosen poses and score partial credit. Janisch et al. (2020) act in a symbolic block world without physics simulation. Bin clearing policies such as QT-Opt remove objects that have no goal pose. Push Anything reaches four objects with a model-based controller. Li et al. (2020) test up to nine blocks, split over several towers, and show the result as a figure without values, and BuilderBench (Ghugare et al., 2025) reports that its evaluated agents solve two of its twenty-seven hard block structures and that no tabula rasa RL baseline reaches non-zero success.

F Ablations and measured constants

We ablate WAVE one element at a time on three of the gantry tasks: cube stacking ($h = 3$), whose stacks are independent local structures, the square, whose goals lie flat on the table and constrain each other only through space, and the pyramid, whose upper goals cannot be filled before the cubes beneath them are in place. Every ablation keeps the full method and the training budget of Section 4 and changes exactly one part, either the selection rule, the side of μ's interface the warp may edit, or the component of the warp the policy may use. Each ablation is trained from scratch with three seeds, with one to seven cubes on cube stacking, one to nine on the square and one to six on the pyramid.

Table 8 shows the results. The WAVE rows repeat Table 2 for the same task. We summarize each ablation by the mean of its success over the counts.

Table 8: Whole task success per object count for the ablations, on cube stacking ($h = 3$), the square and the pyramid, the mean over three seeds with the minimum and maximum over the seeds in parentheses below it. MEAN is the mean over the counts to thirty cubes. WAVE is the three seed column of Table 2 for the same task. Bold marks the highest mean per task and count.

N	3	5	7	10	15	20	25	30	MEAN
cube stacking, $h = 3$, trained on one to seven									
WAVE	**1.00** (1.00–1.00)	0.95 (0.85–1.00)	**1.00** (1.00–1.00)	0.98 (0.95–1.00)	0.92 (0.85–1.00)	**0.95** (0.90–1.00)	0.63 (0.45–0.80)	0.48 (0.30–0.60)	0.82 (0.78–0.86)
nearest-first selection	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.97 (0.95–1.00)	0.97 (0.95–1.00)	**0.95** (0.95–0.95)	0.93 (0.85–1.00)	**0.75** (0.70–0.80)	**0.60** (0.45–0.75)	**0.85** (0.83–0.88)
random selection	0.30 (0.20–0.40)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
input channels only	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.93 (0.90–0.95)	0.85 (0.85–0.85)	0.70 (0.65–0.75)	0.42 (0.20–0.65)	0.81 (0.77–0.85)
output channels only	**1.00** (1.00–1.00)	0.97 (0.95–1.00)	0.82 (0.80–0.85)	0.55 (0.55–0.55)	0.23 (0.20–0.25)	0.12 (0.10–0.15)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.22 (0.19–0.26)
multipliers only	0.97 (0.95–1.00)	0.97 (0.95–1.00)	**1.00** (1.00–1.00)	0.90 (0.90–0.90)	0.82 (0.75–0.90)	0.78 (0.65–0.90)	0.47 (0.40–0.55)	0.25 (0.15–0.35)	0.65 (0.58–0.71)
offsets only	0.97 (0.95–1.00)	**1.00** (1.00–1.00)	0.93 (0.90–0.95)	0.88 (0.85–0.90)	0.62 (0.55–0.70)	0.60 (0.55–0.65)	0.23 (0.15–0.30)	0.10 (0.05–0.15)	0.52 (0.50–0.54)
square, trained on one to nine									
WAVE	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.98 (0.95–1.00)	**0.97** (0.95–1.00)	0.97 (0.90–1.00)	0.85 (0.65–0.95)	0.77 (0.65–0.85)	0.55 (0.45–0.60)	0.87 (0.77–0.92)
nearest-first selection	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	**0.97** (0.95–1.00)	**1.00** (1.00–1.00)	**0.93** (0.90–0.95)	**0.93** (0.85–1.00)	**0.95** (0.90–1.00)	**0.97** (0.94–0.99)
random selection	0.80 (0.80–0.80)	0.30 (0.25–0.35)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
input channels only	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.97 (0.95–1.00)	**0.97** (0.95–1.00)	0.88 (0.85–0.90)	0.75 (0.65–0.85)	0.60 (0.45–0.75)	0.60 (0.60–0.60)	0.76 (0.71–0.80)
output channels only	**1.00** (1.00–1.00)	0.82 (0.65–1.00)	0.68 (0.35–1.00)	0.33 (0.00–0.65)	0.20 (0.00–0.40)	0.05 (0.00–0.10)	0.03 (0.00–0.05)	0.00 (0.00–0.00)	0.14 (0.01–0.28)
multipliers only	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.95 (0.95–0.95)	0.95 (0.90–1.00)	0.95 (0.95–0.95)	0.75 (0.70–0.80)	0.78 (0.65–0.90)	0.60 (0.55–0.65)	0.80 (0.77–0.82)
offsets only	**1.00** (1.00–1.00)	0.97 (0.95–1.00)	0.93 (0.90–0.95)	0.78 (0.70–0.85)	0.72 (0.55–0.90)	0.45 (0.30–0.60)	0.20 (0.10–0.30)	0.17 (0.10–0.25)	0.44 (0.34–0.54)
pyramid, trained on one to six									
WAVE	**1.00** (1.00–1.00)	**1.00** (1.00–1.00)	0.92 (0.85–0.95)	**0.90** (0.80–0.95)	0.60 (0.45–0.70)	0.35 (0.20–0.45)	0.20 (0.15–0.25)	**0.08** (0.05–0.15)	0.47 (0.46–0.49)
nearest-first selection	**1.00** (1.00–1.00)	0.93 (0.90–0.95)	0.50 (0.10–0.90)	0.47 (0.10–0.85)	0.25 (0.00–0.50)	0.15 (0.00–0.30)	0.03 (0.00–0.05)	0.00 (0.00–0.00)	0.19 (0.01–0.37)
random selection	0.93 (0.90–0.95)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)
input channels only	0.97 (0.95–1.00)	0.55 (0.30–0.80)	0.35 (0.15–0.55)	0.17 (0.00–0.35)	0.05 (0.00–0.10)	0.03 (0.00–0.05)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.07 (0.01–0.14)
output channels only	0.95 (0.95–0.95)	0.75 (0.75–0.75)	0.57 (0.50–0.65)	0.42 (0.40–0.45)	0.20 (0.15–0.25)	0.05 (0.00–0.10)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.15 (0.11–0.18)
multipliers only	0.97 (0.95–1.00)	**1.00** (1.00–1.00)	**0.95** (0.90–1.00)	0.88 (0.85–0.90)	**0.72** (0.60–0.85)	**0.53** (0.40–0.65)	**0.33** (0.30–0.35)	0.07 (0.05–0.10)	**0.54** (0.52–0.57)
offsets only	0.80 (0.65–0.95)	0.38 (0.00–0.75)	0.38 (0.00–0.75)	0.15 (0.00–0.30)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.00 (0.00–0.00)	0.05 (0.00–0.11)

F.1 The selection rule. Both the random and the nearest-first rule consider only objects whose goal is currently fulfillable, like the scripted selection of the baselines (Appendix D). The learned selection policy of equation 2 has no such filter. Random selection draws uniformly from these objects. Nearest-first takes the one whose position is closest to the end effector in the plane, an analytic rule that needs no learning and no reward. Figure 10 shows the three tasks.

Random selection has the largest effect in the ablations. Its mean success is 0.00 on all three tasks, and no seed completes an episode from five cubes on cube stacking, nine on the square and five on the pyramid.

Nearest-first is handed a gate that encodes the support structure of the task, which goals can be filled at the moment, and this gate has to be scripted for every task. The learned selection policy receives no such gate and recovers the order of each task from the reward alone. On cube stacking the two lie within each other's seed range (mean 0.85 against 0.82). On the pyramid, whose upper goals

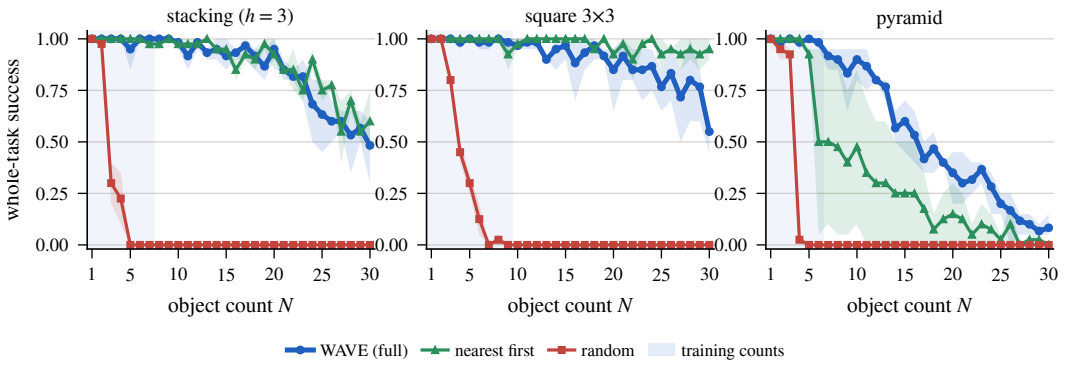

Figure 10: The selection rule, on cube stacking ($h = 3$), the square and the pyramid. Random selection draws uniformly from the objects whose goal is currently fulfillable. Nearest-first takes the fulfillable object closest to the end effector in the plane. The learned selection policy uses no such filter. Shading is the range over the ablation's three seeds, WAVE is its three seed curve, and the tinted band marks the task's training counts.

cannot be filled before the cubes beneath them are placed, the order is nontrivial and the learned selection policy is well ahead (0.47 against 0.19), while the mean success of the single nearest-first seeds ranges from 0.01 to 0.37. On the square, whose goals constrain each other only through space, nearest-first is ahead (0.97 against 0.87), mainly from twenty cubes on, and reaches 0.95 at thirty cubes where WAVE reaches 0.55.

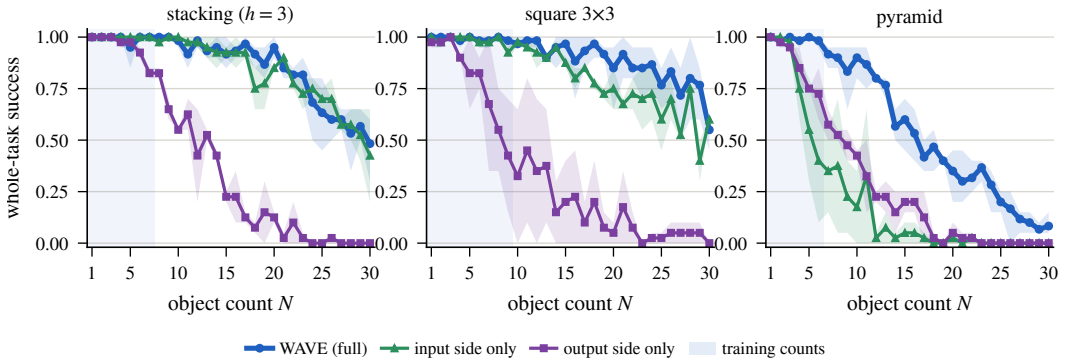

Figure 11: The side of μ's interface the warp may edit. Input channels only edits what μ observes and executes its action unedited (94 parameters). Output channels only edits what μ commands and shows it the scene as the presentation built it (16). Bands as in Figure 10.

F.2 The warp side. The warp emits an affine edit of all 55 channels of μ's interface, 110 parameters in total (Appendix A lists the channels). The input-only ablation restricts it to the 47 channels μ reads, so μ is shown an edited scene but its action is executed unedited (94 parameters). The output-only ablation restricts it to the 8 channels μ commands, so μ sees the scene as the presentation built it and only the command stream is edited (16 parameters). Figure 11 shows the three tasks.

On cube stacking, the input-only warp stays within the seed range of the full warp at nearly every count (mean 0.81 against 0.82). The output-only warp drops from seven cubes (0.82), halves by ten (0.55) and reaches a mean of 0.22. The square shows the same order (0.76 input only, 0.14 output only, 0.87 full warp). On these two tasks, editing the input of the skill is close to sufficient. An output side warp can slow, stop or replace the action of μ, but it cannot change the scene μ is shown, so μ still moves toward whatever stands in its way.

On the pyramid, the input-only warp reaches 0.07, below the output-only warp at 0.15, and both are far below the full warp at 0.47. Here cubes rest on each other, so the release itself has to change. A cube has to be set down on its support and let go without disturbing it, which is an edit of the output. Neither side is sufficient alone on this task, and only the full warp works on all three tasks.

In Appendix G, switching off either side of the full warp slows WAVE down, even on cube stacking. When both sides are trained together, the learned warp uses both. When only one side is trained, that side alone can often solve the task.

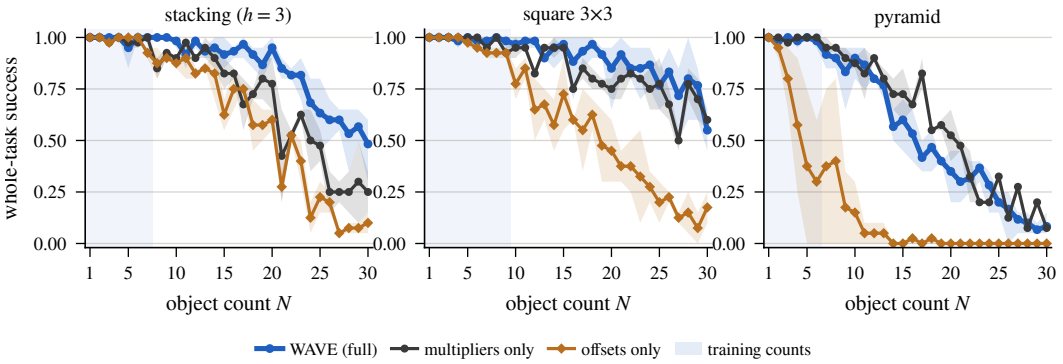

Figure 12: The component of the affine warp the policy may emit. Multipliers only fixes every offset at zero and offsets only fixes every multiplier at one, 55 parameters each against the full 110. Bands as in Figure 10.

F.3 The warp component. The affine warp $z \mapsto m \odot z + o$ has 55 multipliers and 55 offsets. The multipliers-only ablation fixes $o = 0$, and the offsets-only ablation fixes $m = 1$ (Figure 12).

Multipliers only reaches a mean success of 0.65, 0.80 and 0.54 on cube stacking, the square and the pyramid, against 0.82, 0.87 and 0.47 for the full warp. Offsets only reaches 0.52, 0.44 and 0.05. On the pyramid it already fails within the training counts (0.38 at five cubes).

A multiplier can scale or damp a channel, and at $m = 0$ take it over. An offset can shift a channel to any value, but it cannot make μ act more slowly or weakly. On the square and the pyramid the multipliers alone carry the extension. Cube stacking needs both halves.

F.4 Measured constants. We measure p_e and λ on the evaluation episodes of Table 2, and δ_{cov} on replays of the same trained policies. An attempt is the period in which one object stays selected (Section 3.2). p_e is the fraction of attempts that end with the selected cube placed. λ is the mean number of previously placed cubes an attempt unplaces, and $\lambda_{\max}$ the largest such number. δ_{cov} is the fraction of attempts whose crop has a nearest neighbor closer than any seen at the training counts. This reference is a minimum over fewer attempts than training saw, so δ_{cov} is an upper bound. Table 9 lists the values, and Section 3.6 discusses them.

Table 9: The three constants of equation 4 on cube stacking ($h = 3$, left, trained on one to seven cubes) and on the pyramid (right), accumulated per count over the evaluation episodes, with δ_{cov} from replays of the same trained policies, the mean over three seeds, $\rho = (1 - \delta_{\text{cov}})p_e - \lambda$ and the whole-task success at the same count.

cube stacking, $h = 3$

N	p_e	λ	δ_{cov}	ρ	success
3	0.177	0.150	0.0000	+0.027	1.00
7	0.221	0.185	0.0000	+0.036	1.00
10	0.225	0.200	0.0005	+0.025	0.98
20	0.220	0.209	0.0014	+0.011	0.95
30	0.209	0.201	0.0013	+0.008	0.48

pyramid

N	p_e	λ	δ_{cov}	ρ	success
3	0.168	0.159	0.0000	+0.009	1.00
7	0.189	0.184	0.0000	+0.005	0.92
10	0.177	0.190	0.0000	−0.013	0.90
20	0.172	0.202	0.0006	−0.031	0.35
30	0.162	0.197	0.0004	−0.035	0.08

G INTERVENTIONAL ESTIMATION OF WARP NECESSITY

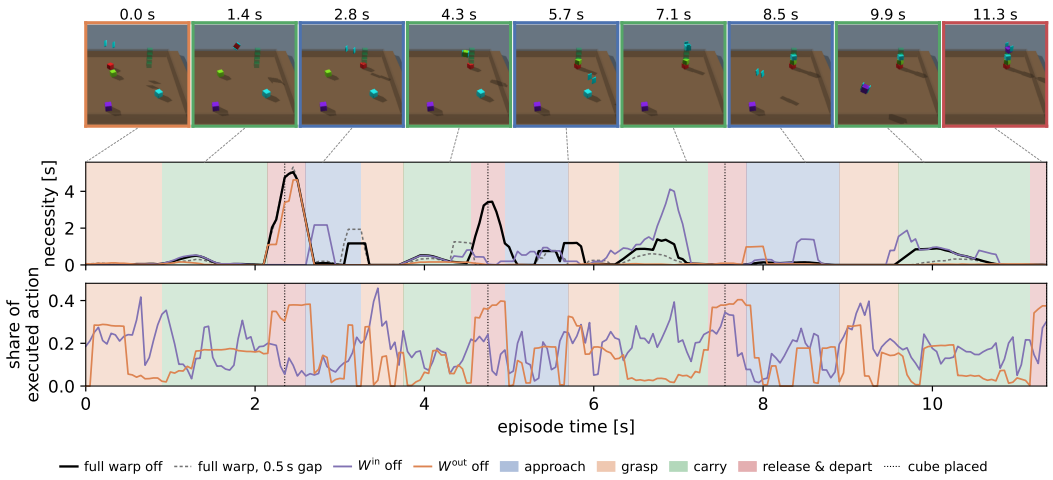

Figure 13: One cube stacking episode with $h = 6$, four cubes in one stack. Top, scene views framed in the color of the manipulation phase. Middle, necessity, the extra time the current phase takes when the full warp (black), only W_{in} (purple) or only W_{out} (orange) is held at the identity, or the full warp pauses for 0.5 s (dashed gray). Bottom, contribution, the share of the executed action that each side accounts for.

To measure when each side of the warp is needed, we split each episode into the phases approach, grasp, carry, release and depart, read from the simulator state. At every step we copy the simulator and continue the copies with one of four changes, six copies for each. Either the full warp, only W_{in} or only W_{out} is held at the identity, or the full warp pauses for 0.5 s. The *necessity* of a change is the extra time the current phase then takes, compared with six unchanged copies. A copy that does not finish the phase within 6 s, or knocks over a placed cube, counts as 6 s.

We also measure the *contribution* of each side, how much it changes the executed action. From the same state we compute the skill action a_μ, the action a_{in} under the input warp only, the action a_{out} under the output warp only, and the executed action a under both. The two-player Shapley value splits $a - a_\mu$ exactly between the sides,

$$\Delta_{\text{in}} = \tfrac{1}{2}\big[(a_{\text{in}} - a_\mu) + (a - a_{\text{out}})\big], \qquad \Delta_{\text{out}} = \tfrac{1}{2}\big[(a_{\text{out}} - a_\mu) + (a - a_{\text{in}})\big],$$

and the contribution of a side is $\|\Delta\|$ relative to the diagonal $2\sqrt{d_a}$ of the action range $[-1, 1]^{d_a}$.

Over 32 held-out episodes with four cubes, a phase takes about a second. Switching W_{out} off costs 1.05 s in release and depart, but only 0.27–0.34 s in the other phases. Switching W_{in} off costs 0.95–1.56 s in every phase.

Figure 13 shows one episode. W_{in} accounts for 20–40% of the executed action throughout, but its necessity stays low during approach and carry, where μ's own feedback absorbs a short gap. At the releases the necessity rises, and it comes almost entirely from W_{out}. μ is a grasp and hold policy and never lets go, so W_{out} overrides its action to release the cube (Section 4.4). The output side thus adds a missing behavior at a few critical moments, and the input side steers μ continuously.

H GUARANTEES

Setting. We recall the two MDPs of Section 3.2. The skill μ is trained in the single-object MDP $M_1 = (\mathcal{S}_1, \mathcal{A}, P_1, R_1, \mathcal{G})$, whose state holds one robot and one movable object and whose transition kernel P_1 carries a single-object state and an action to the next single-object state. We let P_1 act on a (state, goal) pair by carrying the state and leaving the goal fixed, and we write $P_1(s_1, a)$ for both. The warping policy ψ is trained in the N object MDP $M_N = (\mathcal{S}_N, \mathcal{A}, P_N, R_N, \mathcal{G}^N)$, in which the

same robot must bring N objects to their goal poses under P_N and the objects collide with, rest on and occlude each other, and μ is frozen throughout. Every statement below is about P_1 and about the presentations that π_N feeds to μ. μ is deterministic and has no memory, so its action at a control step depends on the presented state and goal only, and $\mathcal{D}_1$ is the set of (state, goal) pairs that its training visited. For a state s of M_N and the selected object x^*, the presentation $\sigma(s, x^*)$ is s with every other object deleted, expressed in the robot frame, together with the goal of x^*. The warp edits every channel of the presentation and of μ's action as $z \leftarrow mz + o$, and multipliers of position channels are measured from the current position of x^*, so a multiplier alone never moves the presented object.

When the warped skill still behaves like the original skill. It is not enough that every presented frame is a valid single-object state. A warp that shows the object 2 cm below its true position at every control step shows plausible frames, yet no solo episode produces that sequence, and μ closes the gripper above the surface. The whole presented trajectory must be one that μ could have produced on its own. An input warp W_{in} and an output warp W_{out} have this property if warping the next state equals stepping from the warped state with μ's own action, $W_{\text{in}} P_1(s_1, W_{\text{out}} a) = P_1(W_{\text{in}} s_1, a)$, which is equation 3 with $P_1' = P_1$. We say the warp *realizes* $(W_{\text{in}}, W_{\text{out}})$ if it presents $W_{\text{in}}\sigma(s, x^*)$ and executes W_{out} of μ's action. Then, as long as nothing outside the attempt touches x^* or the gripper, μ runs a solo episode from $W_{\text{in}}\sigma(s, x^*)$ in warped coordinates. If μ sees $W_{\text{in}}\sigma(s_t)$ and emits a_t, the world executes $W_{\text{out}} a_t$ and $W_{\text{in}}\sigma(s_{t+1}) = W_{\text{in}} P_1(\sigma(s_t), W_{\text{out}} a_t) = P_1(W_{\text{in}}\sigma(s_t), a_t)$ is the next solo state. The attempt therefore succeeds with μ's solo success probability at $W_{\text{in}}\sigma(s, x^*)$ whenever that pair lies in $\mathcal{D}_1$, for any N.

Proposition 1 (guaranteed warps). (i) *Retargeting.* The goal is a component that P_1 neither evolves nor reads, so replacing it by $\tilde{g}$ is such a W_{in} with $W_{\text{out}} = \text{id}$, and an offset on the presented goal channels realizes it. An attempt is then a solo episode of μ toward $\tilde{g}$, and re-deciding $\tilde{g}$ during the attempt concatenates solo segments, because μ has no memory. (ii) *Translation.* P_1 is invariant to translations parallel to the floor, and equal offsets on the presented goal, object and gripper realize one. (iii) *Take over.* At $m = 0$ on every output channel the executed command is the offset, whatever μ outputs. (iv) *Limits.* No other rigid motion of the presented frame is realized by the per channel affine class. A rotation about the vertical needs the action rotated back, which mixes channels. A reflection needs the presented object to move against the carried displacement, which a multiplier measured from the current position cannot do and an offset cannot do without remembering where the attempt began, which the crop does not record. A dilation contradicts the fixed lengths of the robot. Object size, shape and dynamics change P_1 itself, so no map between states of one M_1 covers them. □

Proposition 2 (the absorbable class). Let a deployment differ from M_1 in some respect, and let a warp be held over an attempt in which nothing outside touches x^* or the gripper. The difference is absorbed with a guarantee in each of the following five cases, and in no other. (i) *Invisible changes.* $\pi_N(s)$ depends on s only through $\sigma(s, x^*)$ and $\chi(s, x^*)$, so a change of s that leaves both unchanged leaves the action unchanged, and under the contact assumption of the next paragraph it cannot act on the attempt either. (ii) *Symmetry and recalibration.* A constant warp $z \leftarrow mz + o$ on every channel is a recalibration of each sensor and actuator channel of μ on its own. It realizes $(W_{\text{in}}, W_{\text{out}})$ exactly when that recalibration is a symmetry of P_1. On position channels a symmetry of rigid motion under gravity is an isometry, a per-channel isometry is a translation or an axis reflection, and Proposition 1(iv) excludes the reflections, so the translations of Proposition 1(ii) are the whole symmetric part of the class. A deployment whose sensors and actuators differ from those of M_1 by a per-channel recalibration is absorbed by the constant warp that inverts it, since equation 3 then holds with W_{in} and W_{out} the inverse recalibrations. (iii) *Retargeting.* Proposition 1(i). Because μ has no memory, a goal re-decided at some control step starts a fresh solo episode from the presented state at that step, so a sequence of retargets is a sequence of solo segments, each carrying μ's solo success at its start. The retargets reachable at a state are the points within the offset ladder of the line from x^* to its goal, re-chosen at every control step, which lets the warp produce detours, hovering, retreating and holding. (iv) *Tolerance.* Suppose μ's solo success is Lipschitz in constant input and output biases, $p_1(s_1; b_{\text{in}}, b_{\text{out}}) \geq p_1(s_1) - L_{\text{in}}\|b_{\text{in}}\| - L_{\text{out}}\|b_{\text{out}}\|$ for $\|b\| \leq \delta$, with L_{in}, L_{out} and δ measured by injecting biases into solo episodes. A warp whose non symmetric part is a constant bias inside δ runs the solo loop with that bias injected, so the attempt succeeds with probability at least $p_1(\sigma) - L_{\text{in}}\|o_{\text{in}}\| - L_{\text{out}}\|o_{\text{out}}\|$. (v) *Take over.* At $m = 0$ on a channel the executed value is o whatever μ outputs. With $m = 0$ on every output channel the attempt is an open-loop command sequence re-decided from the crop at every control step. What such a controller can do is known

without μ, and what it cannot do is anything that needs feedback on a quantity the crop omits. *No other case.* For every channel the family $mz + o$ is a line with two special points, $(m, o) = (1, 0)$, where the channel is μ's, and $m = 0$, where it is the warping policy's. At any other point the presented value co-varies with the true one at a gain $m \notin \{0, 1\}$ or carries a bias, which by (ii) realizes no symmetry unless it is a translation and by (iv) carries a bound only inside δ. Such a presentation is one no solo episode produces, and nothing about its success follows from μ's solo competence. □

Why the count enters only through the crop. We assume that objects act on each other only through contact, that a resting object moves only when touched, and that at selection no other object touches x^* or the gripper. An outside object can then only reach the attempt by being pushed into it, and nothing pushes it, so the outcome of an attempt is a function of the crop and of the warp sequence, which ψ computes from the crop. Since objects have a minimum size, a crop of fixed radius holds at most a fixed number $K_{\max}$ of objects, so the attempt success p_e is one function on a crop set that does not depend on N. The learned maps receive inputs from that set only and never N itself. The other objects enter the crop through the per-pair scores, and objects beyond the radius are masked, so an object added beyond it leaves the crop, the warp and the selection unchanged, which a mean or a softmax over every object would not, since their value at a fixed local geometry moves with N. The count enters only through which crops deployment at N produces, and δ_{cov} measures the mass of those crops outside the region training covered. Where the assumption fails, a neighbor that topples for instance, the failure is not hidden and appears in the measured λ and p_e (Table 9).

Theorem 1 (completion for every N). Let p_e lower bound the success probability of an attempt whose crop is covered, let λ upper bound the expected number of placed objects that one attempt unplaces, both conditionally on the history at every state with unplaced objects, and let no attempt unplace more than $\lambda_{\max}$ objects. Let $\rho = (1 - \delta_{\text{cov}})p_e - \lambda > 0$. Then for every $c > 1/\rho$, the probability that the task is not complete within cN attempts is at most $\exp(-2(\rho c - 1)^2 N/(c(1 + \lambda_{\max})^2))$, which is equation 4.

Proof. Two clocks. The environment advances in control steps and the proof advances in attempts, so we fix both. Let $s_0, s_1, s_2, \ldots$ be the scene states of one episode of M_N at successive control steps, and let $0 = \tau_0 < \tau_1 < \tau_2 < \ldots$ be the control steps at which an attempt ends, because its object is placed or the selection policy picks a different object. Attempt t is then the span of control steps from τ_{t-1} to τ_t (Section 3.3), and s_{τ_t} is the scene at the instant it ends. Everything below is indexed by the attempt counter $t = 0, 1, 2, \ldots$, and control steps enter only through the states s_{τ_t}.

The counter Φ_t. Call an object *placed* in a scene state when it rests within tolerance of its goal pose on its intended support, let $\Phi(s) \in \{0, \ldots, N\}$ be the number of placed objects in the state s, and set $\Phi_t = \Phi(s_{\tau_t})$. So Φ_t is a property of one scene, read at the instant attempt t ends, and not a tally of past successes. It rises by one when an attempt places its object and falls when an attempt knocks placed objects over, and the task is finished exactly when $\Phi_t = N$.

The history $\mathcal{F}_t$. Let $\mathcal{F}_t$ be the episode's trajectory up to that same instant, namely $s_0, \ldots, s_{\tau_t}$ together with the selections and warps that produced it, read as a σ-algebra so that conditional expectations given it are defined. A trajectory only grows, $\mathcal{F}_t \subseteq \mathcal{F}_{t+1}$, so the $\mathcal{F}_t$ form a filtration, and Φ_t is measurable with respect to $\mathcal{F}_t$, since it is read off the last state of the trajectory. We state the constants as bounds conditional on $\mathcal{F}_t$. They must therefore hold after every history that can occur in an episode, and an average over whole episodes is not enough. Otherwise, frequent successes early in an episode could hide later attempts whose expected gain is below ρ.

Finally let $T_N = \min\{t : \Phi_t = N\}$ be the completion time. The task is complete within cN attempts exactly when $T_N \le cN$, so the quantity to bound is $\Pr[T_N > cN]$.

Step 1, one attempt gains ρ objects in expectation. Fix any t with $\Phi_t < N$. Attempt $t + 1$ places a new object if two things hold. The crop of the selected object lies in the region training covered, which fails with probability at most δ_{cov}, and the attempt then succeeds, which has probability at least p_e. We count every uncovered crop as a failure, although some of them succeed. Multiplying the two lower bounds the probability of a gain of one by $(1 - \delta_{\text{cov}})p_e$. The same attempt unplaces at most λ already placed objects in expectation and never more than $\lambda_{\max}$ of them, so

$$\mathbb{E}[\Phi_{t+1} - \Phi_t \mid \mathcal{F}_t] \;\ge\; (1 - \delta_{\text{cov}})p_e - \lambda \;=\; \rho \;>\; 0, \qquad \Phi_{t+1} - \Phi_t \in [-\lambda_{\max}, 1].$$

The number of placed objects is a random walk with bounded steps and an expected gain of at least ρ per attempt, so filling N places takes about N/ρ attempts, and choosing $c > 1/\rho$ gives a budget of cN attempts with a margin above N/ρ.

Step 2, remove the expected gain and concentrate. Write $t \wedge T_N = \min(t, T_N)$, which freezes the walk once the task is complete, and set $M_t = \Phi_{t \wedge T_N} - \rho(t \wedge T_N)$. Subtracting the guaranteed gain makes M_t a submartingale, $\mathbb{E}[M_{t+1} \mid \mathcal{F}_t] \geq M_t$, because before T_N its increment is $\Phi_{t+1} - \Phi_t - \rho$, which is non negative in expectation by Step 1, and from T_N on it is exactly zero. Its increments lie in $[-\lambda_{\max} - \rho, 1 - \rho]$, an interval of length $1 + \lambda_{\max}$. Azuma's inequality for a submartingale with increments in an interval of length ℓ states that it is unlikely to end far below where it started, namely $\Pr[M_{cN} - M_0 \leq -a] \leq \exp(-2a^2/(cN\ell^2))$ for every $a > 0$.

Step 3, failure is such a fall. Suppose the task is not complete within cN attempts, so $T_N > cN$ and the walk was never frozen. Then $\Phi_{cN} < N$ while $\Phi_0 \geq 0$, hence $M_{cN} - M_0 = \Phi_{cN} - \Phi_0 - \rho cN < N - \rho cN = -(\rho c - 1)N$, a shortfall that is strictly positive because $c > 1/\rho$. Applying Step 2 with $a = (\rho c - 1)N$ and $\ell = 1 + \lambda_{\max}$ bounds the probability of that event by $\exp(-2(\rho c - 1)^2 N/(c(1 + \lambda_{\max})^2))$, which is equation 4. □

Relation to the ECRL bound. Haramati et al. (2024) bound the error of a scene value that attends over every entity, and the error grows linearly in N because every entity enters the approximated quantity. WAVE approximates no scene value. What has to generalize is p_e, a function on the crop set, so an estimate that is ϵ-close on covered crops is ϵ-close on every scene whose crops are covered, for any N. In equation 4 the count enters through the number of attempts, which tightens the bound with N, and through δ_{cov}, which we therefore measure per count. That locality of the learned quantity gives size-independent guarantees is an older principle (Ruiz et al., 2020; Qu et al., 2022; Yehudai et al., 2021). In WAVE the local neighborhood is the crop, and equation 4 applies the principle to task completion with constants measured on the task.

I Positioning of interface-editing methods

Table 10: Interface-editing methods by where they edit a frozen policy, in what form, whether the edit is a function of the current state, and which shift it is meant to absorb. Only the last column separates this work from all of them, and Proposition 2 states which shifts the edit absorbs with a guarantee.

Method	*Edit site*	*Edit form*	*Re-decided per step?*	*Shift the edit absorbs*
Residual RL (Silver et al., 2018; Johannink et al., 2019; Yuan et al., 2025)	action	additive, bounded	yes	same task, refined
Warp RL (Hirschowitz & Ramos, 2026)	action distribution	spline flow	yes	dynamics
CLAE (Das et al., 2026)	internal activations	affine	yes	behavior mode
ZPRL (Yu et al., 2026)	inserted latent	latent perturbation	yes	same task, refined
DC-Ada (Alqithami, 2026)	observation	transform found by reward-only random search	no	sensor heterogeneity
APB (Park & Lee, 2026)	before + after a backbone	dense linear	no	task (meta-transfer)
Pivoting (Zhang et al., 2023)	observation + action	diagonal scaling	no, once per episode	object properties (dynamics)
Inter-task mappings (Taylor et al., 2007)	state + action	hand-specified map	no	task pair
This work	observation + action	per-channel discrete affine	yes	*scene composition (object count)*

Prior work edits the observation of a frozen policy (Alqithami, 2026; Park & Lee, 2026; Zhang et al., 2023), both its input and its output (Zhang et al., 2023; Park & Lee, 2026; Taylor et al., 2007), or its

internal activations with an affine edit chosen by an RL policy (Das et al., 2026). These edits adapt a policy to a new sensor layout, task, object or behavior mode, and most of them are fixed for a whole episode or task (Table 10). None of them changes the number of objects a policy can handle. WAVE edits the input and output of a frozen single-object skill, chooses a new edit at every step from the neighborhood of the selected object, and uses this edit to generalize from a few objects to many. The ablations of Zhang et al. (2023) also show that the transfer depends on the form of the edit. Their pivoting skill transfers in 30 of 40 trials with both diagonal projections, in 27 without the action projection, in 19 without the state projection, and in none when the diagonal structure is replaced by free form neural maps. Proposition 1 gives a possible reason. Only a few edits come with a guarantee, namely retargeting the goal, translating the scene and taking over a channel. Among all free-form neural maps these edits form a set of measure zero, so training a free-form map is unlikely to find one of them.

The classical transfer literature contains related framings that edit no interface. MDP homomorphisms and inter task transfer (Ravindran & Barto, 2003; Taylor & Stone, 2009) relate two tasks through structural correspondences, options and learned initiation sets (Sutton et al., 1999; Graves et al., 2020) decide when a fixed sub policy may run, and prior policies shape the exploration of a new task (Uchendu et al., 2023; Schmitt et al., 2018; Galashov et al., 2019; Tirumala et al., 2022). None of them edits what a skill reads or emits, so Table 10 does not list them.