CONTACT AS INTERFACE: COMPOSING OBJECT-CENTRIC INTERACTIONS WITH EXECUTOR-AWARE FIELDS FOR LONG-HORIZON PUSHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon planar pushing in clutter faces two coupled challenges: collision-free object paths may require contacts that a robot cannot realize, and uncertain interaction dynamics can cause deviations from planned motions. We present Contact as Interface (CaI), a framework for long-horizon pushing that connects object-level planning and embodiment-specific execution through a reusable contact interface. An inverse interaction policy implements this interface by mapping normalized object geometry and local motion intents to contact actions. We learn this local mapping using only isolated pushes generated automatically in simulation, without complete task demonstrations, and keep the resulting policy fixed across downstream tasks. Deployment requires no object-specific dynamics identification. Executor-aware cost-to-go fields combine object-space reachability with direction-dependent execution costs to guide both nominal path construction and corrective intent search. An embodiment-specific executor selects and executes feasible contact actions. When progress stalls or repeated attempts fail to find feasible contact actions for the current intent, feedback-driven recovery searches for alternative motion intents. On two benchmarks we construct, CaI achieves 82.29% success in coupled local pose control on held-out geometries and 89.29% overall success in long-horizon pushing, exceeding the success rates of the evaluated baselines. Further experiments demonstrate robustness to nonuniform mass distributions, cross-embodiment reuse, and direct sim-to-real transfer without policy fine-tuning. These results show that a fixed local interaction policy can support long-horizon pushing in clutter when object-level planning accounts for robot feasibility and motion intents are adapted in response to observed deviations in object motion.

1 INTRODUCTION

Pushing enables robots to reposition objects that are difficult to grasp, but transport through clutter couples object motion with contact selection and robot execution (Ren et al., 2025; Raicevic et al., 2026). A collision-free object path may require inaccessible contacts, while uncertain interaction dynamics can disrupt subsequent actions (Li et al., 2018). Knowledge of local contact interactions can be reused across transport tasks, even as goals, obstacle layouts, and robot constraints change. The central challenge is therefore to compose these reusable interactions into long-horizon behavior that accounts for robot feasibility and adapts to execution deviations.

Existing approaches connect task objectives to execution through point-motion guidance (Wen et al., 2024; Bharadhwaj et al., 2024; Zheng et al., 2026), contact actions (Zhou et al., 2023; Jiang et al., 2024), or model-based planning (Bui et al., 2026; Shen et al., 2025). Xie et al. (2026) learn high-level contact intentions comprising surface contacts and object subgoals, which low-level MPC realizes. Closest to our setting, Receding-Horizon Pushing (RHP) (Yuan et al., 2026) composes an object-centric contact policy with future robot feasibility checks and failure-driven path-cost updates. Our design organizes directional execution costs into global cost-to-go fields for route construction and corrective-intent search. This representation provides goal-directed guidance away from the nominal path, where finding a feasible contact alone does not determine how to continue the task.

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.

We introduce *Contact as Interface* (CaI), a hierarchical framework that separates task-level intent selection, local interaction prediction, and robot motion execution. Conditioned on normalized object geometry and a local intent specifying translation and rotation, its inverse policy proposes object-relative surface contacts paired with pushing directions. Isolated simulated pushes provide inverse supervision by pairing each observed motion with its generating action, without task-level demonstrations. The planner and feedback-driven recovery mechanism supply or revise intents, while an embodiment-specific executor validates and executes contact actions. This decoupling allows task objectives and robot constraints to shape how local interactions are composed while the learned policy remains fixed across downstream tasks and embodiments.

Execution costs depend on direction: different motions from the same object position may require different contact sides and robot postures. CaI constructs these costs from approximate feasibility assessments and reconfiguration effort, biasing route selection toward accessible contacts. During execution, the current tracking intent may become ineffective, or none of its contact proposals may pass the execution checks. Recovery therefore revises the intended object motion. It searches alternative intents using the inverse policy to propose contact actions and robot feasibility checks to assess their execution, with cost-to-go guidance relating these alternatives to the task goal. The fields guide which object motions to attempt, while the executor validates concrete robot motions online. In summary, CaI makes three contributions:

- We formulate a contact interface that separates reusable local interaction knowledge from task-level decisions and robot-specific execution. We learn a local policy through inverse supervision on isolated simulated pushes and keep it fixed during composition across tasks and embodiments.
- We incorporate directional execution costs into object-level planning to favor routes with robot-realizable contacts. The resulting executor-aware cost-to-go fields provide shared guidance for nominal route construction and corrective-intent search.
- Feedback-driven recovery revises motion intents when nominal path tracking becomes ineffective. Guided by the global cost-to-go field, it searches alternative intents using policy-generated contact proposals and robot feasibility checks, seeking an executable continuation.

2 Related Work

Motion and contact interfaces for manipulation. Manipulation policies use future images (Du et al., 2023), point trajectories (Wen et al., 2024; Bharadhwaj et al., 2024), and point flows (Zheng et al., 2026) to connect task guidance with robot actions. For articulated objects, Where2Act (Mo et al., 2021) predicts action affordances, while VAT-Mart (Wu et al., 2022) also proposes task-aware action trajectories. HACMan (Zhou et al., 2023) and HACMan++ (Jiang et al., 2024) learn object-centric contact actions and spatially grounded primitives. Whereas Xie et al. (2026) learn to select contacts and object subgoals for contact-implicit MPC, CaI predicts contact actions from externally supplied local motion intents. Our policy learns from isolated pushes, with intent selection and robot-specific execution handled separately.

Long-horizon non-prehensile manipulation. Push-Net (Li et al., 2018) adapts planar pushing from interaction history. Push Anything (Bui et al., 2026) uses contact-implicit MPC, while BaB-ND (Shen et al., 2025) searches action sequences under learned forward dynamics. Object-centric planning (Ren et al., 2025) guides robot execution with object-level plans, while CLOI (Raicevic et al., 2026) coordinates object and robot plans through object-pose consensus. RHP (Yuan et al., 2026) also learns a local intent-conditioned contact policy from isolated pushes. It validates future pushes along BIT* (Gammell et al., 2015) paths through robot motion planning and updates soft spatial and directional edge costs after failed checks to guide replanning. Its horizon analysis reports that longer lookahead can accumulate prediction errors and reject physically feasible paths. Failed replanning can then leave execution without a corrective motion reference. CaI builds global cost-to-go fields from soft directional execution costs based on approximate feasibility evidence. These fields guide nominal route construction and corrective-intent search, allowing recovery to seek executable local interactions without first obtaining a complete replacement path.

Feasibility-aware planning. Task and motion planning incorporates feasibility through incremental constraint solving (Dantam et al., 2016), search heuristics incorporating geometry in FFRob (Garrett et al., 2018), and reachability-guided optimization in R-LGP (Ly et al., 2024). Xu et al. (2022) filter

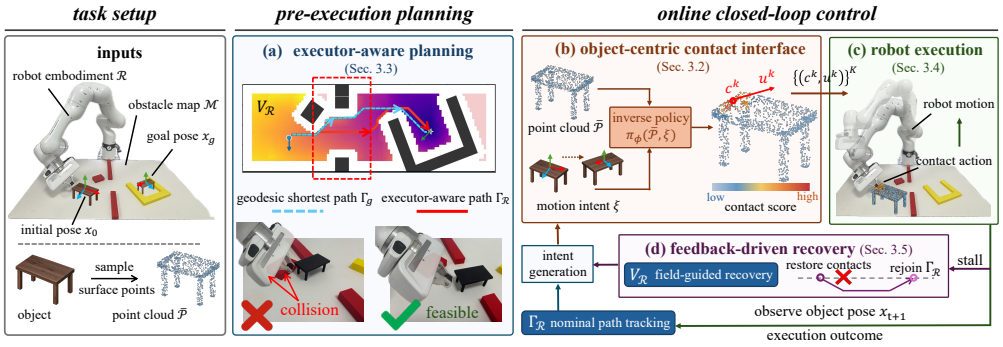

Figure 1: Overview of CaI. (a) Pre-execution planning produces an executor-aware path $\Gamma_{\mathcal{R}}$ and field $V_{\mathcal{R}}$. (b) The inverse policy maps object geometry and local motion intents to contact-action proposals. (c) The executor selects and realizes a feasible proposal. (d) State and execution feedback guide path following and field-guided recovery when progress stalls or contacts become infeasible.

individual candidate actions using learned feasibility checks, while PIGINet (Yang et al., 2023) ranks complete task plans by predicted motion feasibility before trajectory refinement. CaI encodes feasibility evidence and reconfiguration effort as soft directional object-space costs. Since finite probe failures do not establish infeasibility, these costs guide planning without excluding geometrically valid motions that may be feasible with alternative contacts.

3 Method

As shown in Fig. 1, CaI composes local object-centric interactions into long-horizon pushing. We introduce the task and inverse policy in Secs. 3.1 and 3.2, followed by executor-aware planning and embodiment-specific execution in Secs. 3.3 and 3.4. Sec. 3.5 describes closed-loop composition and feedback-driven recovery.

3.1 Problem Formulation

We consider planar non-prehensile manipulation in clutter. A task comprises an obstacle map $\mathcal{M}$, an object point cloud $\mathcal{P} \in \mathbb{R}^{N\times 3}$, initial and goal object poses $x_0, x_g \in \mathrm{SE}(2)$, and a robot embodiment $\mathcal{R}$. The observed object pose at step t is $x_t = (p_t, \theta_t)$, with $p_t \in \mathbb{R}^2$. We assume perceptual estimates of geometry and pose, but no prior knowledge of object-specific mass distribution, friction, or contact dynamics. The objective is to reach a goal region $\mathcal{G}(x_g)$ while avoiding object–obstacle collisions and executing contact actions through kinematically feasible robot motions without unintended collisions. The goal region imposes a position tolerance and, for asymmetric objects, an additional orientation tolerance.

CaI links object-level planning and embodiment-specific execution through a learned object-centric contact interface. The planner uses executor feasibility to guide local motion intents, which the policy maps to contact proposals for executor validation and execution. Observed object poses guide subsequent intents and recovery under unknown interaction dynamics.

3.2 Inverse Interaction Policy

A desired object motion does not directly specify where or how to push, as the resulting translation and rotation depend on the contact location and object geometry. We therefore learn a task-agnostic inverse interaction policy that maps local motion intents and object geometry to surface contact points and pushing directions, without explicitly modeling forward contact dynamics. Geometry, motion, and contact actions are expressed in an object-centric local frame, with frame superscripts omitted for clarity. Let $\mathcal{P} = \{p_i\}_{i=1}^{N}$ denote the observed surface point cloud expressed in this frame, and let $\Delta p \in \mathbb{R}^2$ and $\Delta\theta$ denote the desired local translation and rotation. We normalize the

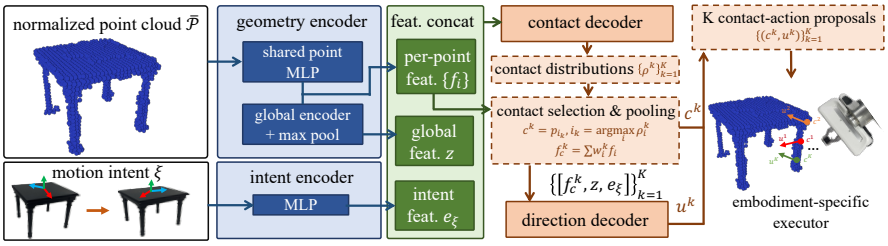

Figure 2: Inverse interaction policy. Object geometry and motion intent guide surface contact selection and contact-conditioned direction prediction, yielding K paired contact-action proposals.

geometry and motion intent as

$$D = \max_i \|p_i\|_2, \qquad \bar{\mathcal{P}} = \mathcal{P}/D, \qquad \xi = \text{normalize}\left(\Delta p/D, \Delta\theta\right). \tag{1}$$

Here, D is the maximum point distance from the local origin, $\bar{p}_i = p_i/D$, and $\text{normalize}(v) = v/\|v\|_2$. Scale normalization reduces dependence on object size, while unit normalization preserves the relative translation–rotation proportions and removes overall motion magnitude. The policy, with learnable parameters ϕ, selects K point indices and their associated pushing directions,

$$\pi_\phi(\bar{\mathcal{P}}, \xi) = \left\{(i_k, u^k)\right\}_{k=1}^K, \tag{2}$$

where $i_k \in \{1, \ldots, N\}$ and $u^k \in \mathbb{R}^2$ is a unit horizontal pushing direction. Retrieving $c^k = p_{i_k} = D\bar{p}_{i_k}$ from the original-scale point cloud yields the contact-action proposal (c^k, u^k). Multiple proposals accommodate alternative contacts for the same intent and allow the executor to select an executable candidate. We use $N = 1024$ and $K = 8$. Surface-point selection determines contact height, while the executor sets the push distance.

Local interaction data. We automatically sample and execute local pushes in simulation, recording geometry $\mathcal{P}$, contact action (c, u_{gt}), and the resulting object motion. Here, u_{gt} is the recorded pushing direction expressed as a unit planar vector in the local frame. The observed motion is normalized into ξ to condition prediction of the recorded action. This provides inverse supervision without requiring each push to achieve a predefined displacement. A local surface patch, represented by the point-index set $\mathcal{C}_{\text{gt}} \subseteq \{1, \ldots, N\}$, defines a soft contact-label distribution $y = \{y_i\}_{i=1}^N$, where y_i is the target probability assigned to point i. Jointly translating and rotating the local frame augments geometry, motion, and action consistently. No teleoperation, complete task demonstrations, or identification of object-specific dynamics is required.

Algorithm 1 Two-stage policy training

Input: Training/validation data, schedule r.
Output: Policy π_ϕ.

Initialize ϕ.
for each Stage-I batch **do**
  Encode inputs and predict ρ^k.
  $w_i^k \leftarrow r y_i + (1 - r)\rho_i^k$.
  Predict u^k using Eq. (3).
  Optimize ϕ with $\mathcal{L}_{\text{I}}$, Eq. (10).
end for
Restore the selected Stage-I checkpoint.
Freeze all but $\mathcal{D}_{\text{dir}}$.
for each Stage-II batch **do**
  Predict ρ^k, then $i_k \leftarrow \arg\max_i \rho_i^k$.
  $w_i^k \leftarrow \mathbb{I}[i = i_k]$. Predict u^k via Eq. (3).
  Keep pairs with $i_k \in \mathcal{C}_{\text{gt}}$.
  if no pair remains, **continue**.
  Optimize $\mathcal{D}_{\text{dir}}$ with $\mathcal{L}_{\text{II}}$, Eq. (11).
end for
return the selected Stage-II policy.

Policy architecture. As shown in Fig. 2, our lightweight PointNet-style policy (Qi et al., 2017) uses a geometry encoder to extract per-point features f_i and a global shape feature z, and an intent encoder to map ξ to e_ξ. The contact decoder concatenates f_i with the broadcast features z and e_ξ to predict K contact distributions ρ^k over the input points. Each distribution assigns probability ρ_i^k to point i and selects $i_k = \arg\max_i \rho_i^k$, yielding the original-scale contact coordinate $c^k = p_{i_k}$. A shared direction decoder $\mathcal{D}_{\text{dir}}$ then conditions each pushing direction on its associated contact feature:

$$f_c^k = \sum_i w_i^k f_i, \qquad u^k = \text{normalize}\big(\mathcal{D}_{\text{dir}}(f_c^k, z, e_\xi)_k\big). \tag{3}$$

Here, w_i^k is the pooling weight and $(\cdot)_k$ selects the corresponding two-dimensional output branch. At deployment, $w_i^k = \mathbb{I}[i = i_k]$, where $\mathbb{I}$ is the indicator function. Point selection anchors contacts to the observed surface, while contact-conditioned direction prediction promotes contact–direction consistency. The network contains approximately 0.45M parameters and receives no object identity, scene map, robot state, or physical parameters.

Two-stage training. Alg. 1 summarizes policy learning. Stage I jointly trains both encoders and decoders with soft contact cross-entropy and directional cosine loss. A relaxed winner-take-all objective (Rupprecht et al., 2017) favors the best-matching proposal while weakly supervising the others, encouraging alternative proposals without averaging incompatible actions. As the teacher-forcing ratio r decreases, contact pooling shifts from soft labels y to predicted distributions. Stage II calibrates the direction decoder for single-point conditioning at deployment. It starts from the validation-selected Stage-I checkpoint and freezes both encoders and the contact decoder. Calibration uses only proposals with $i_k \in \mathcal{C}_{\text{gt}}$, whose selected points lie in the recorded contact region, and skips batches without matches. App. B.1 details data construction, network specifications, and training objectives.

3.3 Executor-Aware Planning

A collision-free object path may require contacts the robot cannot reach without collision. Checking feasibility only during execution can leave the object in states where the robot cannot access the required contacts. CaI thus adds directional executor costs to global planning to favor routes with accessible contacts. Since finite probes provide approximate feasibility evidence, failures incur soft penalties rather than exclude geometrically reachable states. Fig. 3 compares (a) the object-only field and path, (b) the fraction of failed directional probes, and (c) the executor-aware field and path. Panel (b) aggregates directions for visualization, while planning retains direction-specific costs.

Object-space reachability. We first compute a geometric reference path from $\mathcal{M}$ using an object-footprint approximation, retaining a yaw schedule for asymmetric objects. A footprint envelope defines a grid of collision-free object-center positions $\mathcal{S}_{\text{free}}$, on which reverse shortest-path search computes the geodesic distance field V_{G} to the goal cell s_g. This field captures geometric connectivity under the footprint approximation, independently of robot constraints. Executor costs are subsequently evaluated on the same positional grid, while the reference yaw schedule is transferred to the final path rather than jointly optimized with executor feasibility.

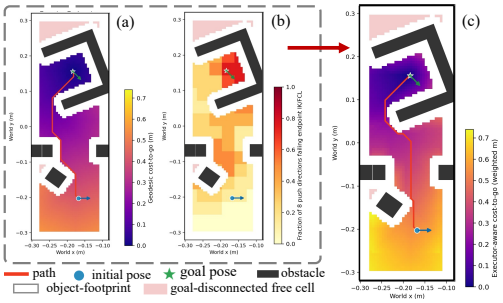

Figure 3: Executor-aware planning. (a) Object-space geodesic field V_{G} and shortest path. (b) Probe failure ratio across eight pushing directions. (c) Executor-aware cost-to-go field $V_{\mathcal{R}}$ and nominal path $\Gamma_{\mathcal{R}}$.

Executor-aware transition costs. Let $s \in \mathcal{S}_{\text{free}}$ denote an object-center grid state and $\mathcal{D}$ the set of eight unit planar directions. For each $d \in \mathcal{D}$, let s_d denote the neighbor of s along d. Admissible directions are given by $\mathcal{D}(s) = \{d \in \mathcal{D} \mid s_d \in \mathcal{S}_{\text{free}}\}$. To avoid exhaustive robot motion planning over the grid, we use coarse state–direction probes to evaluate representative analytic contacts through inverse kinematics and full-arm endpoint collision checks. Probe feasibility and joint-reconfiguration effort determine a directional multiplier $m_{\mathcal{R}}(s, d) \geq 1$, yielding the transition cost

$$\kappa_{\mathcal{R}}(s, d) = \ell(s, d) m_{\mathcal{R}}(s, d), \tag{4}$$

where $\ell(s, d)$ is the geometric edge length.

Coarse probes may overlook geometry-specific contact constraints in narrow passages. We identify narrow passages from the obstacle layout in the map along a preliminary route computed from the probe costs. At sampled traversal states, we evaluate policy proposals using object swept-volume, inverse-kinematics, and robot collision checks. This evidence refines the directional multipliers before the final solve, favoring locally feasible contacts without guaranteeing that the entire contact sequence can be executed.

Cost-to-go fields and nominal path. Reverse Dijkstra search (Dijkstra, 1959) on the directed grid graph with the refined transition costs computes the scalar cost-to-go field,

$$V_{\mathcal{R}}(s) = \min_{d \in \mathcal{D}(s)} \left[\kappa_{\mathcal{R}}(s, d) + V_{\mathcal{R}}(s_d) \right], \qquad V_{\mathcal{R}}(s_g) = 0. \tag{5}$$

To reduce contact switching, an additional solve over augmented states (s, h) penalizes frequent and abrupt turns. Here, $h \in \mathcal{D}$ denotes the incoming translational direction, not object yaw. The resulting $J_{\mathcal{R}}(s, h)$ includes turning penalties, while $V_{\mathcal{R}}$ is independent of h. Tracing Bellman-optimal successors (Bellman, 2010) using $J_{\mathcal{R}}(s, h)$ and transferring the reference yaw schedule yields the executor-aware nominal path $\Gamma_{\mathcal{R}}$. $\Gamma_{\mathcal{R}}$ guides nominal motion intents, while $V_{\mathcal{R}}$ supports progress assessment and feedback-driven recovery (Sec. 3.5). Probe construction, cost refinement, and path optimization are detailed in App. B.2.

3.4 Embodiment-Specific Execution

Since planning provides only approximate feasibility evidence, the executor validates concrete motions from the current robot configuration. It transforms each proposal (c^k, u^k) into the world frame, preserves the surface contact point and height, and supplies an end-effector orientation, push distance, and trajectory for $\mathcal{R}$.

Feasibility-aware contact selection. The executor filters and prioritizes policy proposals by compatibility with the current motion objective, then searches for feasible motion plans in that order within a planning budget. It searches embodiment-specific end-effector orientations and plans staging, pre-contact, and pushing motions subject to inverse-kinematics feasibility, object swept-volume clearance, and full-body robot–obstacle collision checks. Orientations with the intended end-effector contact surface facing the object are preferred, while side or reverse orientations remain available when required by kinematic or clearance constraints. Selection operates on policy-predicted contact–direction pairs without altering their contact points. If no feasible motion plan is found, the executor reports failure to the closed-loop recovery mechanism.

Motion execution and monitoring. The executor follows the planned approach, establishes contact, and executes a bounded Cartesian push. Compatible consecutive intents can reuse the established contact when continuation checks pass, reducing unnecessary disengagement and re-approach. The nominal push horizon adapts to the remaining route cost and local workspace, clearance, terminal-pose, and recovery constraints. Online collision checks and monitoring of contact tracking, object tilt, and force/torque signals can interrupt unsafe or ineffective motion. Observed object poses and execution outcomes guide subsequent decisions while the policy remains fixed.

3.5 Closed-Loop Composition and Recovery

Closed-loop composition. At each step, the planner constructs a local motion intent from the observed object pose and $\Gamma_{\mathcal{R}}$. Translation follows a nearby downstream reference, while asymmetric objects also track path yaw, blending toward the goal orientation near the target. The policy and executor realize the intent through a bounded interaction, reusing contact when continuation remains feasible. New observations guide subsequent decisions without a forward interaction model.

Feedback-driven recovery. Unknown dynamics can make nominal path tracking ineffective or render the required contacts inaccessible. We therefore combine observed progress with online checks of policy-proposed contact actions to assess whether nominal guidance remains useful. Progress is measured by the decrease in $V_{\mathcal{R}}$, or in goal-yaw error during terminal rotation. Persistent lack of progress or repeated feasibility-check failures prompt corrective intents rather than continued tracking of the same reference. Recovery uses local descent of $V_{\mathcal{R}}$ as a reference and searches alternative intents through policy queries and robot feasibility checks. Bounded temporary increases in route cost permit local adjustments when direct descent is not executable. Each interaction is evaluated from new observations, and evidence of progress, viable path reconnection, and feasible contacts supports a return to nominal path tracking. App. B.3 provides the execution and recovery implementation details.

4 EXPERIMENTS

Using the setup in Sec. 4.1, we evaluate local interaction control in Sec. 4.2, long-horizon pushing and robustness to mass redistribution in Sec. 4.3, and component ablations in Sec. 4.4. Sec. 4.5 examines cross-embodiment and sim-to-real transfer without fine-tuning. Code and additional videos are provided in the supplementary material.

4.1 EXPERIMENTAL SETUP

Training data and implementation. We train on 1,520 simulated local pushes from 36 object geometries and validate on 207 separate pushes. Five local-frame augmentations per push yield 7,600 training and 1,035 validation samples. Both stages use AdamW (Loshchilov & Hutter, 2019) with batch size 64 and initial learning rates of 10^{-3} and 3×10^{-4} for Stages I and II, respectively. All CaI evaluations use the same validation-selected Stage-II checkpoint without fine-tuning. Training details and validation curves appear in App. B.1.

Evaluation benchmarks. We construct two complementary benchmarks to assess local interaction capability and its composition into long-horizon behavior. The *local pose-control benchmark* comprises 288 coupled translation–rotation tasks across three unseen asymmetric geometries, without obstacles or robot kinematic constraints. The *long-horizon pushing benchmark* pairs 14 geometrically diverse objects with 60 cluttered maps, yielding 840 tasks (Fig. 4). The maps are equally divided into easy, medium, and hard tiers with increasing geometric constraints. Seen and unseen denote object geometries included in and held out from policy training, respectively. Neither benchmark maps nor complete task trajectories are used for training. Benchmark construction and baseline implementations are detailed in Apps. C.1 and C.4. Evaluation protocols and success criteria appear in Secs. 4.2 and 4.3.

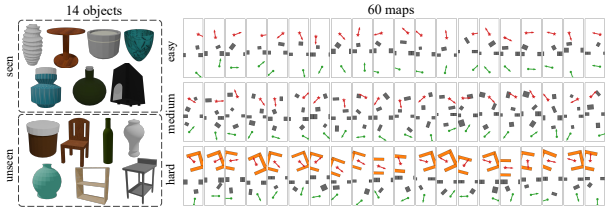

Figure 4: Benchmark objects and maps, with initial/goal poses in green/red and obstacles in gray/orange.

Experimental platforms. All simulations use SAPIEN (Xiang et al., 2020), with robot-embodied environments implemented in ManiSkill (Tao et al., 2025). Local pose control uses an ideal kinematic spherical pusher of radius 0.01 m in a custom SAPIEN scene. Long-horizon pushing uses a 7-DoF Franka Emika Panda with a parallel-jaw gripper held closed during manipulation. Transfer experiments in Sec. 4.5 use a simulated Fetch mobile manipulator and a Franka Research 3. All experiments use Intel Xeon Gold 6530 CPUs and a single NVIDIA RTX 4090 GPU per run.

4.2 LOCAL INTERACTION CONTROL

Evaluation protocol. For local control, learned policies use residual object pose errors as motion intents, without global planning or task-level recovery. Goals for each unseen asymmetric object in Fig. 4 combine translations of $1.5R$ and $2.5R$ along eight equally spaced bearings with yaw changes of $\pm30°$, $\pm60°$, and $\pm90°$, where R is the footprint radius. Success requires position error below $0.25R$ and yaw error below $10°$ without toppling. We compare against Push Anything's mesh-based Sampling C3+ controller (Bui et al., 2026) and RHP-Policy (Yuan et al., 2026), trained on the same local-push data and split as CaI. RHP-Policy variants share a checkpoint: (S) uses CaI's *shared*, radius-scaled, residual-adaptive push-length rule, while (P) uses the policy's *predicted* length. All methods share task instances, physical parameters, pusher geometry, and success criteria.

Coupled pose-control performance. Tab. 1 groups the same coupled trials by translation distance or yaw magnitude. Overall, CaI achieves 82.29% success, compared with 79.17% for RHP-Policy (S), 67.01% for RHP-Policy (P), and 54.17% for Push Anything. The gap between the RHP variants highlights the role of push-length assignment. Under the same rule, CaI outperforms RHP-Policy (S) across all groups, with the largest margin at $90°$ yaw (86.46% versus 80.21%). Both remain comparatively stable as translation distance increases, whereas RHP-Policy (P) and Push Anything

Table 1: Coupled pose-control success (%).

	Translation		Yaw magnitude			Overall
Method	$1.5R$	$2.5R$	30°	60°	90°	All
Push Anything	61.81	46.53	58.33	53.13	51.04	54.17
RHP-Policy (S)	79.86	78.47	77.08	80.21	80.21	79.17
RHP-Policy (P)	74.31	59.72	66.67	72.92	61.46	67.01
CaI	**83.33**	**81.25**	**78.13**	**82.29**	**86.46**	**82.29**

Table 2: Long-horizon pushing performance.

Method	Easy	Medium	Hard	Score
	SR (%) / Avg. completion time (s)			
Seen objects				
RHP ($h = 1$)	33.57 / 221.32	18.57 / 187.73	9.29 / 181.66	11.303
RHP ($h = 3$)	25.00 / 207.92	7.86 / 250.98	6.43 / 350.57	5.975
BIT*–CaI	79.29 / 84.93	55.71 / 132.31	50.00 / 132.16	46.134
CaI	**94.29 / 137.20**	**92.86 / 160.53**	**92.86 / 178.66**	**67.552**
Unseen objects				
RHP ($h = 1$)	40.00 / 210.30	15.00 / 174.90	10.00 / 188.15	11.816
RHP ($h = 3$)	35.00 / 214.14	10.71 / 217.95	10.71 / 282.45	9.384
BIT*–CaI	76.43 / 130.76	53.57 / 157.25	60.00 / 190.82	44.273
CaI	**90.71 / 181.91**	**84.29 / 180.31**	**80.71 / 200.60**	**57.194**

Table 3: Real-world successes (out of 5).

	Object 1	Object 2	Object 3
Map 1	4 / 5	4 / 5	5 / 5
Map 2	5 / 5	5 / 5	5 / 5

Table 4: Robustness to mass redistribution.

Split	Easy	Medium	Hard	Score
	SR (%) / Avg. time (s)			
Seen	85.00 / 156.01	82.14 / 175.64	82.14 / 180.33	58.736
Unseen	86.43 / 178.69	80.71 / 192.04	82.14 / 214.83	54.968
Overall	85.71 / 167.44	81.43 / 183.77	82.14 / 197.58	56.852

Table 5: Planning and recovery ablations.

Variant	Easy	Medium	Hard	Average
	Success Rate (%)			
w/o Executor-Aware Field	87.86	**88.93**	82.14	86.31
w/o Recovery	85.71	79.64	67.86	77.74
CaI (full)	**92.50**	88.57	**86.79**	**89.29**

degrade. These results support the learned contact–direction interface for zero-shot coupled pose control beyond push-length assignment alone.

4.3 Long-Horizon Pushing

Baselines and evaluation protocol. We compare three configurations based on Batch Informed Trees (BIT*) (Gammell et al., 2015). **RHP ($h = 1, 3$)** (Yuan et al., 2026) combines learned contact-point, direction, and push-length prediction with stability-based ranking and failure-guided replanning. It validates up to h virtual pushes through robot motion planning, then executes the first. Its policy uses the same training trajectories and split as CaI. **BIT*–CaI** combines RHP's object planner with our policy and executor, retaining persistent contact and executor-assigned push horizons. It replans after waypoint completion, failure, or stalled progress, without virtual lookahead, executor-aware fields, or field-guided recovery.

All methods share objects, maps, pose observations, physical parameters, and success criteria. We evaluate each object–map pair once with uniform density, seed zero, and a budget of $T_{\max} = 600$ s. Success requires position error below 0.05 m without reaching 75° tilt, plus absolute wrapped yaw error below 15° for asymmetric objects. We report success rate (SR) and mean completion time over successful trials. Completion time combines global planning, intent generation, policy inference, robot motion planning, and simulated control-step execution time. For task set $\mathcal{T}_r$ in tier r of each object split, the efficiency score is

$$E_r = \frac{100}{|\mathcal{T}_r|} \sum_{i \in \mathcal{T}_r} \mathbb{I}_i^{\text{succ}} \max\left(0, 1 - \frac{t_i}{T_{\max}}\right), \tag{6}$$

where t_i is completion time and $\mathbb{I}_i^{\text{succ}}$ indicates success. Failures contribute zero. The aggregate score is $E = 0.2E_{\text{easy}} + 0.3E_{\text{medium}} + 0.5E_{\text{hard}}$.

Long-horizon pushing performance. Tab. 2 shows that CaI achieves the highest success rate across all tiers and both object splits. Extending RHP's validation horizon from one to three pushes does not improve overall success, and failed virtual-action validation remains its dominant termination mode. Despite using the same policy and executor, BIT*–CaI achieves 62.50% overall success versus 89.29% for the complete framework. Most hybrid failures occur when post-interaction states fail the planner's collision check, preventing replanning. With the local policy and executor held fixed, the success gap supports the overall contribution of task-level planning and recovery. Although the hybrid reports shorter completion times on its successful subset, CaI achieves higher aggregate scores. Fig. 6 illustrates long-horizon executions. A comparison of the complete framework with a geometric pushing pipeline on shared symmetric-object tasks is provided in App. C.5.

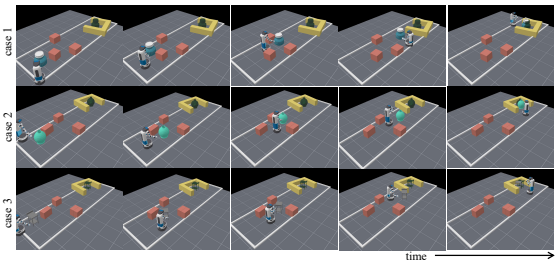

Figure 5: Transfer to a Fetch mobile manipulator.

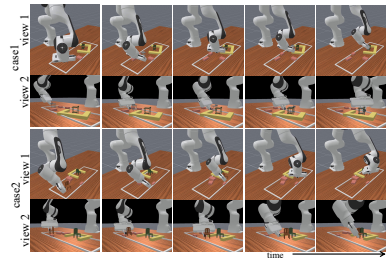

Figure 6: Long-horizon pushing.

Robustness to mass distribution. We repeat all 840 tasks with 25% of each object's mass redistributed to a hidden, diagonally offset point mass, changing its center of mass and inertia but not total mass, collision geometry, or friction. With unchanged geometric inputs and policy weights and without access to inertial parameters, CaI retains 83.10% success versus 89.29% under uniform density (Tab. 4). App. C.2 details the construction.

4.4 Ablation Studies

Component ablations. We evaluate a geometric-field variant without executor-aware guidance and a recovery-disabled variant retaining $V_{\mathcal{R}}$ and $\Gamma_{\mathcal{R}}$. Both preserve the learned policy and online execution checks. Tab. 5 reports success averaged across seen and unseen splits per tier and the unweighted tier mean. Average success falls from 89.29% to 86.31% without the executor-aware field and 77.74% without recovery. The geometric-field variant has lower success rates mainly on easy and hard tasks. Disabling recovery produces the largest gap on hard tasks, at 18.93 percentage points, consistent with the importance of correcting deviations in constrained scenes. Both the recovery-disabled and full variants use $n_{\mathrm{a}} = 4$. Details and audit sensitivity appear in Apps. C and C.3.

4.5 Cross-Embodiment and Sim-to-Real Transfer

Cross-embodiment transfer. We test cross-embodiment transfer on a simulated Fetch mobile manipulator with enlarged objects and the unchanged policy. Executor-aware planning selects object motions and relative base–arm configurations under object and robot swept-volume checks. With its arm retracted, Fetch follows a collision-checked base route accounting for its footprint, obstacles, and current object position before establishing contact at the selected base pose. Fig. 5 shows qualitative examples, with execution details in App. B.3.

Sim-to-real transfer. We deploy the simulation-trained policy on a Franka Research 3 without fine-tuning, estimating object poses with Intel RealSense D455f RGB-D cameras. We test three objects on two obstacle maps, with five initial–goal pose pairs per combination, totaling 30 trials. Fig. 7 shows the setup and executions, and Tab. 3 reports results. These tests support direct transfer of the learned contact interface across the tested objects and layouts under real sensing and interaction uncertainty.

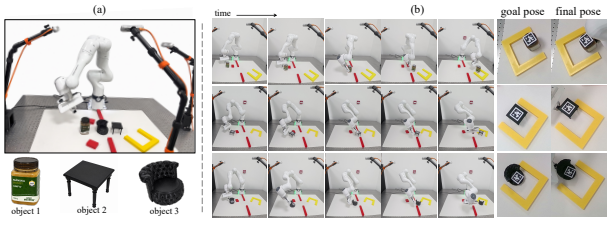

Figure 7: Real-world tests. (a) Setup and objects. (b) Executions and goal–final poses.

5 Conclusion

Contact as Interface (CaI) composes object-centric contacts from a fixed inverse policy trained on isolated simulated pushes into long-horizon pushing. Executor-aware cost-to-go fields combine geometric reachability and directional execution costs to guide nominal paths and feedback-driven recovery. Results show higher overall success than the evaluated baselines, robustness to nonuniform mass distributions, and cross-embodiment and direct sim-to-real transfer without policy fine-tuning.

AI USE STATEMENT

In this work, we used generative AI tools for aiding or polishing writing and generating synthetic datasets. We have not used generative AI tools for retrieval and discovery (e.g., finding related work), research ideation or execution, or drafting sections of the paper, and proving mathematical claims are not applicable to this work. Additionally, we used generative AI tools for reviewing rendered map images and YAML scene specifications for the plausibility of obstacle layouts during the construction of the synthetic long-horizon pushing benchmark. We have reviewed all AI-assisted work. Geometric validity checks were performed programmatically; we verified that all AI-suggested layouts satisfied these geometric constraints before inclusion. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human-subject research or the collection of personal data. The authors have identified no specific ethical concerns associated with this study.

REPRODUCIBILITY STATEMENT

We provide selected policy and training code in the supplementary material. App. B details data collection and augmentation, network architecture, training and checkpoint selection, and the implementation of planning, execution, and recovery. App. C documents benchmark construction, baseline implementations, and evaluation configurations.

REFERENCES

Richard Bellman. *Dynamic Programming*. Princeton University Press, 2010. ISBN 9780691146683.

Homanga Bharadhwaj, Roozbeh Mottaghi, Abhinav Gupta, and Shubham Tulsiani. Track2Act: Predicting Point Tracks from Internet Videos Enables Generalizable Robot Manipulation. In *ECCV*, 2024.

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 *ICRA*, 2026.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. SAM 3: Segment Anything with Concepts. *arXiv preprint arXiv:2511.16719*, 2025.

Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *RSS*, 2016.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1: 269–271, 1959. doi: 10.1007/BF01386390.

Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *NeurIPS*, 2023.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. In *ICRA*, pp. 3067–3074, 2015. doi: 10.1109/ICRA.2015.7139620.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 2018.

Bowen Jiang, Yilin Wu, Wenxuan Zhou, Chris Paxton, and David Held. HACMan++: Spatially-Grounded Motion Primitives for Manipulation. In *RSS*, 2024.

Juekun Li, David Hsu, and Wee Sun Lee. Push-Net: Deep Planar Pushing for Objects with Unknown Physical Properties. In *RSS*, 2018.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Kim Tien Ly, Valeriy Semenov, Mattia Risiglione, Wolfgang Merkt, and Ioannis Havoutis. R-LGP: A Reachability-Guided Logic-Geometric Programming Framework for Optimal Task and Motion Planning on Mobile Manipulators. *arXiv preprint arXiv:2310.02791*, 2024.

Kaichun Mo, Leonidas J. Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2Act: From pixels to actions for articulated 3D objects. In *ICCV*, 2021.

Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *ICRA*, 2012.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017.

Nikola Raicevic, Hyomuk Kim, Shahid Mulla, Hyungjun Doh, Bharath Raam Radhakrishnan, Chenbin Yu, Ki Myung Brian Lee, and Nikolay Atanasov. Closed-loop object-informed control for non-prehensile robot manipulation. *arXiv preprint arXiv:2605.30778v2*, 2026.

Kejia Ren, Gaotian Wang, Andrew S. Morgan, Lydia E. Kavraki, and Kaiyu Hang. Object-centric kinodynamic planning for nonprehensile robot rearrangement manipulation. *arXiv preprint arXiv:2410.00261v4*, 2025.

Christian Rupprecht, Iro Laina, Robert DiPietro, Maximilian Baust, Federico Tombari, Nassir Navab, and Gregory D. Hager. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. In *ICCV*, 2017.

Keyi Shen, Jiangwei Yu, Jose Barreiros, Huan Zhang, and Yunzhu Li. BaB-ND: Long-Horizon Motion Planning with Branch-and-Bound and Neural Dynamics. *arXiv preprint arXiv:2412.09584*, 2025.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-Kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh N, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. ManiSkill3: GPU Parallelized Robot Simulation and Rendering for Generalizable Embodied AI. In *ICLR 2025 Workshops: WRL*, 2025.

Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. VGGT-Ω. In *CVPR*, 2026.

Chuan Wen, Xingyu Lin, John So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point trajectory modeling for policy learning. *arXiv preprint arXiv:2401.00025*, 2024.

Ruihai Wu, Yan Zhao, Kaichun Mo, Zizheng Guo, Yian Wang, Tianhao Wu, Qingnan Fan, Xuelin Chen, Leonidas J. Guibas, and Hao Dong. VAT-Mart: Learning visual action trajectory proposals for manipulating 3D articulated objects. In *ICLR*, 2022.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A SimulAted Part-Based Interactive ENvironment. In *CVPR*, 2020.

Zhixian Xie, Yu Xiang, Michael Posa, and Wanxin Jin. Where to touch, how to contact: A hierarchical RL–MPC framework for geometry-aware sim-to-real manipulation. *arXiv preprint arXiv:2601.10930*, 2026.

Lei Xu, Tianyu Ren, Georgia Chalvatzaki, and Jan Peters. Accelerating integrated task and motion planning with neural feasibility checking. *arXiv preprint arXiv:2203.10568*, 2022.

Zhutian Yang, Caelan Reed Garrett, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Sequence-based plan feasibility prediction for efficient task and motion planning. In *RSS*, 2023.

Zhiyi Yuan, Tianrun Hu, Anxing Xiao, Yuhong Deng, David Hsu, and Hanbo Zhang. Receding-Horizon Pushing with Composable Object-Centric Policies. *arXiv preprint arXiv:2609.23439*, 2026.

Yitian Zheng, Zhangchen Ye, Weijun Dong, Shengjie Wang, Yuyang Liu, Chongjie Zhang, Chuan Wen, and Yang Gao. Translating flow to policy via hindsight online imitation. In *ICLR*, 2026.

Wenxuan Zhou, Bowen Jiang, Fan Yang, Chris Paxton, and David Held. HACMan: Learning Hybrid Actor-Critic Maps for 6D Non-Prehensile Manipulation. In *CoRL*, 2023.

APPENDIX CONTENTS

A Notation 13

B Method Implementation Details 15

- **B.1** Policy Architecture and Training 15
- **B.2** Planning and Feasibility Implementation 18
- **B.3** Execution and Recovery Details 20

C Experimental Details and Additional Results 21

- **C.1** Benchmark Construction 22
- **C.2** Nonuniform Mass Construction 22
- **C.3** Executor-Audit Sensitivity 23
- **C.4** Baseline Implementations 23
- **C.5** Comparison with Geometric Pushing 24
- **C.6** Runtime Analysis 24
- **C.7** Additional Real-World Experiments 25

D Limitations 25

A NOTATION

Tab. 6 summarizes our notation. Superscript w denotes world coordinates, and overloaded symbols are distinguished by context.

Table 6: Notation summary.

Symbol	Meaning
Geometry and motion	
$\mathcal{M},\ \mathcal{R}$	Obstacle map and robot embodiment
$x_t = (p_t, \theta_t)$	Object pose with planar position p_t and yaw θ_t
$x_0,\ x_g,\ \mathcal{G}(x_g)$	Initial pose, goal pose, and goal region
$\mathcal{P},\ p_i$	Surface point cloud and its ith point
$N,\ i$	Input point count and point index
D	Maximum point distance from the local origin
$\bar{\mathcal{P}},\ \bar{p}_i$	Scale-normalized point cloud and point
R	Object footprint radius
$\Delta p,\ \Delta\theta$	Local planar translation and yaw change
ξ	Unit-normalized local motion intent
Contact policy	
$\pi_\phi,\ \phi$	Inverse interaction policy and its parameters
$K,\ k$	Proposal count and index
$i_k,\ c^k$	Selected point index and contact coordinate for proposal k
u^k	Predicted unit planar pushing direction
$c,\ u_{\mathrm{gt}}$	Recorded contact and unit planar pushing direction
$\mathcal{C}_{\mathrm{gt}}$	Recorded contact-patch indices
$y,\ y_i$	Contact-label distribution and pointwise probability
$f_i,\ z,\ e_\xi$	Point feature, shape feature, and intent embedding
$\rho,\ \rho^k,\ \rho_i^k$	Contact-probability matrix, kth contact distribution, and point probability
$w_i^k,\ f_c^k$	Contact-pooling weight and pooled feature
$\mathcal{D}_{\mathrm{dir}}$	Shared direction decoder
Training and validation	
r (training)	Teacher-forcing ratio
$\mathcal{L}_k,\ \lambda_u$	Per-proposal joint loss and direction-loss weight
$k^\star,\ \epsilon$	Best-matching proposal index and total loss weight on other proposals
$\mathcal{L}_{\mathrm{I}},\ \mathcal{L}_{\mathrm{II}}$	Joint-learning and direction-calibration objectives
b	Mini-batch sample index
$i_{bk},\ u_b^k$	Selected point index and direction for sample b, proposal k

Continued on the next page

Table 6 (continued)

Symbol	Meaning
$\mathcal{C}_{\mathrm{gt},b},\ u_{b,\mathrm{gt}}$	Contact-patch indices and target direction for sample b
h_{bk}	Contact-match indicator for sample b, proposal k
k_{val}	Label-selected validation proposal
H_{oracle}	Label-selected contact hit rate
E_{oracle}	Mean label-selected direction error in degrees
S_{val}	Stage-I checkpoint-selection score
Coordinate transforms	
$A,\ Q,\ a$	Local-to-world transform, rotation, and origin
$p_i^w,\ c^w,\ u^w$	World-frame point, recorded contact, and pushing direction
$T_-,\ T_+$	Object-to-world transforms before and after a push
ΔT	Observed rigid motion in the sampled local frame
$R_\Delta,\ t_\Delta$	Rotation and translation of ΔT
$R_t,\ \tau_t$	Policy-frame rotation and origin in world coordinates
$\bar{R}_t$	Three-dimensional extension of planar rotation R_t
$c_t^{k,w},\ u_t^{k,w}$	World-frame contact and direction for proposal k at step t
Executor-aware planning	
$\mathcal{S}_{\mathrm{free}}$	Free object-center grid under the footprint approximation
$s,\ s_t,\ s_g$	Grid state, current state, and goal cell
$\mathcal{D},\ d,\ \mathcal{D}(s)$	Grid directions, a direction, and admissible directions at s
$s_d,\ \ell(s,d)$	Neighbor along d and connecting edge length
$m_{\mathcal{R}}(s,d)$	Directional executor-cost multiplier
$\delta_q(s,d)$	Joint-reconfiguration effort for feasible endpoint probes
$\lambda_q,\ \lambda_f$	Reconfiguration and probe-failure penalty weights
$\kappa_{\mathcal{R}}(s,d)$	Executor-aware transition cost
V_{G}	Object-only geodesic distance to the goal
$V_{\mathcal{R}}(s)$	Executor-aware scalar cost-to-go
h (path planning)	Incoming translational direction
$\lambda_e,\ \lambda_a$	Turn-event and squared normalized-angle penalty weights
$J_{\mathcal{R}}(s,h)$	Executor-aware cost-to-go with turning penalties
$\Gamma_{\mathcal{R}}$	Executor-aware nominal path
Execution and recovery	
x_t^{ref}	Downstream reference pose on the nominal path
$\Delta p_t,\ \Delta\theta_t$	Desired local translation and yaw change at step t
L_t^{nom}	Nominal push horizon
$\alpha,\ L_{\min},\ L_{\max}$	Push-horizon gain, lower bound, and upper bound
ΔV_t	Observed decrease in scalar cost-to-go
n_{a}	Failure-streak threshold for executor-audit warnings
d_t^{rec}	Lowest-cost strict-descent direction for recovery
Evaluation	
h (RHP)	Virtual-push validation horizon
r (evaluation), $\mathcal{T}_r$	Difficulty tier and its task set within an object split
$i,\ t_i,\ T_{\max}$ (evaluation)	Task index, completion time, and time budget
$\mathbb{I}_i^{\mathrm{succ}}$	Task-success indicator
$E_r,\ E$	Tier-specific and aggregate efficiency scores
SR	Success rate
(S), (P)	RHP-Policy with shared or predicted push length
Conventions	
$\mathrm{SE}(2),\ \mathrm{SE}(3)$	Planar and three-dimensional rigid-motion groups
$\mathrm{SO}(2),\ \mathrm{SO}(3)$	Planar and three-dimensional rotation groups
$\mathbb{I}$	Indicator function
$\mathrm{normalize}(v)$	Unit-vector normalization
$\mathrm{clip}(v,a,b)$	Scalar clipping to $[a,b]$
$\ominus$	Relative rigid motion in the policy frame
$\angle(h,d)$	Unsigned angle between grid directions
$(\cdot)_{xy},\ (\cdot)_k$	Horizontal-component extraction and branch selection
$\|\cdot\|_2,\ \langle\cdot,\cdot\rangle$	Euclidean norm and inner product

B METHOD IMPLEMENTATION DETAILS

B.1 POLICY ARCHITECTURE AND TRAINING

We detail data construction, network implementation, and the two-stage training objectives. Alg. 2 summarizes data construction, and Fig. 8 presents the network layers and tensor dimensions. All augmented samples from a source trajectory share one data split, preventing the same interaction from appearing in both training and validation.

Algorithm 2 Local interaction dataset construction

Input: Objects, simulator, per-object quotas.
Output: Samples $(\bar{\mathcal{P}}, \xi, y, \mathcal{C}_{\mathrm{gt}}, u_{\mathrm{gt}})$, grouped by trajectory.

while any object quota is unmet **do**
  Select an under-quota object and sample its configuration and approach pose.
  Execute a push and record geometry, contacts, direction, and poses T_-, T_+.
  if filtering fails, **continue**.
  Apply random local point removal.
  repeat five times ▷ *Data augmentation*
    Sample frame A: origin within the projected convex hull, yaw in $[0, 2\pi)$.
    Sample N points, retaining recorded contacts.
    Transform points, contacts, and direction via Eq. (7).
    $\Delta T \leftarrow A^{-1} T_+ T_-^{-1} A$.
    Extract $(\Delta p, \Delta\theta)$ via Eq. (8).
    Construct $\mathcal{C}_{\mathrm{gt}}$ from five-nearest-point contact neighborhoods.
    $y_i \leftarrow \mathbb{I}[i \in \mathcal{C}_{\mathrm{gt}}]/|\mathcal{C}_{\mathrm{gt}}|$.
    Normalize $(\bar{\mathcal{P}}, \xi)$ via Eq. (1).
    Store the sample with its trajectory ID.
  end repeat
end while
Split data by trajectory.

Interaction collection and filtering. The simulator samples object configurations and gripper approach poses, establishes contact, and executes a short straight push with fixed end-effector orientation. The commanded push length is sampled uniformly from $[0.05, 0.12]$ m. The horizontal direction follows the inward contact normal with optional in-plane perturbation. These collection actions are not policy predictions. Supervision pairs recorded contacts and executed directions with observed object motion, without prescribed target poses.

The validator rejects incomplete recordings, unintended arm contacts, and all-zero object point clouds. Negligible motion is rejected only when planar displacement is below 0.005 m and absolute yaw change below 5°, retaining primarily rotational pushes. Changes in the object's up direction above 10° are excluded to retain approximately planar interactions.

Contact labels and local-frame augmentation. During preprocessing, random local point removal simulates incomplete surface observations. The training cloud combines contact points from the first recorded contact frame with farthest-point samples from the remaining observed geometry, yielding $N = 1024$ points. For each retained contact, we identify its five nearest input points. Their index union defines $\mathcal{C}_{\mathrm{gt}} \subseteq \{1, \ldots, N\}$, whose binary mask is normalized into the soft target distribution $y = \{y_i\}_{i=1}^N$.

Each interaction is re-expressed in five local frames, which need not coincide with the model frame used to report x_t. Let $A \in \mathrm{SE}(3)$ map a sampled local frame to world coordinates, with rotation $Q \in \mathrm{SO}(3)$ and origin $a \in \mathbb{R}^3$. The planar origin is sampled as a convex combination of projected hull vertices using Dirichlet weights with unit concentrations. The origin lies at zero height, and the frame yaw is sampled uniformly from $[0, 2\pi)$. World-frame points p_i^w, contacts c^w, and the recorded unit pushing direction $u^w \in \mathbb{R}^3$ are converted as

$$p_i = Q^\top (p_i^w - a), \qquad c = Q^\top (c^w - a), \qquad u_{\mathrm{gt}} = \mathrm{normalize}\big((Q^\top u^w)_{xy}\big), \tag{7}$$

where $(\cdot)_{xy}$ retains the horizontal components. Contact labels are constructed on the transformed cloud before scale normalization.

Let $T_-, T_+ \in \mathrm{SE}(3)$ be the recorded object-to-world transforms before and after the push. The same rigid motion, expressed in the sampled frame, is

$$\Delta T = A^{-1} T_+ T_-^{-1} A = \begin{bmatrix} R_\Delta & t_\Delta \\ 0^\top & 1 \end{bmatrix}, \qquad \Delta p = (t_\Delta)_{xy}, \qquad \Delta\theta = \operatorname{atan2}\big((R_\Delta)_{21}, (R_\Delta)_{11}\big). \tag{8}$$

Here, $R_\Delta \in \mathrm{SO}(3)$ and $t_\Delta \in \mathbb{R}^3$ are the rotation and translation of ΔT. This conversion accounts for displacement of the sampled origin under object rotation, which rotating the object-center displacement alone would omit. Eq. (1) then produces $\bar{\mathcal{P}}$ and ξ using the frame-specific scale D. Augmentation changes the coordinate representation without altering the interaction. At deployment, the same motion conversion and normalization are applied to the current and desired object poses before each policy query.

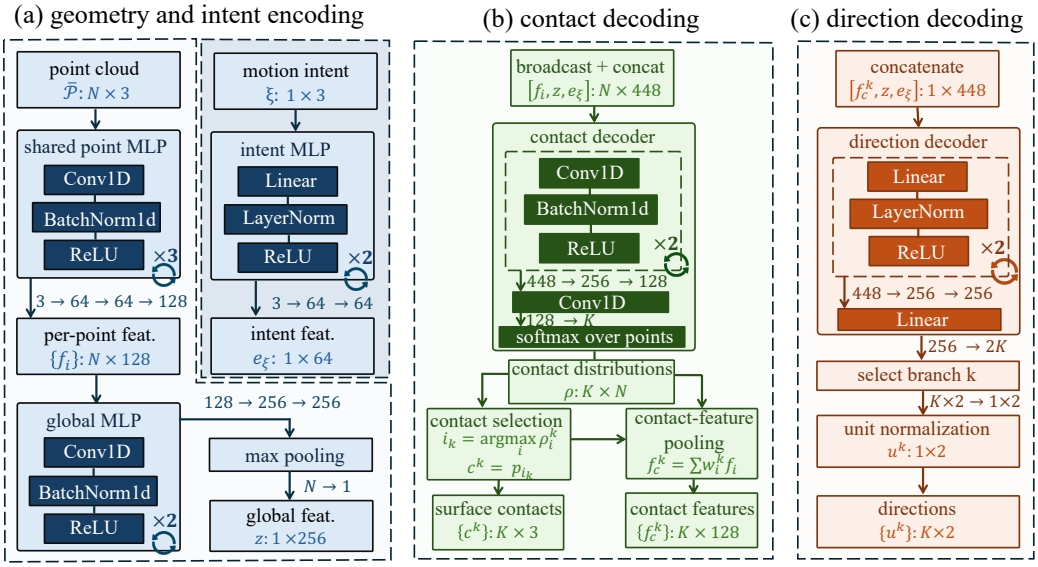

Figure 8: Detailed policy architecture. (a) Geometry and intent encoding. (b) Contact decoding and feature pooling. (c) Contact-conditioned direction decoding for proposal k. Tensor shapes omit the batch dimension. Multipliers indicate stacked layers with separate parameters, not recurrent computation. We use $N = 1024$ points and $K = 8$ proposals.

Network layers. Fig. 8(a) shows the two encoders. The geometry encoder processes $\bar{\mathcal{P}}$ with point-wise MLPs, implemented as kernel-size-one convolutions shared across points. The first MLP produces 128-dimensional point features f_i. The second processes these features before max pooling over points to obtain the 256-dimensional shape feature z. The intent MLP maps ξ to the 64-dimensional embedding e_ξ. Geometry layers use batch normalization and ReLU, while both intent layers use layer normalization and ReLU.

In Fig. 8(b), broadcasting z and e_ξ and concatenating them with each f_i gives 448-dimensional per-point inputs. The contact decoder outputs K logits per point, with softmax over the N points yielding a contact distribution ρ^k for each proposal k. Its hidden layers use batch normalization and ReLU, while the final logit layer is linear. Contact selection retrieves original-scale coordinates, $c^k = p_{i_k} = D\bar{p}_{i_k}$, where $i_k = \arg\max_i \rho_i^k$. Different proposals may share the same contact point. Contact features f_c^k use mixed ground-truth and predicted weights in Stage I, and one-hot weights in Stage II and at deployment.

Fig. 8(c) depicts direction decoding for one proposal. For each k, the shared decoder receives the 448-dimensional concatenation (f_c^k, z, e_ξ) and outputs $2K$ values. These are reshaped into K two-dimensional branches, of which only branch k is retained and unit-normalized to obtain u^k. Applying this operation to all K contact features yields K paired directions. The decoder uses layer normalization and ReLU at its hidden layers and a linear output layer. The decoder shares its

hidden layers across proposals while retaining proposal-specific output branches, rather than using K independent networks.

Stage I: joint contact and direction learning. Contact-feature pooling uses $w_i^k = ry_i + (1-r)\rho_i^k$, where the teacher-forcing ratio $r \in [0,1]$ is linearly annealed from one toward zero. This shifts direction prediction from labeled to predicted contact features. For each training sample, the loss for proposal k combines soft contact cross-entropy and directional cosine loss:

$$\mathcal{L}_k = -\sum_i y_i \log \rho_i^k + \lambda_u \big(1 - \langle u^k, u_{\mathrm{gt}} \rangle\big), \tag{9}$$

where λ_u weights the direction loss and u_{gt} is the local unit planar direction defined in Eq. (7). We aggregate the per-proposal losses with an ϵ-relaxed winner-take-all objective (Rupprecht et al., 2017):

$$\begin{aligned} k^\star &= \arg\min_k \mathcal{L}_k, \\ \mathcal{L}_{\mathrm{I}} &= (1-\epsilon)\mathcal{L}_{k^\star} + \frac{\epsilon}{K-1} \sum_{k \neq k^\star} \mathcal{L}_k. \end{aligned} \tag{10}$$

Here, ϵ is the total loss weight shared uniformly among the non-winning proposals. We use $\lambda_u = 1$ and $\epsilon = 0.05$ and minimize the mean objective over each mini-batch. The best-matching proposal receives most of the weight, while the others retain a small learning signal, encouraging alternative contact modes without averaging their actions. A validation-selected Stage-I checkpoint initializes Stage II.

Stage II: contact-frozen direction calibration. Both encoders and the contact decoder are frozen. We select hard contacts $i_k = \arg\max_i \rho_i^k$, use one-hot pooling weights, and update only the direction decoder. A training push supplies direction supervision only near its recorded contact. With b indexing mini-batch samples, define $h_{bk} = \mathbb{I}[i_{bk} \in \mathcal{C}_{\mathrm{gt},b}]$. For batches with $\sum_{b,k} h_{bk} > 0$, we average over all contact-matched sample–proposal pairs:

$$\mathcal{L}_{\mathrm{II}} = \frac{\sum_{b,k} h_{bk} \big(1 - \langle u_b^k, u_{b,\mathrm{gt}} \rangle\big)}{\sum_{b,k} h_{bk}}. \tag{11}$$

Here, i_{bk} and u_b^k are the selected point index and direction for sample b and proposal k. Samples without a contact-matched proposal contribute no loss, and batches without any such proposal are skipped. The frozen encoders and contact decoder remain in evaluation mode, keeping their normalization statistics fixed. This calibration leaves the learned contact distributions unchanged.

Training configuration and checkpoint selection. Both stages use AdamW (Loshchilov & Hutter, 2019) with batch size 64, weight decay 10^{-4}, and gradient-norm clipping at 1.0. Both use inverse-frequency sampling with replacement to balance object-shape categories. Stage I optimizes Eq. (10) with an initial learning rate of 10^{-3}. Stage II initializes from the selected Stage-I checkpoint and optimizes Eq. (11) with an initial learning rate of 3×10^{-4}. Both use cosine learning-rate decay (Loshchilov & Hutter, 2017) without restarts, with scheduling horizons of 60 and 20 epochs, respectively. The Stage-I teacher-forcing ratio decreases linearly from one at epoch 1 to zero at epoch 30, unless training stops earlier. Validation uses the hard-selected contacts that condition direction prediction at deployment.

For Stage-I checkpoint selection, each validation sample uses its contact and direction labels to identify the best-matching proposal:

$$k_{\mathrm{val}} = \arg\min_k \left[1 - \mathbb{I}[i_k \in \mathcal{C}_{\mathrm{gt}}] + \lambda_u \big(1 - \langle u^k, u_{\mathrm{gt}} \rangle\big)\right]. \tag{12}$$

Unlike the training loss, this criterion uses a binary contact-miss penalty rather than contact cross-entropy. The checkpoint maximizes

$$S_{\mathrm{val}} = H_{\mathrm{oracle}} - \frac{E_{\mathrm{oracle}}}{180^\circ}, \tag{13}$$

where H_{oracle} is the fraction of validation samples whose selected index $i_{k_{\mathrm{val}}}$ lies in $\mathcal{C}_{\mathrm{gt}}$, and E_{oracle} is the mean angular error between the selected direction and u_{gt}, measured in degrees over all validation samples. This label-assisted selection evaluates the available proposals, not the executor's online selection rule.

Stage II selects the checkpoint with the lowest conditional validation direction error. For each validation sample, we retain proposals satisfying $i_k \in \mathcal{C}_{\mathrm{gt}}$, take their minimum angular error, and average over samples with at least one such proposal. Samples without a matched contact are excluded from this conditional metric. Since the encoders and contact decoder are frozen, contact coverage on the fixed validation set remains unchanged throughout Stage II. These label-assisted criteria are used for validation, not for selecting actions during deployment. Checkpoint selection uses validation data only, without reference to test performance.

Training stops after 12 and 6 consecutive epochs without validation improvement in Stages I and II, respectively. For the deployed policy, Stage I runs for 24 epochs and selects epoch 12, which maximizes S_{val}. Stage II then runs for 13 epochs and selects epoch 7, reducing the conditional validation direction error from 2.705° before calibration to 1.567°. All CaI evaluations use this Stage-II checkpoint without further fine-tuning. Selection therefore follows validation performance rather than the lowest training loss or final epoch.

Fig. 9 shows epoch averages of mini-batch training losses on the left axes and validation criteria on the right. Validation uses S_{val} for Stage I and conditional validation direction error for Stage II, whose epoch zero represents the inherited checkpoint.

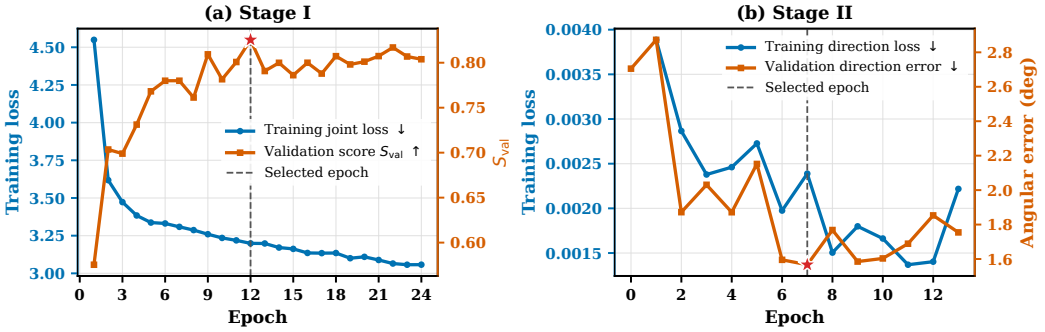

Figure 9: Training losses (blue, left axes) and validation selection metrics (orange, right axes). Dashed lines and red stars mark the selected epochs. Arrows indicate the preferred metric direction.

B.2 Planning and Feasibility Implementation

Geometric discretization. The reference-path planner uses a 0.006 m positional grid. Symmetric objects use circular footprint proxies and require no yaw tracking. Asymmetric objects use square proxies with 48 orientation bins. If no feasible path is found, the planner tests alternative predefined proxy sizes. These are geometric approximations rather than exact mesh footprints. The subsequent positional field uses a circular object envelope and eight neighboring motion directions. Executor costs are evaluated on this planar grid, not on the full orientation state. The reference path's yaw schedule is transferred to the final positional route for asymmetric objects.

Endpoint feasibility probes. Probes are computational tests at hypothetical object positions, not executed pushes or dynamics rollouts. In the Franka Panda implementation, each block of 6×6 fine-grid cells uses the free cell nearest its center as a representative. For each of eight directions, an analytic contact is placed on the rear of the circular object envelope. These coarse probes do not query the policy, and their contacts need not lie on the actual object surface. They use a geometry-based low-contact height, nominally 0.013–0.020 m above the estimated object base, further clipped between 0.008 m and the object height minus 0.003 m. This representative height does not replace the

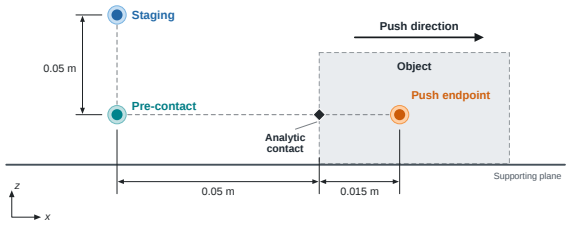

Figure 10: Coarse endpoint probes. IK and full-arm map collision checks are evaluated at three end-effector targets, excluding the hypothetical object. Dashed connectors are not validated motions. The object outline is schematic and not to scale.

policy-selected contact height during execution. The pre-contact point is 0.05 m behind the contact, the staging point is 0.05 m above pre-contact, and the short-push endpoint is 0.015 m beyond the contact. Fig. 10 illustrates these end-effector targets. We test pitches $\{0.3, 0.6\}$ rad by default and yaw offsets $\{0, \pi/2, -\pi/2, \pi\}$ relative to the nominal contact orientation. For each orientation, IK uses the robot's kinematic model to solve sequentially for staging, pre-contact, and push endpoints, using each solution to initialize the next query. At each solution, the Flexible Collision Library (FCL) (Pan et al., 2012) checks the robot's URDF-defined link collision geometry against the map obstacles. These coarse checks exclude the hypothetical object's point cloud. A probe succeeds if at least one orientation admits IK solutions free of robot–obstacle collisions at all three endpoints. These probes check only endpoints, not the connecting motions, and therefore do not establish trajectory feasibility. Their evidence is assigned to the free cells in the corresponding block.

Before passage refinement, the directional multiplier in Eq. (4) is

$$m_{\mathcal{R}}(s,d) = \begin{cases} 1 + \lambda_q \, \delta_q(s,d), & \text{feasible endpoints,} \\ 1 + \lambda_f, & \text{otherwise.} \end{cases} \tag{14}$$

For a feasible probe, $\delta_q(s,d)$ is the minimum mean absolute displacement of the seven arm joints from their configuration at field construction to the staging configuration, over orientations that pass all three endpoint checks, capped at 2.5 rad. The weights $\lambda_q = 0.25$ and $\lambda_f = 2.0$ penalize joint reconfiguration and unsuccessful probes, respectively. The unit baseline preserves geometric path length. Endpoint tests provide only a route prior, while the online executor separately plans and checks the actual motions.

Passage-based cost refinement. We identify constrained passages along the preliminary executor-aware route from the obstacle layout. Obstacle grouping in the pre-built map restricts candidate obstacle pairs, which are then evaluated using geometric clearance and path-traversal criteria. We sample object states along the identified passage axes. At each state, the object point cloud is transformed to the hypothetical pose, and the policy is queried for motion along the passage axis. Candidate pushes must advance through the passage, satisfy object swept-volume clearance, and admit feasible staging, pre-contact, and push configurations. These checks also sample joint-space interpolations between endpoint configurations for robot–obstacle collisions, with a maximum per-joint increment of 0.01 rad. The object swept-volume test assumes a short translation along the candidate pushing direction. It checks geometric clearance for this assumed translation, without predicting the actual object motion under contact dynamics.

The nominal sampling interval and probe push length are 0.03 m and 0.02 m, respectively. We record the fraction of sampled states with a candidate passing these checks and sequences of at least two consecutive such states. A sequence provides local traversal evidence even when no tested candidate passes at other sampled states. For passages with such a sequence, additional edge penalties discourage misalignment with the approach or passage axis and excessive lateral displacement. Otherwise, forward traversal receives a soft penalty while reverse and escape directions remain available. The respective penalty weights for misalignment, lateral displacement, and traversal without such a sequence are 2.0, 3.0, and 4.0. These penalties update $m_{\mathcal{R}}$ before recomputing $V_{\mathcal{R}}$ and $J_{\mathcal{R}}$. The checks are performed separately at each sampled state and do not verify contact-switching motions between states. Passing these checks at consecutive states therefore does not guarantee continuous physical execution under unknown dynamics.

Direction-aware path optimization. For a given set of directional multipliers, reverse Dijkstra search computes $V_{\mathcal{R}}$ using Eq. (5). To penalize turns, we also solve on states (s,h), where $h \in \mathcal{D}$ is the incoming translational direction, not the object yaw. The augmented value satisfies

$$\begin{aligned} J_{\mathcal{R}}(s,h) = \min_{d \in \mathcal{D}(s)} \Big[& \kappa_{\mathcal{R}}(s,d) + J_{\mathcal{R}}(s_d, d) \\ & + \lambda_e \mathbb{I}[\angle(h,d) > 0] + \lambda_a \left(\frac{\angle(h,d)}{\pi} \right)^2 \Big], \end{aligned} \tag{15}$$

with $J_{\mathcal{R}}(s_g, h) = 0$ for every h. Here, $\angle(h,d) = \arccos(h^\top d) \in [0, \pi]$ is the unsigned angle between unit directions. We use $\lambda_e = 0.003$ m and $\lambda_a = 0.012$ m to penalize turn events and squared normalized turn angles, respectively. Successor tracing minimizes the sum of the transition cost, turning penalties, and augmented value at the next state, as in Eq. (15). The first transition has no incoming direction and incurs no turning penalty.

The initial solve uses the endpoint-probe multipliers to obtain a preliminary route. Passage checks refine these multipliers, after which both values are recomputed and the final route is traced. We use $V_{\mathcal{R}}$ and $J_{\mathcal{R}}$ for the final values in the main text. For asymmetric objects, reference yaw is unwrapped and interpolated along the geometric reference path to assign the new route's yaw schedule. This is not a joint optimization of object yaw and executor feasibility. The final path supplies regular motion references, while the scalar field remains the online progress and recovery potential.

B.3 Execution and Recovery Details

Local-to-world transformation. At decision step t, let $R_t \in \mathrm{SO}(2)$ and $\tau_t \in \mathbb{R}^3$ denote the planar orientation and three-dimensional origin of the policy local frame in world coordinates. With $\bar{R}_t = \mathrm{diag}(R_t, 1)$, the world-frame contact action is

$$c_t^{k,w} = \bar{R}_t c^k + \tau_t, \qquad u_t^{k,w} = R_t u^k. \tag{16}$$

The local frame need not coincide with the model frame used to report x_t. The original-scale selected point is preserved, including its vertical coordinate. The executor supplies the remaining wrist orientation and motion trajectory rather than replacing the learned surface contact.

Push horizon. During regular execution, the nominal push horizon is

$$L_t^{\mathrm{nom}} = \mathrm{clip}\big(\alpha V_{\mathcal{R}}(s_t), L_{\min}, L_{\max}\big), \tag{17}$$

where $\mathrm{clip}(v, a, b)$ restricts v to $[a, b]$ and s_t is the current object-center grid cell. We use $\alpha = 0.7$, $L_{\min} = 0.02$ m, and $L_{\max} = 0.06$ m. This is a nominal local horizon, not a predicted object displacement. Workspace, swept-volume, passage, terminal-pose, and recovery constraints further adjust the executed segment. If the field query is unavailable, the Euclidean position error replaces $V_{\mathcal{R}}(s_t)$ in this horizon rule.

Contact execution and continuity. During regular translation, proposals are ranked by alignment with the desired planar motion. During terminal yaw correction, proposals are ranked by a lever-arm-normalized planar moment score in the desired rotation direction, subject to a geometric position-error check. This is a geometric ranking criterion, not a dynamics prediction. After filtering and deduplication, candidates are tested in priority order within the planning budget. The orientation search prefers the gripper's contacting side facing the object, with side or reverse orientations available as fallbacks. An established contact can be retained when the new intent is compatible, the continuation is collision-free, and observed contact following remains valid. During short executed continuation segments, observed object displacement is checked for forward progress and excessive lateral deviation. Otherwise, the executor attempts a checked disengagement before selecting another proposal. Retreat and re-approach are planned and checked, rather than assuming that reversing the previous push is always safe. Collision, contact-following, object-tilt, and force/torque feedback can interrupt individual motion segments. In simulation, force and external joint torque are estimated from simulator contact signals and robot Jacobians, rather than obtained from independent physical force or torque sensors.

Online motion intent. We identify the nearest waypoint on $\Gamma_{\mathcal{R}}$ by planar position and select a downstream waypoint as the translational reference. For asymmetric objects, yaw feedback tracks the downstream path orientation relative to the observed object yaw. This angular correction is bounded and gradually blended toward the goal-yaw error near the target. Symmetric objects use zero angular intent. Together, these references define the desired object pose x_t^{ref}, yielding

$$(\Delta p_t, \Delta\theta_t) = x_t^{\mathrm{ref}} \ominus x_t, \tag{18}$$

where $\ominus$ expresses the desired rigid-body motion in the policy local frame using the transformation in Eq. (8). The resulting motion is normalized using Eq. (1).

Progress and executor audits. Field progress is measured by

$$\Delta V_t = V_{\mathcal{R}}(s_t) - V_{\mathcal{R}}(s_{t+1}), \tag{19}$$

with $\Delta V_t > 0$ indicating progress when both field queries are valid. Terminal yaw correction uses the decrease in absolute yaw error multiplied by the footprint radius. If field queries are unavailable, progress falls back to observed object displacement after execution. Stall detection applies progress tolerances over consecutive updates rather than reacting to a single nonpositive value.

The auditor tests a limited set of policy candidates at the current state and sampled states ahead on the nominal path. A state contributes to the failure streak only when at least one candidate is tested and all tested chains fail. A persistent-infeasibility warning is issued after n_{a} consecutive such sampled states, or n_{a} consecutive failed audits of the current state. Empty tested-candidate sets break the corresponding failure streak and are handled separately. The main configuration uses $n_{\mathrm{a}} = 4$. This threshold governs audit warnings, not every recovery trigger. Two consecutive audited current-state failures, persistent stalls, blocked path reconnections, or exhausted execution candidates can separately activate recovery. Warnings about future path states can instead trigger a warning response while retaining the nominal path as the guidance reference. During passage alignment and traversal, the local passage controller temporarily suppresses selected global stall, reconnection, and execution-failure triggers to avoid conflicting mode switches.

Field-guided recovery search. Recovery treats $V_{\mathcal{R}}$ as a progress potential. At a valid grid state with a descending neighbor, the reference direction selects the lowest-valued one:

$$d_t^{\mathrm{rec}} = \arg \min_{\substack{d \in \mathcal{D}(s_t) \\ V_{\mathcal{R}}(s_d) < V_{\mathcal{R}}(s_t)}} V_{\mathcal{R}}(s_d). \tag{20}$$

This rule ranks neighbors by their field values without adding the immediate transition cost $\kappa_{\mathcal{R}}(s_t, d)$, so it need not select a Bellman-optimal successor. Repeated application at successive grid states forms a short look-ahead displacement and stops when no lower-valued neighbor exists. Near the field boundary, queries may project to a nearby finite cell only when the connection improves geometric clearance. Unavailable field guidance falls back to the nominal path.

Field descent is a reference rather than the complete recovery policy. When the reference intent is not executable, alternative planar directions are queried through the same policy and robot feasibility checks. During active recovery, the local search may temporarily increase route cost by a bounded amount. For the Franka Panda, the search uses a maximum depth of five edges and expands at most 25 nodes by default, with nominal steps of 0.02 m. Detour states may exceed the search-start route cost by at most 0.04 m. Candidate chains propagate robot configurations and must admit two feasible task-directed continuation steps. These checks assess geometric and kinematic feasibility, not future object motion under interaction dynamics. Recovery retains local motion intents and contact-sector and wrist-orientation preferences for ranking new candidates. Each action is updated and checked using the next observed object pose.

Mode transitions and return to path tracking. The controller distinguishes nominal tracking, warning response, active recovery, and path rejoining. Warning response addresses future feasibility warnings while retaining nominal guidance. Active recovery instead uses field-guided search. Returning to path tracking normally requires a viable local path, recent progress, and fresh feasible action-chain evidence, with no active risk or audit warning. Two consecutive updates satisfying these conditions return the controller to nominal operation. If field guidance is unavailable, candidate or execution failures persist, or the recovery-step budget is exhausted, the controller may instead make a limited attempt to rejoin the path. This tests another intent source and does not certify that recovery has succeeded. Repeated path-rejoining attempts with continued candidate or planning failures can terminate the task as recovery exhaustion. Low progress alone does not satisfy this condition, and the overall task budget also bounds execution.

Fetch execution adaptation. For Fetch, executor-aware planning augments object position with a pushing direction and a discrete relative base–arm configuration template. Transition checks use geometric approximations to assess object, base, and deployed-arm clearance. The executor selects a base pose, arm posture, and torso height to execute the policy-selected contact action. During relocation, it follows a collision-checked base route with the arm retracted, treating the current object as an obstacle. It then establishes contact and executes bounded base-driven or arm-driven pushes under pose and contact feedback. The policy weights and object-relative contact interface remain unchanged, with robot-specific adaptations confined to planning and execution.

C Experimental Details and Additional Results

Evaluation scope. RHP ($h = 1$), RHP ($h = 3$), BIT*–CaI, and CaI are evaluated on all 840 object–map tasks with shared perception inputs, object-symmetry conventions, and success criteria.

Component-ablation configurations. The geometric-field variant in Tab. 5 uses object-only geometric guidance without executor-aware field costs. It retains policy-only contact proposals, per-

sistent contact, online execution checks, and recovery guided by geometric progress and execution feedback. The recovery-disabled variant retains executor-aware planning and uses $n_{\mathrm{a}} = 4$, matching the full configuration, while disabling task-level recovery. Online execution checks and safety handling remain active. The threshold sweep in Fig. 11 separately examines audit sensitivity within the full framework.

C.1 Benchmark Construction

Local pose-control benchmark. We use the three asymmetric geometries in the unseen split of Fig. 4, excluding rotationally symmetric objects because their yaw is not a task objective. Each object starts from the nominal planar pose $(0, 0, 0)$ in an obstacle-free scene. Goals enumerate the Cartesian product of two translation distances, $1.5R$ and $2.5R$, eight bearings spaced by $45°$, and six yaw changes, $\pm 30°$, $\pm 60°$, and $\pm 90°$. The footprint radius R is the maximum distance from the scaled mesh's projected vertices to their planar convex-hull centroid. This gives 96 coupled pose goals per object and 288 tasks in total, rather than separate translation and rotation tests. A fixed mesh-derived reference point cloud is reused across goals for each object. The learned controllers subsample this cloud for policy inference. Task records specify the mesh and scale, initial and goal poses, point-cloud reference, and physical parameters. Success criteria are given in Sec. 4.2, and controller settings are detailed in App. C.4.

Long-horizon pushing benchmark. We procedurally generate obstacle layouts and initial–goal poses for a rectangular pushing workspace. Layouts combine a barrier opening with scattered box obstacles, with initial and goal positions on opposite sides of the barrier. Difficulty increases through more constrained passages and goal access: easy maps have open goal regions, medium maps add a goal-side obstruction, and hard maps place the goal in a U-shaped pocket. Candidate layouts are screened for start–goal clearance, approximate pusher access near the goal, and object-space connectivity. During initial map generation, connectivity is checked by grid search using a reference circular footprint and a workspace reachability mask. The generator also rejects unobstructed straight-line tasks and near-duplicate layouts. These checks assess approximate geometric connectivity, not the feasibility of a complete robot-executable pushing sequence.

The selected maps are rescaled and translated for the evaluation workspace. The final set contains 20 maps per tier, with a 0.219×0.642 m planar workspace and 0.02 m-high obstacles. Fixed YAML specifications define the obstacles, workspace, and initial–goal poses used in evaluation. Pairing these 60 maps with seven seen and seven unseen object geometries gives 840 tasks. Generative AI tools assisted review of rendered maps and YAML specifications for layout plausibility, while geometric checks were performed programmatically. Neither the benchmark layouts nor complete task trajectories are used to train the interaction policy.

C.2 Nonuniform Mass Construction

For the mass-redistribution experiment in Sec. 4.3, we retain 75% of the original mass as a uniformly dense body and assign the remaining 25% to an offset point mass. Let a_x, a_y denote the scaled mesh's axis-aligned bounding-box half-extents in the object frame. The point mass's horizontal offset from the original center of mass is $\frac{0.60}{\sqrt{2}}\big(\min(a_x, R), \min(a_y, R)\big)$, where R is the footprint radius. Its height equals the original center-of-mass height, clipped to 15–85% of the object height above its base. All trials use the `moderate_diagonal` profile with `dynamics_seed=0`, which fixes both horizontal offsets as positive. Each object's mass distribution is therefore fixed across maps, rather than resampled per trial.

The uniform body's mass and inertia about its original center of mass are scaled by 0.75. We compute the combined center of mass by mass-weighted averaging and the full inertia tensor about that center using the parallel-axis theorem. The resulting principal moments and mass-frame pose are assigned to the simulator. The horizontal center-of-mass shift is one quarter of the point mass's horizontal offset and is therefore bounded by $0.15R$. The point mass changes only mass properties, without adding visual or collision geometry. Total mass, object geometry, and friction remain unchanged, and the policy receives no inertial parameters or additional fine-tuning.

C.3 Executor-Audit Sensitivity

We vary the consecutive-infeasibility threshold n_a from one to five. This threshold controls confirmation of executor-audit warnings, rather than all recovery triggers, as detailed in App. B.3.

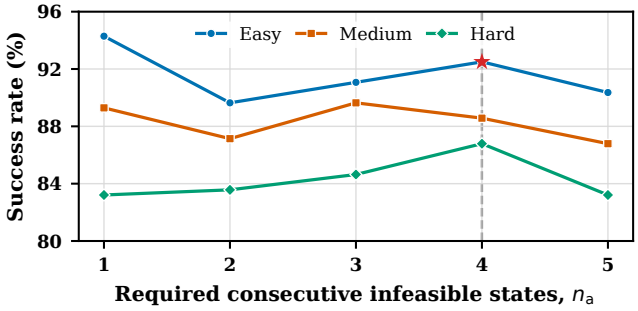

Figure 11: Audit-threshold sensitivity.

Fig. 11 shows that $n_a = 1$ performs best on easy tasks with 94.29% success, whereas $n_a = 4$ achieves the highest hard-task and average success rates of 86.79% and 89.29%, respectively. Increasing the threshold to five reduces these rates to 83.21% and 86.79%. The results suggest a balance between reacting to isolated probe failures and delaying persistent-infeasibility warnings. We use $n_a = 4$ for the full configuration.

C.4 Baseline Implementations

Receding-Horizon Pushing. We use RHP (Yuan et al., 2026) with its native object-space BIT* planner, contact policy, stability-based ranking, and robot motion generator. The policy retains its architecture and training objective but uses the same local-push trajectories and split as CaI. A single validation-selected checkpoint is shared by both validation horizons and the local-control variants below. Each decision validates up to $h \in \{1, 3\}$ virtual pushes, stopping earlier at the path endpoint. Each push includes approach, pre-contact, push, retreat, and lift motions. Only the first validated push is physically executed before observing the object again. Thus, h counts virtual pushes, not waypoints or executed actions. Failed validation triggers replanning, with up to four attempts per decision and bounded memory of five failed poses and five failed transitions. Its object planner tries point-cloud collision-inflation radii of 0.05, 0.02, and 0.01 m in order.

BIT*–CaI. This hybrid uses RHP's object planner and failure-guided replanning with the fixed CaI policy and executor. It retains all K proposals, policy-selected contact heights, end-effector orientation search, online feasibility checks, and persistent contact. Native waypoint queries generate motion intents, and replanning occurs after waypoint completion or failed or stalled interactions. No executor-aware field, virtual action-chain validation, or field-guided recovery is used. In Eq. (17), the distance from the current object center to its nearest projection on the path, plus the remaining path arc length, replaces $V_{\mathcal{R}}(s_t)$, retaining the same gain, bounds, and local safety adjustments. To accommodate observed poses within the native planner's clearance buffer, this hybrid additionally tries positive inflation radii of 0.005, 0.002, 0.001, and 0.0005 m. Collision checking remains active. If either the start or goal state remains collision-invalid at the minimum radius, planning terminates. This is a pipeline comparison, not an isolated planner ablation.

RHP-Policy and local contact selection. RHP-Policy removes global planning and robot motion generation, applying its stability-ranked Top-1 action through the ideal spherical pusher. Variants (S) and (P) differ only in push-length assignment. Variant (S) shares CaI's nominal length of $0.55R$, where R is the footprint radius. When the absolute yaw error is at most 10°, the position-error magnitude clipped to $[0.08R, 0.55R]$ determines the length. Variant (P) instead uses the policy prediction after restoring physical units, without horizon clipping. A nonpositive or nonfinite prediction terminates the trial as a failure. Neither rule modifies the selected contact or direction. For CaI, the K proposals are ranked using a single cosine score that compares the concatenated planar pushing direction and radius-normalized signed moment about the object origin with the residual pose error, whose translation is also normalized by R. This ranking is purely geometric and does not require a forward model of the interaction dynamics.

Push Anything. We use the non-learning Sampling C3+ controller (Bui et al., 2026) with the object mesh and current state, retaining its mesh-normal contact sampling and receding-horizon contact optimization. The adapter replaces the robot reach region with the benchmark pusher workspace and adjusts the sampling clearance to the shared sphere radius. The reported experiments retain controller-specific contact-transition routines.

C.5 Comparison with Geometric Pushing

Geo-Analytic follows object-only geodesic guidance, pushing toward the footprint center along the motion intent. It checks robot feasibility online without global executor costs. Since it lacks explicit yaw control, we compare complete pipelines on shared uniform-density tasks involving symmetric objects. Success criteria and metrics follow Sec. 4.3.

Table 7: Geometric pushing comparison.

Method	Easy	Medium	Hard	Score
	SR (%) / Avg. time (s)			
Seen symmetric objects				
Geo-Analytic	80.00 / 124.86	77.50 / 137.41	63.33 / 175.30	53.011
CaI	**93.33 / 136.74**	**91.67 / 162.50**	**93.33 / 173.90**	**67.606**
Unseen symmetric objects				
Geo-Analytic	68.33 / 101.00	78.33 / 149.33	66.67 / 185.12	52.066
CaI	**88.33 / 195.89**	**83.33 / 177.25**	**73.33 / 214.57**	**53.067**

Tab. 7 shows score increases from 53.011 to 67.606 on seen objects and from 52.066 to 53.067 on unseen objects. The largest success-rate gain occurs on hard seen-object tasks, from 63.33% to 93.33%. On unseen objects, longer mean completion times over successful trials limit the aggregate-score gain despite higher success rates. For successful CaI trials on the shared task set, the runtime breakdown in App. C.6 attributes 23.72% of mean completion time to pre-execution planning and 0.35% to separately timed policy queries.

C.6 Runtime Analysis

Tab. 8 decomposes mean completion time for successful CaI trials on the shared symmetric-object tasks. Executor-Aware Planning covers initial field and nominal-path construction. Closed-Loop Composition and Recovery covers online intent generation, feasibility audits, and recovery, excluding separately reported policy inference and robot motion planning. Policy queries within initial planning and online audits are included in those stages. Embodiment-Specific Execution includes robot motion planning and physical execution, the latter measured in simulated control-step time.

Table 8: Runtime decomposition of CaI in the geometric pipeline comparison.

Method component	Time (s)	Share (%)
Executor-Aware Planning	40.13	23.72
Closed-Loop Composition and Recovery	7.32	4.33
Inverse Interaction Policy	0.60	0.35
Embodiment-Specific Execution	121.08	71.59
Robot motion planning	5.91	3.49
Physical execution	115.17	68.10
Total	169.13	100.00

The mean total is 169.13 s. Initial planning takes 40.13 s (23.72%), and separately timed policy queries account for 0.60 s (0.35%). Physical execution dominates at 115.17 s. All components are averaged over the same successful trials. Rounded values may not sum exactly to the reported total.

The fields are constructed once and reused throughout each task. Their construction cost could be amortized across repeated executions with unchanged planning inputs through caching. All reported completion times include initial field and path construction.

C.7 Additional Real-World Experiments

Object point clouds are reconstructed offline using SAM 3 (Carion et al., 2025) segmentation and VGGT-Omega (Wang et al., 2026) reconstruction, and provided with scene maps. During execution, object poses estimated from AprilTag observations across three Intel RealSense D455f RGB-D cameras are fused in the robot base frame.

Success requires planar position error below 0.05 m and tilt below $20°$, measured from the fused object pose. Asymmetric objects additionally require wrapped absolute yaw error below $15°$.

Fig. 12 supplements the results in Tab. 3 with execution sequences using the same simulation-trained policy without fine-tuning. Videos are provided in the supplementary material.

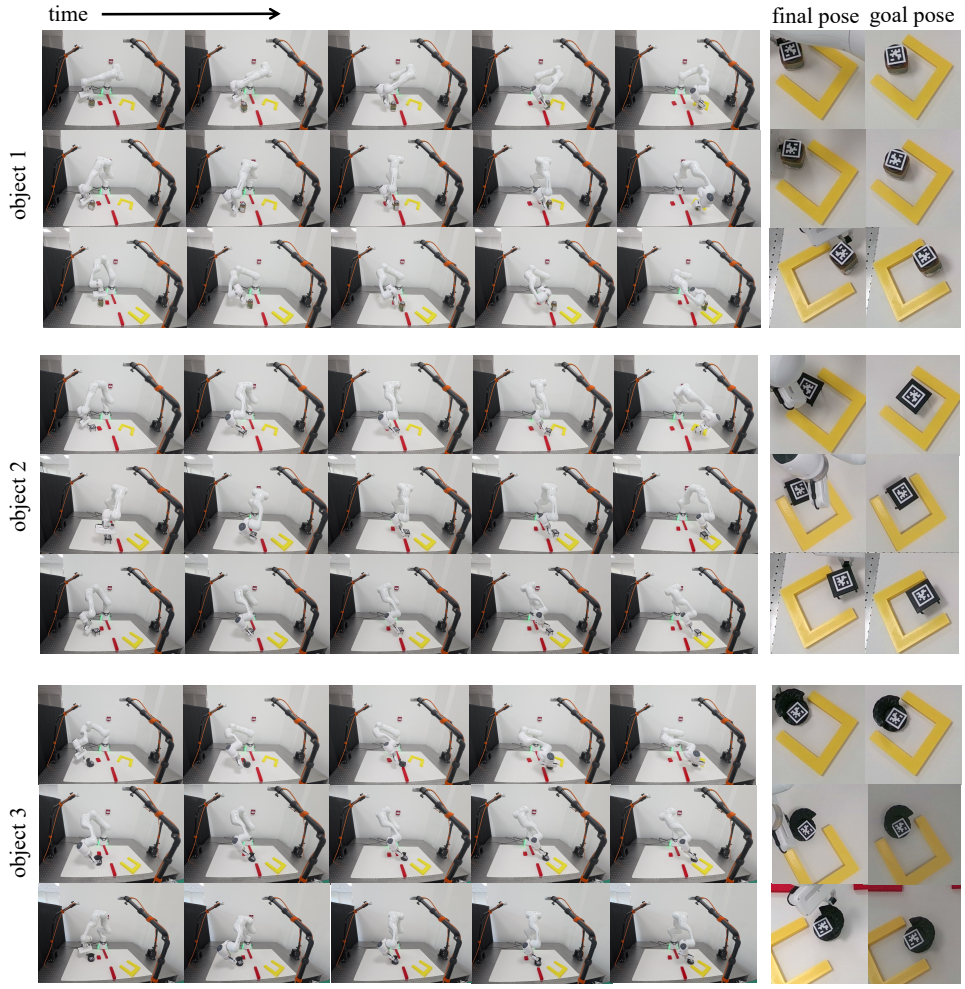

Figure 12: Additional real-world pushing with a Franka Research 3. Row groups correspond to Objects 1–3, with three trials each. Each row shows five execution snapshots from left to right, followed by the final and goal object poses.

D Limitations

Our real-world evaluation assumes access to object and scene geometry acquired in advance, along with online pose estimates. Executor-aware fields guide planning through planar geometric approximations and finite contact probes, rather than certifying complete execution, making online motion validation and feedback essential.