DYNAMICMANIP: ENABLING DYNAMIC MANIPULATION FROM A SINGLE STATIC DEMONSTRATION

Anonymous authors
Paper under double-blind review

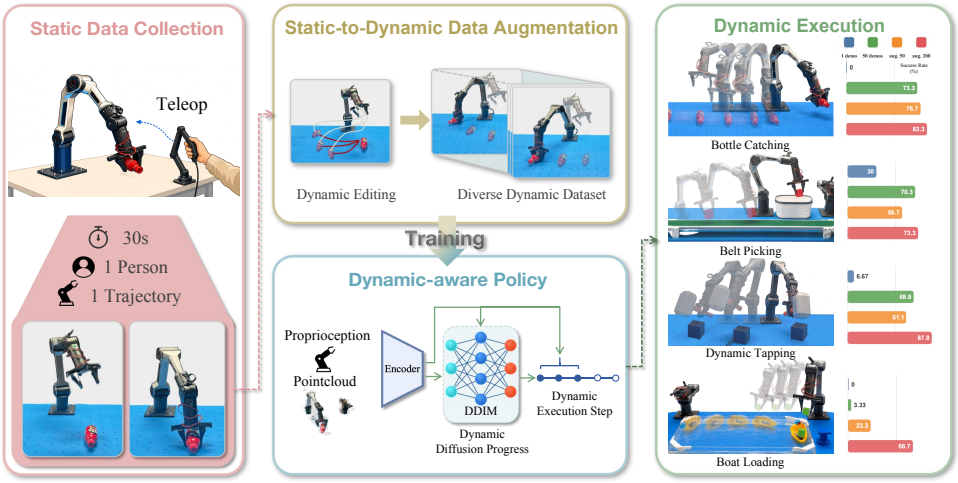

Figure 1: **DynamicManip** generates diverse dynamic manipulation data from a single static task demonstration in the real world. Leveraging the augmented data, our dynamic-aware adaptive policy enables generalizable dynamic manipulation with efficient inference.

ABSTRACT

Dynamic manipulation is a critical capability for robots operating in complex and dynamic environments, where robots must interact with objects that are moving or require rapid adjustments. However, learning models for dynamic manipulation tasks face two major challenges: (1) the combinatorial complexity of dynamic scenarios leads to substantial data requirements, and (2) rapid variations in dynamics require real-time and accurate policy execution. In this paper, we propose **DynamicManip** to address these challenges through an efficient data augmentation pipeline and a low-latency imitation policy. We first propose a static-to-dynamic augmentation pipeline that synthesizes diverse dynamic manipulation demonstrations from a single static demonstration. Second, we introduce a dynamic-aware adaptive policy that adjusts its inference frequency according to task dynamics, enabling responsive and effective dynamic manipulation. Third, we build a dynamic manipulation benchmark, which includes diverse dynamic tasks with an automatic evaluation system for scalable and consistent assessment. Extensive experiments in both simulation and the real world demonstrate that DynamicManip not only provides significant improvements in data efficiency but also achieves better performance in dynamic manipulation tasks, with a mean success rate 18.4 percentage points higher and policy-query latency 32.9% lower.

1 INTRODUCTION

A long-term goal in robotics is to deploy robots that assist people in everyday environments (Argall et al., 2009; Billard & Kragic, 2019; Wei et al., 2025a). To realize this vision, robots must handle

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.

moving targets, such as catching a flying ball or stopping a glass from slipping off a table (Zhang et al., 2025a). These tasks require robots to continuously perceive, track, and respond to moving objects through closed-loop perception-action cycles.

However, dynamic manipulation remains significantly challenging compared to static tasks. Two fundamental challenges stand in the way (Xie et al., 2026). First, collecting data for dynamic tasks is inherently difficult, owing to the combinatorial explosion of object velocities, trajectories, and robot response timings (Lin et al., 2025b; Zhao et al., 2026). Second, dynamic manipulation demands real-time model inference to cope with rapid state variations (Team, 2025; Zhang et al., 2025b; Liao et al., 2025; Black et al., 2025; Sendai et al., 2025). Traditional methods typically train on quasi-static datasets and trade inference speed for precision, making them less suitable for dynamic tasks (Chi et al., 2025).

To overcome these limitations, we present **DynamicManip**, which addresses the challenges of dynamic manipulation learning through (1) a static-to-dynamic data augmentation pipeline that synthesizes diverse dynamic demonstrations from a single static trajectory while preserving the robot behavior patterns and underlying physical consistency; and (2) a dynamic-aware adaptive policy that adapts its inference rate to evolving manipulation dynamics, improving responsiveness and efficiency while maintaining or improving task success in our experiments.

For data augmentation, we introduce a static-to-dynamic pipeline that flexibly modifies object dynamics and robot motion. Specifically, the pipeline first reconstructs the manipulation environment, allowing point cloud rendering of both objects and the robot under arbitrary configurations. It then edits object trajectories and performs motion planning to synthesize paired observations and actions with geometrically feasible robot-object interaction trajectories.

Based on the augmented data, DynamicManip improves responsiveness by adaptively adjusting its inference strategy across manipulation stages. Specifically, we define a dynamic-stage indicator that specifies whether each stage requires high or low computational effort. Our augmentation pipeline automatically annotates the dynamic-stage indicator using heuristics from robot-object interaction dynamics. A lightweight auxiliary head is then trained to predict the indicator at test time, enabling the policy to adjust diffusion denoising steps or action-chunk execution adaptively.

To support our framework, we introduce the DynamicManip Benchmark. Built on RoboTwin 2.0 (Chen et al., 2025), our benchmark provides configurable dynamic task environments for data collection and automatically evaluates task success rates and response latency.

Simulation and real-world experiments show that DynamicManip expands a single static human demonstration into a diverse dynamic dataset, reducing repeated dynamic teleoperation for the evaluated tasks. Moreover, our proposed policy achieves substantially faster execution while maintaining high manipulation precision, enabling responsive closed-loop control in dynamic scenarios and improving task success rates across diverse dynamic manipulation tasks.

2 Related Work

2.1 Robot Manipulation

Robot manipulation is a central research area in robotics and a key pathway toward general-purpose embodied intelligence (Bai et al., 2025; Wei et al., 2025a; Billard & Kragic, 2019). Recent progress in Imitation Learning (IL) (Zhao et al., 2023; James et al., 2020; Liu et al., 2023; Brohan et al., 2022; Zitkovich et al., 2023) and Vision-Language-Action (VLA) models (Black et al., 2024; Intelligence et al., 2025; 2026; Liu et al., 2024; Kim et al., 2024; Bjorck et al., 2025) has shown strong capability in learning action distributions from large-scale expert demonstrations and predicting action sequences conditioned on observations, while complementary efforts have explored specialized manipulation capabilities such as language-guided dexterous grasp generation (Wei et al., 2024) and foundation-model-based grasp transfer with force-aware adaptation (Wei et al., 2025b). Most existing manipulation methods and benchmarks are evaluated in static or slowly changing settings. These settings simplify teleoperation by limiting variation in object motion and allow policies to tolerate moderate inference latency.

2.2 Dynamic Tasks

Dynamic manipulation is pervasive in everyday and industrial environments, from catching a slipping glass to sorting products on moving conveyors. Tracking and responding to rapidly or unpredictably moving objects requires rapid perception, predictive planning, and low-latency action generation within closed-loop perception-action cycles. Dynamic tasks also require demonstrations covering diverse object motions, interaction timings, and robot-object temporal alignments. Despite its importance, research on general dynamic manipulation under uncertain motion and fine-grained contact constraints remains relatively limited. Existing methods largely focus on static settings or structured scenarios with predictable motion, such as DBC-TFP (Zhang et al., 2025a) and GEM (Li et al., 2025) in conveyor-like environments. Concurrent VLA approaches, including RDT-2 (Team, 2025), RTVLA (Ma et al., 2025), and VLASH (Tang et al., 2025), demonstrate real-time interaction with fast-moving targets but operate with large contact margins and lack precise manipulation. Although works such as DynamicVLA (Xie et al., 2026; Kim et al., 2014; Akinola et al., 2021) attempt to address this problem, a significant gap remains between current capabilities and the demands of real-world dynamic manipulation.

2.3 Data Collection and Augmentation

High-quality and diverse demonstration data are critical for learning robust manipulation policies(Belkhale et al., 2023; Khazatsky et al., 2024; Lin et al., 2025a; Zhu et al., 2025; Lyu et al., 2026; Xiao et al., 2025), yet collecting such data remains expensive(Fang et al., 2023; O'Neill et al., 2024; Geng et al., 2025; Lin et al., 2025b), especially for dynamic tasks with large combinatorial variation(Gao et al., 2026; Ye et al., 2026; Lyu et al., 2025). Data augmentation has therefore emerged as a promising direction to scale limited demonstrations(Yu et al., 2025). Existing approaches can be broadly categorized into several groups. 2D visual augmentation methods (e.g., GenAug (Chen et al., 2023; Mandlekar et al., 2023)) modify appearance through image-level transformations but do not enforce 3D consistency. Implicit 3D reconstruction methods (e.g., RoboSplat (Yang et al., 2025)) improve visual fidelity via neural scene representations, yet typically do not adapt action trajectories accordingly. Geometric augmentation approaches (e.g., RoCoDA (Ameperosa et al., 2025)) apply SE(3)-equivariant transformations in state space but are largely limited to rigid global transformations. Simulation-based methods (e.g., FAR-Dex (Bai et al., 2026)) can generate physically plausible trajectories but may suffer from sim-to-real discrepancies. Grasp-level synthesis methods such as BiDexGrasp (Lin et al., 2026) generate physically feasible bimanual grasp configurations across diverse object geometries and sizes, but do not model temporally coordinated robot trajectories under evolving object motion. At the trajectory level, more recent work such as DemoGen (Xue et al., 2025) explores trajectory editing for tabletop tasks, but only generates quasi-static variations and does not address the temporal dynamics critical for dynamic manipulation. In contrast, our method turns minimal static demonstrations into geometrically feasible and physically consistent dynamic demonstration data by jointly modifying object motion and adapting the corresponding robot actions.

3 Method

3.1 Overview

We study dynamic manipulation from extremely limited demonstrations, where the robot must learn to interact with moving objects while maintaining low perception-action latency. DynamicManip addresses this problem by coupling data generation with adaptive policy inference: a static-to-dynamic augmentation pipeline converts a single static demonstration into diverse dynamic episodes with timestep-level interaction-stage labels, and a dynamic-aware imitation learning policy uses these labels to learn stage prediction and automatically adjust its inference behavior from online observations during deployment.

3.2 Static-to-Dynamic Data Augmentation

Our augmentation pipeline starts from the observation that a static demonstration already specifies *how* the robot should interact with the object, while dynamic variations arise mainly from *where* and *when* the interaction occurs. Therefore, instead of collecting dynamic demonstrations directly,

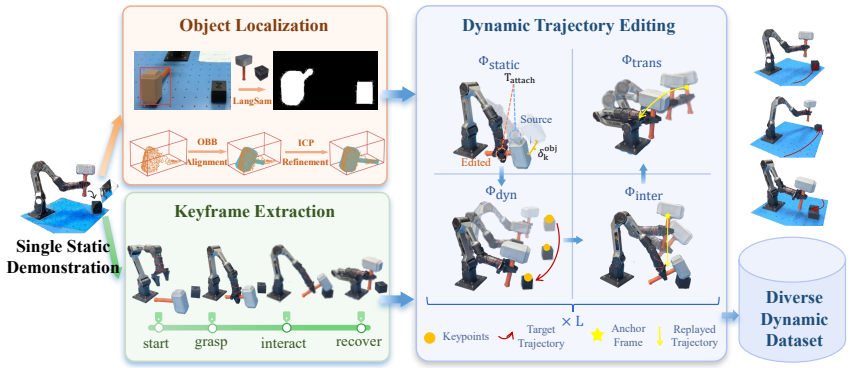

Figure 2: The pipeline of our static-to-dynamic data augmentation. From a single static demonstration, we localize the manipulation object and target and reconstruct their geometry. We then extract action-based keyframes, where labels such as start, grasp, interact, and recover only serve as intuitive annotations for segmenting the source motion into reusable pieces. Dynamic trajectory editing then organizes stages such as Dynamic Target Alignment (Φ_{dyn}) to synthesize diverse trajectories, which are generated through motion planning and replaying local interaction segments.

we preserve the local interaction patterns in the source episode and edit the object motions, phase anchors, and transition targets to synthesize new dynamic episodes. Given a single source episode $\mathcal{E}_{\text{src}} = \{(\mathbf{p}_t, \mathbf{e}_t, \mathbf{q}_t)\}_{t=0}^{T}$, where $\mathbf{p}_t \in \mathbb{R}^{N\times 3}$ is the scene point cloud, $\mathbf{e}_t \in SE(3)$ is the end-effector pose, and $\mathbf{q}_t \in \mathbb{R}^d$ is the robot state, our pipeline generates K augmented dynamic episodes through geometric reconstruction, keyframe extraction, and dynamic trajectory editing.

Geometric reconstruction. Single-view point clouds are incomplete and view-dependent, so spatial edits can expose surfaces occluded in the source demonstration. We generate a CAD mesh from RGB images, align it to the observed object point cloud $\mathcal{P}_o$ using OBB-based coarse initialization followed by multi-resolution ICP refinement (Besl & McKay, 1992), and resample it into a canonical object point cloud. For the robot, we use the URDF model and joint configuration $\mathbf{q}_t$ to compute link poses via forward kinematics and sample the transformed link meshes into $\mathcal{P}_t^{\text{robot}}$.

Keyframe Extraction. We annotate sparse keyframes in the single static demonstration to mark the boundaries between approach, interaction, and disengagement phases. These keyframes provide anchor frames for transforming and recomposing the motion segments across augmented episodes. Keyframe annotation takes approximately two minutes of human effort (Appendix Table 7).

Dynamic trajectory editing. Given the editable scene representation and the annotated phase segments, we synthesize dynamic episodes by reusing the source interaction segments under newly sampled object poses, object motions, and transition targets. For the k-th augmented episode, we sample a set of task-specific augmentation parameters $\xi_k \sim \mathcal{D}$, which specify object poses, object motion trajectories, phase anchors, and transition goals. The sampled parameters determine how each reusable segment is transformed, planned, or replayed in the new dynamic scene. Guided by ξ_k, we instantiate and compose four phase operators to generate a complete dynamic trajectory:

Static Object Acquisition ($\mathbf{\Phi}_{\text{static}}$) Many dynamic tasks begin with a static preparation stage, where the robot grasps or picks up an object before interacting with a moving target. Since the object is static, this segment can be augmented by applying a rigid transformation $T_k^{\text{static}} \in SE(3)$ to the entire phase: $\mathbf{p}'_t = T_k^{\text{static}}\mathbf{p}_t$, $\mathbf{e}'_t = T_k^{\text{static}}\mathbf{e}_t$. This transformation increases spatial diversity while preserving the relative motion pattern between the robot and object.

Dynamic Target Alignment ($\mathbf{\Phi}_{\text{dyn}}$). In this phase, the robot aligns its end-effector with a moving object or a predicted target before executing the interaction. Given a synthesized object motion $T_{\text{obj}}^k(t)$, we first compute a time-varying interaction target $\mathbf{g}^k(t)$ from the task-specific prediction rule. For example, $\mathbf{g}^k(t)$ may correspond to an object-frame relative pose or a predicted landing point derived from a parabolic object trajectory. A motion planner then generates a smooth trajectory

to reach these target-conditioned poses while satisfying robot constraints. This construction reuses the same static demonstration across diverse object motions while preserving smooth, kinematically feasible alignment motions.

Contact Interaction Execution ($\mathbf{\Phi_{inter}}$). This phase executes short contact-rich motions that are difficult to specify with motion planning alone, such as gripper closure during dynamic grasping or hammer strikes. Let A be the anchor frame of the source interaction segment, such as the object or end-effector pose at the contact keyframe. For the k-th augmented episode, the edited trajectory defines a new anchor frame A_k, and we transfer the local interaction motion by $\mathbf{e}'_t = A_k A^{-1} \mathbf{e}_t$. This anchor-based replay preserves the fine local motion pattern while allowing the same interaction to occur at different object positions and orientations.

Pose-Conditioned Transition ($\mathbf{\Phi_{trans}}$). In this phase, the robot moves between reusable motion segments, such as transporting an object, reaching the next interaction pose, or returning to the reset configuration. Given a recorded keyframe pose $\mathbf{e}_{\text{key}}$ and a pose offset $T_k^{\text{trans}} \in SE(3)$, we define the transformed target pose as $\mathbf{e}_{\text{key}}^k = T_k^{\text{trans}} \mathbf{e}_{\text{key}}$. A motion planner then generates a feasible trajectory from the current state to $\mathbf{e}_{\text{key}}^k$. This transition phase connects the edited interaction segments into a continuous executable trajectory.

Composing trajectories with stage labels. The four phase operators can be composed to instantiate complete dynamic manipulation trajectories. For example, a common trajectory structure is $\Phi_{\text{static}} \rightarrow (\Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}} \rightarrow \Phi_{\text{trans}})^L$, where L denotes the number of interaction cycles. Different tasks can be generated by reordering, skipping, or repeating these operators according to the task structure. During composition, each timestep inherits its generating phase operator's identity as a stage label, requiring no additional annotation.

By sampling different augmentation parameters ξ_k, the pipeline produces diverse dynamic trajectories from a single keyframe-annotated static demonstration. The resulting stage-labeled trajectories are used in the next section to train a dynamic-aware policy.

3.3 Dynamic-aware Imitation Learning Policy

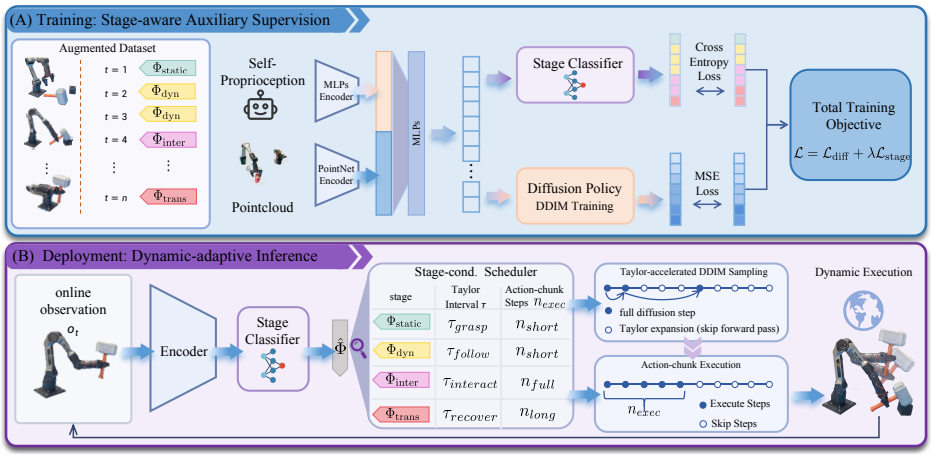

Figure 3: **Overview of the dynamic-aware adaptive policy.** During training, stage labels automatically extracted from augmented trajectories provide auxiliary supervision for stage prediction. During deployment, the predicted stage is used to adaptively regulate the policy inference frequency according to the current task dynamics.

During training, a lightweight auxiliary head learns to predict the interaction stage from visual observations using stage labels from the augmented data. During deployment, the predicted stage selects the denoising schedule and action-chunk execution length.

Dynamic-stage Awareness Learning To make the policy aware of its interaction stage, we supervise it with the stage labels automatically produced by the augmentation pipeline. Specifically, every

augmented trajectory carries a stage label $\ell_t \in \{\Phi_{\text{static}}, \Phi_{\text{dyn}}, \Phi_{\text{inter}}, \Phi_{\text{trans}}\}$ for each timestep t, reflecting the interaction logic that produced it. We attach a lightweight classification head h_ϕ to the shared observation encoder. Given encoded features $\mathbf{f}_t$, the head predicts a stage distribution $\hat{\mathbf{p}}_t = h_\phi(\mathbf{f}_t)$ over the four stages, supervised by a cross-entropy loss:

$$\mathcal{L}_{\text{stage}} = -\frac{1}{T}\sum_{t=1}^{T} \log \hat{p}_t(\ell_t). \tag{1}$$

The total training objective combines the diffusion loss with this auxiliary term:

$$\mathcal{L} = \mathcal{L}_{\text{diff}} + \lambda\, \mathcal{L}_{\text{stage}}, \tag{2}$$

where λ balances action fidelity and stage awareness. This auxiliary objective encourages the encoder to encode the current stage, which helps the policy adaptively adjust its inference strategy.

Dynamic-adaptive inference. At deployment, the trained stage classifier predicts the current interaction stage at each policy-query cycle: $\hat{\ell}_t = \arg\max_k \hat{p}_t(k)$. Conditioned on this stage label, the policy dynamically adjusts its computational allocation, including the denoising schedule and the action-chunk execution length.

The denoising schedule controls how frequently the diffusion model performs full forward computation during DDIM sampling (Song et al., 2020). For intermediate denoising steps where full computation is skipped, we adopt a Taylor-based acceleration mechanism (Liu et al., 2025) that treats each residual block output $\mathbf{y}(s)$ as a smooth function of the diffusion step index s and approximates it via a Taylor expansion around a recently computed anchor step s_0:

$$\mathbf{y}(s) \approx \sum_{k=0}^{K} \frac{1}{k!}\, \mathbf{y}^{(k)}(s_0)\, (s - s_0)^k, \tag{3}$$

where the derivatives $\mathbf{y}^{(k)}(s_0)$ are estimated via backward finite differences between consecutive full-computation steps (Appendix A.3). By skipping expensive forward passes at intermediate denoising steps, this approximation reduces inference latency while maintaining action accuracy.

We also adapt the action-chunk execution length: the number of predicted actions executed before the next policy query. Shorter chunks provide more frequent feedback, whereas longer chunks reduce inference calls and improve motion continuity.

The predicted stage $\hat{\ell}_t$ determines three scheduling parameters: the TaylorSeer computation interval τ, the Taylor expansion order K, and the action-chunk execution length n_{exec}. We assign these parameters using heuristic stage-conditioned rules, based on the expected trade-offs between responsiveness, motion consistency, and inference efficiency in each stage:

In Φ_{static}, we use frequent full recomputation and short execution chunks to avoid early grasp failure.

In Φ_{dyn}, we use a large Taylor interval τ_{follow} to reduce closed-loop latency and short action-chunk execution n_{short} for frequent feedback while tracking moving targets.

In Φ_{inter}, we use a smaller interval τ_{interact} and a longer action-chunk prefix $n_{\text{long}} > n_{\text{short}}$ to preserve contact quality and motion smoothness.

In Φ_{trans}, returning has the least stringent timing requirements, so we use the largest interval, higher-order expansion, and full action-chunk execution to reduce compute overhead.

Overall, this heuristic stage-aware scheduler preserves frequent feedback in dynamic stages and reduces computation in less time-critical stages.

4 DynamicManip Benchmark

To support the evaluation of dynamic manipulation capabilities, we introduce the DynamicManip Benchmark based on the RoboTwin 2.0 platform. We design five dynamic task environments for convenient data collection and policy evaluation, as illustrated in Figure 4.

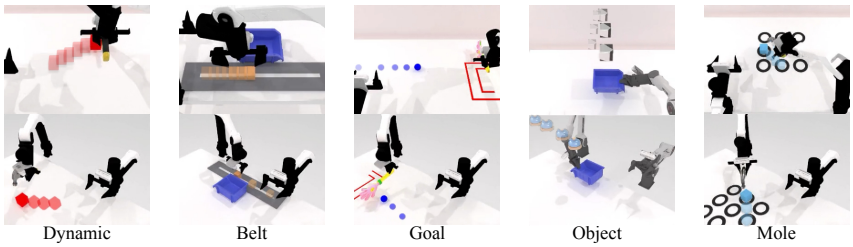

Figure 4: Visualization of the tasks in the DynamicManip Simulation Benchmark.

Data Collection. We extend the RoboTwin 2.0 pipeline with a dynamic-aware heuristic planner that tracks moving targets, predicts future positions, and triggers grasps, and a contact-release mechanism that transfers control from scripted trajectories to the physics engine upon physical contact. We record expert demonstrations at 30 FPS.

Latency-Aware Automatic Evaluation. Our task-specific evaluator detects success using physical signatures, such as state-machine-based contact detection for tapping. To account for policy inference latency, we advance the simulator by $N_{\text{delay}} = \text{round}(t_{\text{infer}}/\Delta t_{\text{sim}})$ steps before applying each predicted action, so the scene evolves during inference. Manual inspection of 100 episodes per task confirmed perfect agreement with human judgment.

5 Experiments

We comprehensively evaluate the proposed DynamicManip framework in both simulated and real-world dynamic environments. Our experiments are designed to answer the following key questions:

Q1: Regarding data efficiency, **can our augmented data achieve comparable performance to conventional demonstrations** while reducing the collection cost?

Q2: In dynamic scenarios, **how does our dynamic-aware policy perform** in terms of reaction speeds and task success rates compared to the baseline?

Q3: **How do the key data-generation components and inference-scheduling variants affect task performance**?

5.1 Experiment Settings

Simulation Setup. We evaluate DynamicManip on the DynamicManip Benchmark. We use two ARX-X5 robotic arms with a pair of parallel grippers and a RealSense D435 head camera for visual perception. More details on the simulation environment can be found in the supplementary material.

Real-World Setup. We evaluate on two AgileX Piper robots across four tasks: Belt Picking, Dynamic Tapping, Boat Loading, and Bottle Catching. We report success rates over 30 trials per task; training data quantities are specified in Table 2. Appendix C.2.1 details the apparatus and data collection, with the setup and task execution shown in Figures 7 and 6.

Evaluation Metrics. *Success Rate* (Suc.) measures the proportion of successful task trials. Successful trials include cases such as hitting a moving target, catching an object, or intercepting a goal-bound item.

Training and Inference Details. We train for 300 epochs and use DDIM sampling with 10 denoising steps under our dynamic-aware computation schedule at inference. We use PyTorch with a single RTX 4090 GPU for training and a single RTX 4070 GPU for real-world inference. Training hyperparameters are listed in Appendix A.2.

Baseline. We use 3D-Diffusion-Policy (Ze et al., 2024), reproduced following the official RoboTwin defaults. For data-efficiency experiments, we fix the DP3 configuration and vary the training data. For policy comparisons, we use the same demonstration set and evaluation protocol.

Table 1: Simulation data-efficiency comparison. All entries report success rate (%). Upper rows use simulated demos; lower rows use dynamic episodes generated from one static demonstration.

Setting	Dynamic Tapping	Belt Picking	Object Catching	Mole Whacking	Goal Blocking
1 demo	27	0	8	3	12
10 demos	15	5	54	21	46
50 demos	19	31	37	63	94
200 demos	55	85	50	83	87
1 demo $\xrightarrow{\text{aug.}}$ 50 eps	45	44	70	63	88
1 demo $\xrightarrow{\text{aug.}}$ 200 eps	52	82	64	80	88

5.2 Data Efficiency of Augmented Demonstrations (Q1)

We compare DP3 policies trained on augmented data with those trained on simulated demonstrations in the DynamicManip Benchmark and manually teleoperated dynamic demonstrations in the real world. We report task performance and data-collection cost.

Simulation analysis. A single static demonstration contains useful interaction structure, but training directly on it provides limited coverage of dynamic object motions and interaction timings. We train the same DP3 policy configuration for 300 epochs across all settings, varying only the training data source and scale. As shown in Table 1, policies trained on our augmented episodes achieve success rates comparable to those trained on 200 simulated demonstrations. The gains come from combining geometric reconstruction with trajectory editing, which preserves the local robot-object interaction pattern while varying object motion, spatial placement, and phase timing.

Table 2: Real-world data-efficiency comparison. We report success rates (%) over 30 trials per task and setting. Dynamic Tapping contains three strikes per trial and is evaluated over 90 strike events; the other tasks use one binary outcome per trial. Higher is better.

Training Data	Dynamic Tapping	Belt Picking	Boat Loading	Bottle Catching
Manual dyn. demos (1 demo)	6.67	30.0	0.00	0.00
Manual dyn. demos (50 demos)	68.9	70.0	3.33	73.3
Ours: 1 static demo $\rightarrow$ 50 aug. eps	61.1	56.7	23.3	76.7
Ours: 1 static demo $\rightarrow$ 200 aug. eps	**87.8**	**73.3**	**66.7**	**83.3**

Real-world validation We further evaluate whether the augmented data remain effective in real-world physical setups (Table 2). Starting from a single static demonstration, increasing the number of augmented episodes consistently improves policy performance. With sufficient augmented data, the policy surpasses the success rate of a model trained on 50 manually teleoperated dynamic demonstrations. These results support static-to-dynamic data generation for real-world dynamic manipulation.

Human effort and collection time. Preparing 200 augmented episodes requires one operator, approximately 11 minutes of human effort for the static demonstration and pre-generation processing, and 31 minutes of machine computation. Collecting 200 dynamic demonstrations by manual teleoperation requires two operators and approximately 400 minutes. Appendix Table 6 compares the data preparation costs, with the processing breakdown in Table 7.

5.3 Real-Time Performance of the Dynamic-Aware Policy (Q2)

To isolate the effect of the dynamic-aware policy, we compare DynamicManip with DP3 using the same demonstration set and latency-aware evaluation protocol. Table 3 reports end-to-end policy-query latency and success rate over 100 trials per task.

Table 3: Latency and success-rate comparison between DP3 and our dynamic-aware policy under the Latency-Aware Automatic Evaluation System. Lat. means end-to-end inference latency measured in milliseconds(ms), success rate is evaluated under 100 trials and reported as a percentage(%).

	Dynamic Tapping		**Belt Picking**		**Object Catching**		**Mole Whacking**		**Goal Blocking**	
Method	Lat.	Suc.	Lat.	Suc.	Lat.	Suc.	Lat.	Suc.	Lat.	Suc.
DP3 (Ze et al., 2024)	60.0	20.0	45.0	78.0	61.4	48.0	44.4	38.0	44.6	42.0
Ours	**38.0**	**42.0**	**32.9**	**80.0**	**36.0**	**64.0**	**30.6**	**44.0**	**33.9**	**88.0**

Table 4: Transposed ablation of cumulative data-generation variants and inference schedules (success rate, %). Scheduling variants use the same checkpoint and evaluation episodes. Ours additionally includes local interaction replay and predicted-stage scheduling.

Task	**Cumulative Data Generation**			**Inference Scheduling**			**Ours**
	Spatial Only	Quasi-static	Planned Motion	Uniform DDIM	Fixed Taylor	Short Chunks	
Object Catching	2	26	53	19	34	48	**64**
Goal Blocking	4	34	75	38	67	64	**88**

Across the five tasks, DynamicManip reduces mean policy-query latency by 32.9% and increases mean success by 18.4 points. It improves both metrics on every task, with the largest success gain on Goal Blocking (+46 percentage points). These results show that stage-aware adaptive inference improves real-time responsiveness without trading off task success under latency-aware execution.

5.4 Ablation Studies on Data Generation and Inference Scheduling (Q3)

Table 4 reports two controlled ablations. For data generation, we cumulatively add spatial transforms, quasi-static editing, planned dynamic motion, and local interaction replay under a fixed policy architecture, training configuration, and evaluation protocol. For inference scheduling, we vary only the scheduling strategy, using the same checkpoint and evaluation episodes.

Each successive addition improves both tasks, indicating that the data-generation components are complementary rather than redundant. Planned dynamic motion provides the largest stepwise gain, with further improvement from local interaction replay. Our predicted-stage strategy performs best on both tasks, supporting joint adaptation of denoising, Taylor acceleration, and action-chunk length over a fixed schedule.

6 Conclusion

We present *DynamicManip* to address data scarcity and inference latency in dynamic robot manipulation. We introduce an efficient static-to-dynamic augmentation pipeline that synthesizes diverse, physically plausible demonstrations from only a single static demonstration, alongside a dynamic-aware adaptive policy that automatically regulates inference frequency for responsive closed-loop control. We establish a dynamic manipulation benchmark with an automatic evaluation system for scalable assessment. Extensive experiments in both simulation and real-world settings demonstrate that our approach significantly improves data efficiency and task success rates while achieving faster inference than existing baselines. By effectively bridging the gap between minimal static data and complex reactive execution, our framework represents a robust and scalable step toward deploying robots in unstructured, fast-paced human environments.

AI USE STATEMENT

We used Codex, Claude, and GitHub Copilot to support implementation and statistical analysis, including debugging, refactoring, simulation and parallel data synthesis optimization, data post-processing, visualization, and confidence interval calculations. During early development, we also used these tools to compare pipeline components and made the final selections through empirical testing. We generated object meshes with Hyper3D and inspected and aligned them before use.

For manuscript preparation, we used AI for translation, language editing, formatting, and review. We reviewed the edited text, tested the assisted code, and checked statistical interpretations against the original results. We take responsibility for the paper's content.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce our results. Training hyperparameters are listed in Appendix A, along with the network architecture and the inference procedure. We use PyTorch with a single RTX 4090 GPU for training and a single RTX 4070 GPU for real-world inference, and the augmentation engine is parallelized across a cluster of four NVIDIA RTX 4090 GPUs. Appendix B describes the static-to-dynamic augmentation pipeline, which begins with a single static demonstration and synthesizes massive and diverse augmented episodes through mesh generation, precise mesh alignment, dynamic trajectory editing, and point cloud post-processing. Human teleoperation is limited to one static demonstration per task. The pipeline also requires sparse keyframe annotations and an object mesh generated from a single RGB image, and it synthesizes all dynamic training data from these inputs. Table 7 reports the task-specific setup and data-generation time, including mesh-size correction, phase composition, and motion-distribution specification. Appendix C documents the dynamic manipulation benchmark, encompassing both simulation and real-world environments, together with the task-specific automatic evaluation criteria and confidence intervals. Appendices D–G report additional ablations, out-of-distribution and robustness analyses, additional real-world evaluations, and a comparison with a tracking-and-planning pipeline. Appendix H discusses limitations. We will release the codebase for reproduction and further development upon acceptance; it includes the static-to-dynamic data augmentation pipeline, our dynamic-aware adaptive policy, and the expert demonstration collection and automatic evaluation scripts for the simulation benchmark built on RoboTwin 2.0.

REFERENCES

Iretiayo Akinola, Jingxi Xu, Shuran Song, and Peter K Allen. Dynamic grasping with reachability and motion awareness. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9422–9429. IEEE, 2021.

Ezra Ameperosa, Jeremy A Collins, Mrinal Jain, and Animesh Garg. Rocoda: Counterfactual data augmentation for data-efficient robot learning from demonstrations. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13250–13256. IEEE, 2025.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Shuanghao Bai, Wenxuan Song, Jiayi Chen, Yuheng Ji, Zhide Zhong, Jin Yang, Han Zhao, Wanqi Zhou, Wei Zhao, Zhe Li, et al. Towards a unified understanding of robot manipulation: A comprehensive survey. *arXiv preprint arXiv:2510.10903*, 2025.

Yushan Bai, Fulin Chen, Hongzheng Sun, Yuchuang Tong, En Li, and Zhengtao Zhang. Far-dex: Few-shot data augmentation and adaptive residual policy refinement for dexterous manipulation. *arXiv preprint arXiv:2603.10451*, 2026.

Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Data quality in imitation learning. *Advances in neural information processing systems*, 36:80375–80395, 2023.

Paul J Besl and Neil D McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446): eaat8414, 2019.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

R Qi Charles, Hao Su, Mo Kaichun, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 77–85. Ieee, 2017.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

Zoey Chen, Sho Kiami, Abhishek Gupta, and Vikash Kumar. Genaug: Retargeting behaviors to unseen situations via generative augmentation. *arXiv preprint arXiv:2302.06671*, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.

Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, et al. Dreamdojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.

Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, et al. Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

Hyper3D. Hyper3d: Ai-powered 3d model generator, 2024. URL https://hyper3d.ai/.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Seungsu Kim, Ashwini Shukla, and Aude Billard. Catching objects in flight. *IEEE Transactions on Robotics*, 30(5):1049–1065, 2014.

Zhuoling Li, Xiaoyang Wu, Zhenhua Xu, and Hengshuang Zhao. Train once, deploy anywhere: Realize data-efficient dynamic object manipulation. *arXiv preprint arXiv:2508.14042*, 2025.

Aileen Liao, Dong-Ki Kim, Max Olan Smith, Ali-akbar Agha-mohammadi, and Shayegan Omidshafiei. Delay-aware diffusion policy: Bridging the observation-execution gap in dynamic tasks. *arXiv preprint arXiv:2512.07697*, 2025.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=pISLZG7ktL.

Mu Lin, Yi-Lin Wei, Jiaxuan Chen, Yuhao Lin, Shuoyu Chen, Jiangran Lyu, Jiayi Chen, Yansong Tang, He Wang, and Wei-Shi Zheng. Bidexgrasp: Coordinated bimanual dexterous grasps across object geometries and sizes. *arXiv preprint arXiv:2604.06589*, 2026.

Yuhao Lin, Yi-Lin Wei, Haoran Liao, Mu Lin, Chengyi Xing, Hao Li, Dandan Zhang, Mark Cutkosky, and Wei-Shi Zheng. Typetele: Releasing dexterity in teleoperation by dexterous manipulation types. *arXiv preprint arXiv:2507.01857*, 2025b.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15853–15863, 2025.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Jiangran Lyu, Ziming Li, Xuesong Shi, Chaoyi Xu, Yizhou Wang, and He Wang. Dywa: Dynamics-adaptive world action model for generalizable non-prehensile manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11058–11068, 2025.

Jiangran Lyu, Kai Liu, Xuheng Zhang, Haoran Liao, Yusen Feng, Wenxuan Zhu, Tingrui Shen, Jiayi Chen, Jiazhao Zhang, Yifei Dong, et al. Lda-1b: Scaling latent dynamics action model via universal embodied data ingestion. *arXiv preprint arXiv:2602.12215*, 2026.

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, et al. curobo: Parallelized collision-free minimum-jerk robot motion generation. *arXiv preprint arXiv:2310.17274*, 2023.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

RDT Team. Rdt2: Enabling zero-shot cross-embodiment generalization by scaling up umi data, September 2025. URL https://github.com/thu-ml/RDT2.

Yi-Lin Wei, Jian-Jian Jiang, Chengyi Xing, Xian-Tuo Tan, Xiao-Ming Wu, Hao Li, Mark Cutkosky, and Wei-Shi Zheng. Grasp as you say: Language-guided dexterous grasp generation. *Advances in Neural Information Processing Systems*, 37:46881–46907, 2024.

Yi-Lin Wei, Haoran Liao, Yuhao Lin, Pengyue Wang, Zhizhao Liang, Guiliang Liu, and Wei-Shi Zheng. Cyclemanip: Enabling cyclic task manipulation via effective historical perception and understanding. *arXiv preprint arXiv:2512.01022*, 2025a.

Yi-Lin Wei, Zhexi Luo, Yuhao Lin, Mu Lin, Zhizhao Liang, Shuoyu Chen, and Wei-Shi Zheng. Omnidexgrasp: Generalizable dexterous grasping via foundation model and force feedback. *arXiv preprint arXiv:2510.23119*, 2025b.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6d pose estimation and tracking of novel objects. In *CVPR*, 2024.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020.

Xuan Xiao, Jiahang Liu, Zhipeng Wang, Yanmin Zhou, Yong Qi, Shuo Jiang, Bin He, and Qian Cheng. Robot learning in the era of foundation models: A survey. *Neurocomputing*, 638:129963, 2025.

Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

Zhengrong Xue, Shuying Deng, Zhenyang Chen, Yixuan Wang, Zhecheng Yuan, and Huazhe Xu. Demogen: Synthetic demonstration generation for data-efficient visuomotor policy learning. *arXiv preprint arXiv:2502.16932*, 2025.

Sizhe Yang, Wenye Yu, Jia Zeng, Jun Lv, Kerui Ren, Cewu Lu, Dahua Lin, and Jiangmiao Pang. Novel demonstration generation with gaussian splatting enables robust one-shot manipulation. *arXiv preprint arXiv:2504.13175*, 2025.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Justin Yu, Letian Fu, Huang Huang, Karim El-Refai, Rares Andrei Ambrus, Richard Cheng, Muhammad Zubair Irshad, and Ken Goldberg. Real2render2real: Scaling robot data without dynamics simulation or robot hardware. *arXiv preprint arXiv:2505.09601*, 2025.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

Yifan Zhang, Ruiping Wang, and Xilin Chen. Dynamic behavior cloning with temporal feature prediction: Enhancing robotic arm manipulation in moving object tasks. *IEEE Robotics and Automation Letters*, 2025a.

Yuanhang Zhang, Tianhai Liang, Zhenyang Chen, Yanjie Ze, and Huazhe Xu. Catch it! learning to catch in flight with mobile dexterous hands. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14385–14391. IEEE, 2025b.

Qianyou Zhao, Duidi Wu, Yuliang Shen, Junlai Li, Qiaojun Yu, Shuo Zhang, Jin Qi, and Jie Hu. Hail: Human-in-the-loop assembly imitation learning system. *Journal of Manufacturing Systems*, 86: 991–1006, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. *arXiv preprint arXiv:2504.02792*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

APPENDIX

A NETWORK ARCHITECTURE & TRAINING DETAILS

A.1 NETWORK ARCHITECTURE

Our model adopts a diffusion-based framework that incorporates dynamic-aware perception and action generation capabilities. The architecture is designed to process multi-modal observations and generate precise robot actions through a denoising diffusion process.

A.1.1 INPUT MODALITIES.

The model takes two types of inputs at each timestep:

`Point clouds`: Single-view point clouds serve as high-dimensional visual observations, denoted as $\mathbf{o}_t^h \in \mathbb{R}^{N\times 3}$, where $N = 2048$ points are obtained via Farthest Point Sampling (FPS). We define the observation horizon as t_o and in our experiments, we set $t_o = 6$.

`Proprioception`: Low-dimensional robot state observations $\mathbf{o}_t^l \in \mathbb{R}^M$ capture joint positions and gripper configurations across a temporal window. The proprioception shares the observation horizon as t_o.

Point Cloud Encoder. We first crop the point cloud using a 3D bounding box and downsample to $N = 2048$ points via FPS. We employ a PointNet (Charles et al., 2017) encoder to independently encode each of the K_{high} sampled frames, then concatenate the features along the channel dimension to obtain the final visual representation $\mathbf{f}^h$.

State Encoder. Proprioceptive observations are encoded *per frame* with a lightweight MLP. For each observation step, the robot state vector is projected into a state feature token via the MLP. The state feature is then concatenated with the point-cloud feature at the same timestep.

A.1.2 Loss Function

We employ FiLM (Perez et al., 2018) conditioning to inject the conditional feature f_c into the denoising network, which predicts the noise associated with the robot action $\mathbf{a} \in \mathbb{R}^{k_a}$, where k_a denotes the action dimension specific to the task. The primary training objective minimizes the denoising score matching loss:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\mathbf{a}_0 \sim p_{\text{data}},\, t,\, \boldsymbol{\epsilon} \sim \mathcal{N}(0,\mathbf{I})} \left[\left\| \boldsymbol{\epsilon} - \epsilon_\theta(\mathbf{a}_t, t) \right\|^2 \right], \tag{4}$$

where $\mathbf{a}_0$ is sampled from the demonstration data distribution p_{data}, t is a sampled diffusion timestep, and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ is Gaussian noise. The network ϵ_θ predicts the added noise given the noisy action $\mathbf{a}_t$ and timestep t. We employ DDIM (Song et al., 2020) for inference sampling.

Dynamic-Stage Awareness Learning Every augmented trajectory carries a stage label $\ell_t \in \{\Phi_{\text{static}}, \Phi_{\text{dyn}}, \Phi_{\text{inter}}, \Phi_{\text{trans}}\}$ at each timestep t, reflecting the interaction logic that produced it. We leverage these labels as free supervision signals by attaching a lightweight classification head h_ϕ to the shared observation encoder. Given the fused feature f_{lh} processed through a dense fusion network ϕ_{dense}, the head predicts a stage distribution $\hat{\mathbf{p}}_t = h_\phi(\phi_{\text{dense}}(f_{lh}))$ over the four stages, supervised by a cross-entropy loss:

$$\mathcal{L}_{\text{stage}} = -\frac{1}{T} \sum_{t=1}^{T} \log \hat{p}_t(\ell_t). \tag{5}$$

The dense fusion network ϕ_{dense} enhances feature representations that are subsequently used by the diffusion decoder, while h_ϕ is intentionally kept as a lightweight single-layer MLP to mitigate the risk of overfitting the auxiliary head. The total training objective combines both losses:

$$\mathcal{L} = \mathcal{L}_{\text{diff}} + \lambda \mathcal{L}_{\text{stage}}, \tag{6}$$

where $\lambda = 0.1$ balances action fidelity and stage awareness. This auxiliary objective encourages the encoder to learn representations that disambiguate interaction phases, which are later exploited at inference time with negligible additional computation.

A.2 Training Configuration

We train all models using the AdamW optimizer with an initial learning rate of 2.0×10^{-4}, betas $[0.95, 0.999]$, weight decay 1×10^{-6}, and batch size 128. Training proceeds for 300 epochs with a cosine learning rate schedule and 500 warmup steps. We employ Exponential Moving Average (EMA) with momentum 0.9999 on model parameters, activated after the first training step.

The noise scheduler follows a squared cosine schedule with 100 diffusion timesteps during training, with $\beta_{\text{start}} = 0.0001$ and $\beta_{\text{end}} = 0.02$. For inference, we use DDIM sampling with 10 denoising steps to efficiently generate action sequences.

Data augmentation is minimal to preserve the precise geometric relationships required for manipulation: we only apply random translation within $\pm$2cm in the horizontal plane during point cloud preprocessing. All models are trained on a single NVIDIA RTX 4090 GPU.

A.3 Inference

A.3.1 Inference Sampling.

We employ DDIM for efficient inference sampling. The reverse diffusion process is formulated as:

$$\mathbf{a}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \left(\frac{\mathbf{a}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{a}_t, t)}{\sqrt{\bar{\alpha}_t}} \right) + \sqrt{1-\bar{\alpha}_{t-1}}\epsilon_\theta(\mathbf{a}_t, t), \tag{7}$$

where $\bar{\alpha}_{t-1}$ and $\bar{\alpha}_t$ are the cumulative noise schedule coefficients at timesteps $t-1$ and t, respectively. We use 10 denoising steps during inference for efficient real-time deployment.

A.3.2 TAYLORSEER ACCELERATION

During inference, the diffusion model must solve the reverse denoising process iteratively, which requires multiple forward passes through the 1D U-Net. Each DDIM step involves computing activations for all residual blocks across three resolution levels, making inference computationally expensive and limiting the real-time control frequency of the policy. To address this bottleneck, we introduce a Taylor-based acceleration mechanism that replaces selective U-Net forward passes with Taylor series extrapolation during the denoising trajectory.

Principle. The core observation is that adjacent DDIM steps produce highly correlated intermediate features — the U-Net's residual block outputs vary smoothly as the noise level changes incrementally along the reverse trajectory. Let $f^{(l)}(t_i)$ denote the output of residual block l at denoising step t_i. We maintain a cache of Taylor coefficients computed via backward finite differences over consecutive steps:

$$c_0 = f^{(l)}(t_i), \quad c_1 = \frac{f^{(l)}(t_i) - f^{(l)}(t_{i-1})}{t_i - t_{i-1}}, \quad c_2 = \frac{c_1^{(i)} - c_1^{(i-1)}}{t_i - t_{i-1}}, \tag{8}$$

and so on up to the maximum order K. For a future step $t_{i+\delta}$ where we skip the full forward pass, the Taylor prediction is:

$$\hat{f}^{(l)}(t_{i+\delta}) = \sum_{j=0}^{K} \frac{c_j}{j!} (t_{i+\delta} - t_i)^j. \tag{9}$$

Effect. TaylorSeer reduces inference latency by avoiding redundant full U-Net evaluations at selected DDIM steps. The latency values reported in the main paper are measured as end-to-end policy-query latency per control cycle, from the arrival of an observation to the output of the action chunk. This measurement includes observation encoding, DDIM sampling, stage-aware scheduling, and action selection.

For clarity, we use module-level profiling only to characterize the denoising computation itself. On a single NVIDIA RTX 4090 GPU, replacing a selected full U-Net computation with Taylor extrapolation reduces the cost of that denoising computation from approximately 46 ms to 22 ms, corresponding to about a $2.09\times$ speedup for the approximated computation. The actual end-to-end latency depends on the stage-specific schedule, the number of full recomputation steps, and non-denoising overheads. We therefore report the measured end-to-end latency in Table 3 of the main paper rather than deriving it by analytically multiplying the per-step profiling numbers. This acceleration is especially beneficial for real-time robot control, where the policy must repeatedly sample action trajectories under a strict control-frequency budget. Since Taylor operates only on intermediate U-Net features during inference, it does not modify the diffusion training objective, the DDIM update rule, or the policy output representation, making it a plug-and-play acceleration module for the deployed diffusion policy.

Computation Scheduling We do not apply Taylor approximation at every step. Instead, we adopt a periodic recomputation strategy that interleaves full forward passes with Taylor predictions. Specifically:

- **First step:** always computed fully to establish the initial anchor.
- **First k steps (default $k = 3$):** always computed fully, as the early denoising steps contain the most informative signal about the target action distribution.
- **Every τ steps (default $\tau = 5$):** computed fully to refresh the Taylor coefficients and prevent extrapolation drift.
- **All other steps:** replaced by Taylor prediction of order K (default $K = 1$, i.e., linear extrapolation).

With $\tau = 5$, $K = 1$, and $k = 3$ under $T = 10$ DDIM inference steps, 6 out of 10 steps use Taylor prediction while only 4 steps require full U-Net evaluation, reducing the number of full forward passes by approximately 60%.

Per-Stage Adaptive Scheduling Different interaction stages have distinct requirements for inference accuracy and re-planning frequency. We define four task stages corresponding to the data augmentation phases: Φ_{static} (static object acquisition), Φ_{dyn} (dynamic target alignment), Φ_{inter} (contact

interaction execution), and Φ_{trans} (pose-conditioned transition). For each stage, the predicted stage classifier output $\hat{\ell}_t$ determines three scheduling parameters: the TaylorSeer computation interval τ, the Taylor expansion order K, and the number of executed action steps n_{exec} from the predicted action chunk. The scheduling parameters are listed in Table 5.

Table 5: Stage-specific heuristic TaylorSeer and execution step scheduling. τ is the recomputation interval, K is the Taylor polynomial order, and n_{exec} is the number of action steps executed per inference call.

Stage	$\boldsymbol{\tau}$ (interval)	$\boldsymbol{K}$ (order)	$\boldsymbol{n}_{\text{exec}}$
Φ_{static}	3	1	2
Φ_{dyn}	7	1	2
Φ_{inter}	3	1	8
Φ_{trans}	8	2	8

A.3.3 Execution Step Configuration

The policy operates in a receding horizon control loop. At each inference call, the model predicts an action chunk of length $H = 8$ (the *action horizon*) via DDIM sampling with $T = 10$ denoising steps. Only a subset of n_{exec} actions from this chunk are actually executed before the next inference call is triggered. The remaining actions are discarded, and a fresh prediction is computed from the updated observations.

In our framework, n_{exec} is adjusted per Table 5. The static acquisition stage uses frequent recomputation and short execution chunks to avoid early grasp errors. The dynamic alignment stage keeps short chunks for frequent feedback while using a larger Taylor interval to reduce latency. The interaction and transition stages execute longer chunks because they benefit more from motion continuity and are less dependent on immediate visual correction.

B Efficient Data Augmentation Pipeline

In this section, we provide a detailed description of the data augmentation pipeline used in our framework. The process begins with a single **static** demonstration, from which we systematically synthesize massive and diverse augmented episodes featuring complex dynamic target movements. To seamlessly scale up the training dataset, our augmentation engine is parallelized across a cluster of four NVIDIA RTX 4090 GPUs. This fully automated pipeline encompasses several critical computational steps—including mesh generation, precise mesh alignment, dynamic trajectory editing, and point cloud post-processing—which we describe in detail below.

B.1 Task-Specific Setup Cost

Task-specific data preparation requires only limited human intervention. The planner, reachability constraints, and motion primitives are reused across tasks, while only a small number of task-specific components need to be configured.

B.2 Mesh Generation

To ensure the precise geometric reconstruction of the manipulation scene, we capture an RGB image of the manipulation object and generate a 3D mesh using a 3D generative model (Hyper3D, 2024). This generated mesh is then utilized in the subsequent mesh alignment step, where it is integrated to form a complete 3D representation of the object. This step is essential for accurate trajectory editing and the generation of realistic augmented episodes.

B.3 Mesh Alignment and Geometric Reconstruction

A crucial part of our data augmentation pipeline is the alignment of known CAD meshes to the observed point cloud. Since point clouds captured from a single viewpoint typically only contain visible

Table 6: Human operators and elapsed human/machine time for data preparation. Our pipeline is separated into static teleoperation, pre-generation processing, and automated trajectory generation; Table 7 provides the complete processing breakdown.

Data Preparation Setting	Human Operators	Human Time	Machine Time (RTX 4090)
Manual teleop. (50 dyn. demos)	2	$\sim$ 100 min	–
Manual teleop. (200 dyn. demos)	2	$\sim$ 400 min	–
Ours: teleop. (1 static demo)	1	$\sim$ 30 s	–
Ours: pre-generation processing	1	$\sim$ 10 min	$\sim$ 1 min
Ours: automated aug. (50 eps)	0	–	$\sim$ 7 min
Ours: automated aug. (200 eps)	0	–	$\sim$ 30 min
Ours: total (200 eps)	**1**	$\sim$ **11** min	$\sim$ **31** min

Table 7: Task-specific setup and data-generation time. All times are approximate and reported in minutes.

Step	Human	Machine
One static teleoperated demonstration	1	–
Keyframe annotation	2	1
Mesh-size correction	2	–
Phase composition	1	–
Motion-distribution specification	5	–
Automated generation of 200 trajectories	–	30
Total	**11**	**31**

Note. When keyframes are annotated online during demonstration collection, the total cost is reduced to approximately 9 minutes of human effort and 30 minutes of machine computation. The planner, reachability constraints, and motion primitives are reused across tasks.

surfaces, this step helps in reconstructing complete 3D assets, which are then used in subsequent stages.

B.3.1 Coarse-to-Fine Mesh Alignment

The mesh alignment is performed using a coarse-to-fine approach, where an initial alignment is followed by a refinement step to improve accuracy. The steps are as follows:

- **Coarse Alignment (OBB-based Initialization):** We begin by aligning the mesh to the observed point cloud using an Oriented Bounding Box (OBB). The OBB is computed to estimate the initial transformation between the mesh and the point cloud.
- **Fine Alignment (Iterative Closest Point - ICP):** After the coarse alignment, we apply a multi-resolution ICP algorithm to iteratively refine the alignment. ICP minimizes the distance between the points in the point cloud and the vertices of the mesh. This is achieved by solving for the transformation that best matches the observed points to the mesh at different resolutions.

The mathematical formulation of the ICP step is as follows:

$$\min_{\mathbf{T}} \sum_{i=1}^{N} \|\mathbf{p_i} - \mathbf{T}\mathbf{m_i}\|^2$$

where $\mathbf{p_i}$ is the point from the observed point cloud, $\mathbf{m_i}$ is the corresponding mesh vertex, and $\mathbf{T}$ is the transformation matrix that minimizes the point-to-point distance.

The visualization of the mesh alignment process is shown in Figure 5.

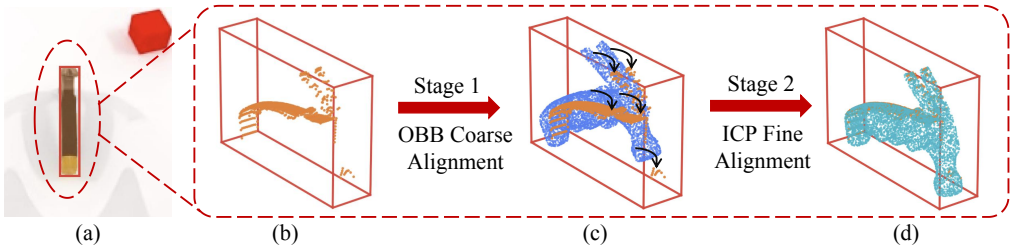

Figure 5: Visualization of the mesh alignment process. (a) shows the ego view image of a manipulation object, (b) shows the initial point cloud captured from the RGB-D sensor, (c) shows the coarse alignment using OBB, and (d) shows the refined alignment after applying ICP. The aligned mesh is then used for geometric reconstruction in the augmentation pipeline.

B.3.2 Canonical Point Cloud Generation

Once the mesh is aligned, the aligned mesh is resampled into a canonical point cloud that is spatially invariant. This canonical point cloud is used in subsequent steps and allows for geometrically coherent transformations during editing.

B.4 Dynamic Trajectory Editing

The core of our data augmentation pipeline lies in the *Dynamic Trajectory Editing* engine, which systematically decomposes a single static demonstration into four distinct physical phases: Static Object Acquisition (Φ_{static}), Dynamic Target Alignment (Φ_{dyn}), Contact Interaction Execution (Φ_{inter}), and Pose-Conditioned Transition (Φ_{trans}). Rather than relying on simple linear interpolations, our engine dynamically synthesizes and warps trajectories by coupling geometric registration, physical priors, and real-time motion planning. This ensures that the generated trajectories are not only physically feasible but also exhibit robust spatial generalization.

B.4.1 Static Object Acquisition (Φ_{static})

In tasks involving auxiliary tools (e.g., grasping a hammer or picking up a basket), the initial segment of the demonstration involves grasping or acquiring a static object. Because the object remains stationary during this phase, we parameterize this segment by annotating its start and end keyframes, denoted as $t_{\text{start}}^{\text{static}}$ and $t_{\text{end}}^{\text{static}}$, respectively.

To generalize this acquisition behavior to arbitrary initial poses in the augmented environment, we apply a rigid transformation $T_k^{\text{static}} \in \mathbb{SE}(3)$ to the recorded trajectory. The modified end-effector pose $\mathbf{e}'_t$ and object trajectory $\mathbf{p}'_t$ at any time step $t \in [t_{\text{start}}^{\text{static}}, t_{\text{end}}^{\text{static}}]$ are generated by replaying the source trajectory under this transformation:

$$\mathbf{p}'_t = T_k^{\text{static}} \mathbf{p}_t, \quad \mathbf{e}'_t = T_k^{\text{static}} \mathbf{e}_t \tag{10}$$

This rigid formulation effortlessly ensures spatial generalization, allowing the policy to reliably acquire tools or objects under diverse spatial displacements without requiring additional manual demonstrations.

B.4.2 Dynamic Target Alignment (Φ_{dyn})

This phase aligns the end-effector with a moving object or a predicted interaction target before contact. Instead of collecting dynamic teleoperation for this alignment behavior, we compute a time-varying target from the synthesized object motion and a task-specific prediction rule. Formally, given the synthesized object trajectory $T_{\text{obj}}^k(t)$ and augmentation parameters ξ_k, the interaction target is generated as

$$\mathbf{g}^k(t) = \Gamma_{\text{task}}\big(T_{\text{obj}}^k(0:t), \xi_k\big)\,, \tag{11}$$

where Γ_{task} denotes the task-specific target-generation rule. For object-relative interactions, $\mathbf{g}^k(t)$ can be an end-effector pose computed from a relative transform extracted from a contact keyframe:

$$\mathbf{g}^k(t) = T^k_{\text{obj}}(t) T_{\text{offset}} T^*_{\text{rel}}. \tag{12}$$

For interception or catching tasks, $\mathbf{g}^k(t)$ can instead be a predicted interception or landing target computed from the object's motion model, such as a parabolic trajectory estimate.

To reach this dynamic target, a reactive motion planner generates smooth trajectories under a maximum velocity constraint $v_{\max}$. The planner replans at a high frequency to correct tracking deviations and projects the resulting trajectory into the robot's reachable workspace, cropping segments that violate kinematic limits. The alignment phase terminates once the end-effector is sufficiently close to the generated target under a task-defined distance metric $d(\cdot, \cdot)$:

$$d\big(\mathbf{e}(t), \mathbf{g}^k(t)\big) < \epsilon. \tag{13}$$

This formulation matches the main paper's generalized target-conditioned alignment phase while still retaining fixed object-frame relative poses as a special case.

B.4.3 Contact Interaction Execution (Φ_{inter})

The contact phase captures the precise, high-frequency interaction behaviors between the robot and the object, such as striking a target or executing a dynamic tap. Since these local contact patterns are highly sensitive to physical constraints and difficult to plan from scratch, we preserve the local kinematic interaction pattern.

This phase is bounded by annotated start and end keyframes $[t^{\text{inter}}_{\text{start}}, t^{\text{inter}}_{\text{end}}]$. We model the local trajectory relative to a source anchor frame A, which represents the contact state in the demonstration. During augmentation, we warp this segment to a new target anchor A_k (determined by the terminal pose of the preceding Φ_{dyn} phase), translating and rotating the entire interaction sequence:

$$\mathbf{e}'_t = A_k A^{-1} \mathbf{e}_t, \quad \forall t \in [t^{\text{inter}}_{\text{start}}, t^{\text{inter}}_{\text{end}}] \tag{14}$$

By shifting the anchor frame, we replay the interaction segment at different object positions and orientations while preserving the local motion pattern.

B.4.4 Pose-Conditioned Transition (Φ_{trans})

After completing an interaction, the robot must safely transition to a target spatial configuration, such as returning to a neutral home position or moving to an intermediate waypoint (e.g., returning to the center in a whack-a-mole task).

Rather than replaying a rigid, collision-prone path, we leverage our motion planner to generate collision-free trajectories to a transformed keyframe pose. This phase requires only a single keyframe annotation to identify the target pose $\mathbf{e}_{\text{key}}$. To introduce spatial variation and prevent policy overfitting, we apply a stochastic pose offset $T^{\text{trans}}_k \in \mathbb{SE}(3)$ to the target configuration:

$$\mathbf{e}^k_{\text{key}} = T^{\text{trans}}_k \mathbf{e}_{\text{key}} \tag{15}$$

The motion planner then computes a smooth trajectory from the robot's current state to $\mathbf{e}^k_{\text{key}}$. The integration of the motion planner ensures that the robot can robustly transition across diverse configurations while satisfying collision and joint-limit constraints.

B.5 Task-Specific Phase Composition Logic

DynamicManip composes the four phase operators $\{\Phi_{\text{static}}, \Phi_{\text{dyn}}, \Phi_{\text{inter}}, \Phi_{\text{trans}}\}$ into task-specific trajectories. Motion planning provides geometrically feasible transitions under robot constraints, while anchor-based replay preserves local interaction patterns from the source demonstration. This section describes the phase compositions and tracking logic used in simulation and real-world tasks.

B.5.1 Simulation Benchmark Tasks

We compose the phase operators to generate repeated interactions and interception trajectories with varying target motions.

- **Dynamic Tapping:** We generate repeated striking motions using a cyclic structure: $\Xi_{\text{tap}} = \Phi_{\text{static}} \rightarrow (\Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}})^L \rightarrow \Phi_{\text{trans}}$. The trajectory begins with tool acquisition (Φ_{static}), followed by repeated target alignment (Φ_{dyn}) and replay of the striking motion (Φ_{inter}). The final Φ_{trans} returns the robot to the reset configuration.
- **Mole Whacking:** Because the target appears at different locations, we insert a reset after each interaction: $\Xi_{\text{mole}} = \Phi_{\text{static}} \rightarrow (\Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}} \rightarrow \Phi_{\text{trans}})^L$. The Φ_{trans} step returns the robot to the center before the next interaction.
- **Object Catching:** We compose container acquisition and dynamic target alignment: $\Xi_{\text{catch}} = \Phi_{\text{static}} \rightarrow \Phi_{\text{dyn}}$. The robot grasps the container and tracks the predicted landing position of the incoming object.
- **Goal Blocking:** The robot pre-positions the paddle and performs a rapid, reactive interception: $\Xi_{\text{block}} = \Phi_{\text{static}} \rightarrow \Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}}$.
- **Belt Picking:** We separate transport and interaction into the following phases: $\Xi_{\text{belt}} = \Phi_{\text{trans},1} \rightarrow \Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter},1} \rightarrow \Phi_{\text{trans},2} \rightarrow \Phi_{\text{inter},2}$. The first transition $\Phi_{\text{trans},1}$ moves the robot to a pre-grasp pose (e.g., orienting the end-effector vertically downward) for target tracking. After reactive alignment (Φ_{dyn}) and grasping ($\Phi_{\text{inter},1}$), $\Phi_{\text{trans},2}$ explicitly handles collision-free transport to the target bin, culminating in a precise release ($\Phi_{\text{inter},2}$).

B.5.2 Real-World Benchmark Tasks

For real-world tasks, we adapt the phase compositions to the robot's reachable workspace and use finite interaction cycles and terminal resets.

- **Boat Loading (Reachability-Aware Tracking):** The boat can move outside the robot's reachable workspace. We use an augmented tracking phase (Φ^*_{dyn}) to include these boundary cases in the generated demonstrations: $\Xi_{\text{boat}} = \Phi_{\text{static}} \rightarrow \Phi^*_{\text{dyn}} \rightarrow \Phi_{\text{inter}} \rightarrow \Phi_{\text{trans}}$. Even when the boat temporarily moves out of the end-effector's reachable zone (e.g., points A or B), Φ^*_{dyn} actively projects the target to the nearest feasible boundary points (e.g., A^* or B^*). This projection keeps the generated tracking trajectory within the reachable workspace. Once the boat returns to a reachable placement position, the trajectory proceeds to placement (Φ_{inter}).
- **Real-World Dynamic Tapping:** We limit each real-world trial to three strikes ($L = 3$): $\Xi_{\text{tap-real}} = \Phi_{\text{static}} \rightarrow (\Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}})^3 \rightarrow \Phi_{\text{trans}}$.
- **Bottle Catching:** We use a tracking-to-grasp sequence to intercept the rolling bottle before it falls: $\Xi_{\text{bottle}} = \Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter}}$.
- **Real-World Belt Picking:** We append a terminal reset to return the robot to its initial configuration between trials: $\Xi_{\text{belt-real}} = \Phi_{\text{trans},1} \rightarrow \Phi_{\text{dyn}} \rightarrow \Phi_{\text{inter},1} \rightarrow \Phi_{\text{trans},2} \rightarrow \Phi_{\text{inter},2} \rightarrow \Phi_{\text{trans},3}$. The final $\Phi_{\text{trans},3}$ returns the arm to the reset configuration.

B.6 Point Cloud Post-Processing

Once the trajectory is edited and the augmented episode is generated, we perform post-processing on the point cloud. The primary goal is to merge point clouds from various stages, ensuring that the final scene representation is cohesive and complete.

B.6.1 Point Cloud Merging and Cleanup

Point clouds from different stages (e.g., Grasp, Follow, Interact, Recover) are merged to form the final scene. We ensure that redundant points are removed, and the points are uniformly distributed in the final representation.

We use a k-d tree for efficient nearest-neighbor search during the cleanup process. For each point in the scene, we check if it is too close to an object or part of the robotic arm and remove it if necessary. The cleaning is based on a threshold distance δ, which is set to 2 cm experientially.

B.6.2 Final Point Cloud Formatting

After cleanup, the point cloud is formatted to comply with the HDF5 data structure, where each augmented episode is stored with its associated metadata (e.g., end-effector poses, joint actions, etc.). The final point cloud is stored as a 2048-point sample for each frame, ensuring consistency across the augmented dataset.

$$\text{Final Point Cloud} = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_{2048}}\}$$

where each point $\mathbf{p_i} = [x_i, y_i, z_i, r_i, g_i, b_i]$ represents the 3D coordinates and color of a point in the scene (in this case, color channels are unused).

Once the point clouds have been processed, the final augmented episodes are saved in HDF5 format. This format allows for efficient storage and retrieval of large datasets, and each augmented episode contains all the relevant data for training robotic systems.

- **Episode Data:** For each episode, we store the point cloud, end-effector pose, joint action, and episode boundaries.
- **Metadata:** Additional metadata includes task-specific parameters and transformation matrices used during augmentation.

C Benchmark Details

Overview

We construct a comprehensive benchmark for dynamic manipulation tasks, encompassing both simulation and real-world environments. The simulation benchmark covers five dynamic tasks with 200 expert demonstrations each, built on the RoboTwin 2.0 platform using the SAPIEN (Xiang et al., 2020) physics simulator. The real-world benchmark includes four dynamic tasks with 50 demonstrations each, collected using AgileX Piper robotic arms and an Intel RealSense L515 RGB-D camera.

C.1 Simulation Benchmark

Our simulation benchmark is built upon the RoboTwin 2.0 platform, using the SAPIEN physics simulator as backend.

C.1.1 Environment Setup

The simulation environment consists of several key components:

3D Assets

We use a diverse set of 3D assets for the manipulation objects, leveraging both existing object datasets from RoboTwin and custom-designed assets. For the customized assets, we create them using Hyper3D from a single RGB image and then manually adjust the scale of the generated meshes to ensure they are of appropriate size for the tasks. We then annotate each mesh with *contact points* and *function points* that are relevant for the specific manipulation tasks.

Automatic data collection system. The RoboTwin 2.0 platform provides a built-in data collection system that allows us to record expert demonstrations in a structured format. For each manipulation task, we design an environment script that sets up the scene with the relevant objects, initializes the robot in a predefined starting configuration, and describes the task-specific interaction logic. During trajectory planning, the system plans expert demonstration trajectories using CUROBO (Sundaralingam et al., 2023), which generates smooth and feasible trajectories that achieve the task objectives. During grasping, the system iteratively tries the annotated contact points and selects the one that results in a successful grasp.

In data collection, we record multi-modal observations including RGB-D images from the head-mounted camera with 2048 points sampled from the point cloud, proprioceptive states including joint positions, end-effector poses, and gripper states, and language instructions describing the task from

Table 8: Sim data confidence intervals

Setting		Dynamic Tapping	Belt Picking	Object Catching	Mole Whacking	Goal Blocking
1 demo		[19%, 36%]	[0%, 4%]	[4%, 15%]	[1%, 8%]	[7%, 20%]
10 demos		[9%, 23%]	[2%, 11%]	[44%, 63%]	[14%, 30%]	[37%, 56%]
50 demos		[13%, 28%]	[23%, 41%]	[28%, 47%]	[53%, 72%]	[88%, 97%]
200 demos		[45%, 64%]	[77%, 91%]	[40%, 60%]	[74%, 89%]	[79%, 92%]
1 + aug	(50)	[36%, 55%]	[35%, 54%]	[60%, 78%]	[53%, 72%]	[80%, 93%]
	(200)	[42%, 62%]	[73%, 88%]	[54%, 73%]	[71%, 87%]	[80%, 93%]

the task-specific templates. All data is recorded at 30 Hz control and observation frequency and saved in HDF5 format for efficient storage and retrieval.

C.1.2 Task descriptions.

Dynamic Tapping. The robot needs to continuously track a moving object and execute a timely tapping motion to contact the target while it is in motion.

Belt Picking. The robot needs to track an object moving on a conveyor belt, grasp it within the reachable workspace, and place it into a basket.

Object Catching. The robot needs to grasp a basket and estimate the trajectory of a thrown object and move a basket to the predicted landing position to catch it.

Mole Whacking. The robot needs to interact with a 3×3 whack-a-mole setup, where a mole appears from one hole and moves randomly toward another while leaving a visible motion trace. The robot must anticipate the mole's future position and strike it at the appropriate time.

Goal Blocking. The robot needs to block an incoming ball from entering the goal by moving a handheld stick or paddle to intercept the ball's trajectory.

C.1.3 Evaluation Metrics.

We develop task-specific automatic evaluation criteria for each scenario to ensure objective and reproducible performance measurement. For Dynamic Tapping, success is determined by detecting physical contact between the robot and the moving target. For Belt Picking, we evaluate whether the object is successfully transported and placed within a predefined target region. For Object Catching, we measure whether the object is correctly captured inside the basket based on spatial inclusion. For Mole Whacking, success is defined by whether the robot strikes the correct target location corresponding to the mole's appearance or predicted trajectory. For Goal Blocking, we verify whether the ball is successfully intercepted before entering the goal region.

All evaluations are conducted within the RoboTwin simulation environment, allowing efficient, scalable, and fully automated benchmarking. In addition, we record the robot's motion trajectories during task execution for further analysis of motion smoothness and dynamic tracking performance.

C.1.4 Confidence Intervals for Simulation Evaluation

We report 95% confidence intervals to quantify the uncertainty of success-rate estimates from finite simulation evaluations. For each setting, we evaluate the policy over $n = 100$ trials and compute the empirical success rate $\hat{p} = x/n$, where x is the number of successful trials. We use the Wilson score interval:

$$\mathrm{CI}_{95\%} = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \quad z = 1.96.$$

As shown in Table 8, increasing the number of training episodes generally improves task success, though the effect varies across tasks. The augmented-data settings show that augmentation from a single expert demonstration can provide useful additional training data.

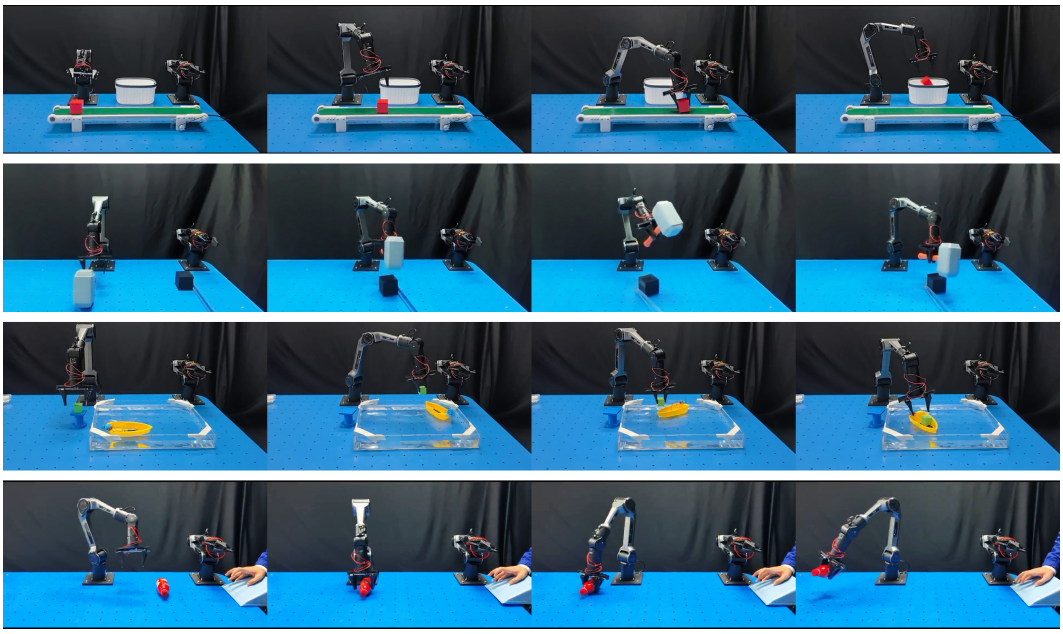

Figure 6: Visualization of autonomous policy inference and execution across four real-world dynamic manipulation tasks. From top to bottom: Belt Picking, Dynamic Tapping, Boat Loading, and Bottle Catching.



Figure 7: Visualization of real-world robot setup and the objects used in our experiments.

C.2 Real-world Benchmark

Our real-world benchmark is designed to validate the effectiveness of the proposed data augmentation strategy in practical robotic scenarios. To this end, we construct four dynamic manipulation tasks and evaluate policies trained with our augmented data against strong baselines. Figure 6 visualizes autonomous policy inference and execution across these tasks. Compared to simulation, real-world dynamic tasks introduce system latency and sensing noise, making precise timing and tracking more challenging. This places stricter requirements on the robustness and responsiveness of the learned policy, providing a more convincing evaluation of our approach.

C.2.1 Environment Setup

The real-world platform consists of two Piper robotic arms equipped with parallel grippers. We use a RealSense L515 depth camera for visual perception. Figure 7 shows the apparatus and experimental objects, which have diverse geometries and physical properties.

Table 9: Real-world data confidence intervals

Setting		Dynamic Tapping	Belt Picking	Boat Loading	Bottle Catching
1 demo		[3%, 14%]	[17%, 48%]	[0%, 11%]	[0%, 11%]
50 demos		[59%, 78%]	[52%, 83%]	[1%, 17%]	[56%, 86%]
1 + aug	(50)	[51%, 71%]	[39%, 73%]	[12%, 41%]	[59%, 88%]
	(200)	[79%, 93%]	[56%, 86%]	[49%, 81%]	[66%, 93%]

Data collection is performed using a master–slave teleoperation setup with two GELLO devices. All demonstrations are recorded in the same format as in simulation to ensure consistency between training and deployment. The dataset is stored in HDF5 format and collected at 20 Hz.

For each timestep, we record the robot joint positions and velocities, end-effector poses, and gripper states. In addition, we capture synchronized RGB-D observations, which are further processed into point clouds with 2048 sampled points. Camera intrinsic and extrinsic parameters are also stored to support geometric reasoning.

C.2.2 Task Descriptions

We evaluate four representative dynamic manipulation tasks in the real-world setting:

Dynamic Tapping. The robot tracks a moving object and executes a precisely timed tapping motion to make contact.

Belt Picking. The robot tracks an object moving on a conveyor belt, grasps it at the appropriate moment, and places it into a basket.

Boat Loading. The robot grasps an object and places it onto a motorized boat moving on a water surface, requiring coordination with the boat's motion.

Bottle Catching. The robot intercepts and grasps a rolling bottle before it falls off the edge of the table.

C.2.3 Evaluation Metrics

For each task, we perform 30 trials and report the average success rate. Task-specific automatic evaluation criteria are defined as follows: for Dynamic Tapping, success is determined by whether the robot successfully contacts the moving target; for Belt Picking, whether the object is grasped and placed into the basket; for Boat Loading, whether the object is successfully placed onto the moving boat; and for Bottle Catching, whether the bottle is successfully intercepted and grasped before falling.

These metrics enable consistent and objective evaluation across all tasks.

C.2.4 Confidence Intervals for Real-world Evaluation

Table 9 reports 95% Wilson confidence intervals using 90 independent strike events across 30 trials for Dynamic Tapping and 30 trials for each of the other tasks. We compute the success rate as $\hat{p} = x/n$, where x is the number of successful strikes or trials, respectively.

D Additional Ablations

D.1 Cumulative Data-Generation Ablation

Each row of Table 10 cumulatively adds one component of the augmentation pipeline. All variants share the demo, reconstruction, keyframes, data budget, policy, training, and evaluation (success rate, %). Spatial transforms provide geometry variation but no dynamics; quasi-static editing diversifies trajectories; planning supplies global robot–object motion; and replay preserves local contact timing

Table 10: Cumulative data-generation ablation (success rate, %). Each row adds one component; all variants share the demo, reconstruction, keyframes, data budget, policy, training, and evaluation.

Cumulative setting	Object Catching	Goal Blocking	Mole Whacking
Spatial transforms	2	4	15
+ Quasi-static editing	26	34	29
+ Planned dynamic motion	53	75	47
Ours	68	87	80

Table 11: Stage-aware inference ablation in the latency-aware setting (success rate, %). All variants use the same checkpoint and evaluation episodes; only the inference schedule differs.

Schedule	Object Catching	Goal Blocking
Full stage-aware	68	87
Uniformly reduced DDIM	19	38
Fixed Taylor acceleration	34	67
Fixed short chunks	48	64

and interaction details. The stepwise gains therefore support complementary, rather than redundant, roles.

D.2 Stage-Aware Inference Ablation

Using the same checkpoint and episodes, we changed only the inference scheduling. We evaluated the policies in a harder, more realistic setting where the simulation continues evolving during policy inference, which better assesses the model's real-time responsiveness in dynamic tasks. Uniform DDIM reduction harms action quality; fixed Taylor acceleration partly recovers it but cannot allocate precision by stage; fixed short chunks replan often, but inference latency makes observations and actions stale. The full method jointly adapts denoising, acceleration, and chunk length using predicted, rather than oracle, interaction stages.

E Out-of-Distribution and Robustness Analyses

E.1 Unseen Dynamics and Objects

Table 12 reports success rates under distribution shift in object motion and object instances. Performance on unseen objects remains comparatively robust, while unseen direction changes cause the largest degradation on Object Catching.

E.2 Sensitivity to Mesh, Calibration, and Stage-Prediction Errors

We varied one error source at a time; entries pair success rate (%) with the perturbation.

E.3 Keyframe Perturbation

The framework is not sensitive to key-frame selection within the tested range. We randomly shifted each keyframe by up to ± 10 frames and evaluated the resulting performance.

E.4 Point-Cloud Noise

The mesh-based renderer produces geometrically consistent point clouds and explicitly models real sensor noise by adding random noise and shifts in the pipeline. We injected noise points during deployment on the bottle-catching task.

Table 12: OOD evaluation (success rate, %). We vary object motion and object instances one at a time.

Setting	Dynamic Tapping	Goal Blocking	Object Catching
In-domain	52	87	68
Unseen direction changes	42	72	38
Out-of-range velocities	38	68	64
Unseen objects	58	80	56

Table 13: Sensitivity to a single error source at a time (success rate, %).

Source	Clean	Mild	Moderate	Severe
Mesh error (scale)	78.3	79.0 ($\pm$5)	75.7 ($\pm$10)	52.7 ($\pm$20)
Calibration error (shift)	78.3	78.0 (5 mm/1°)	73.7 (10 mm/3°)	47.7 (30 mm/10°)
Wrong-stage rate	78.3	78.7 (5%)	76.3 (10%)	66.0 (40%)

F Additional Real-World Evaluations

F.1 New Embodiment and Task

We conducted a new real-world pull-back-car task on a new AgileX Piper + BrainCo Revo2 dexterous hand embodiment. The same protocol achieved an 87% success rate.

F.2 Variation in Surface Conditions

We also evaluated the policy under different environment dynamics.

F.3 Transfer to Another Action-Chunk Policy

The proposed learning strategy is also policy-agnostic in the tested cases rather than specific to DP3 or individual tasks. It can be incorporated into another policy that predicts action chunks by adding phase prediction and phase-conditioned replanning. To verify this generality, we applied the same learning component—without task-specific redesign—to the VLA policy π_0 (Black et al., 2024).

π_0 predicts a 50-step action sequence from a single observation and therefore does not directly observe temporal motion cues across consecutive observations; our learning strategy triggers new replanning at critical moments, allowing it to better capture motion relationships.

G Comparison with a Tracking-and-Planning Pipeline

To probe whether a tracking-and-planning pipeline could replace the learned policy, we implemented a FoundationPose-based pipeline (Wen et al., 2024) and evaluated it on Belt Picking, Object Catching, and Goal Blocking.

Implementing the pipeline required more than three hours of human effort per task (one person, using a state-of-the-art coding agent). Failure reasons for the tracking-and-planning pipeline include tracking failures (60%) and task-related failures.

H Limitations

Despite the promising results demonstrated by DynamicManip, our framework has several limitations that point to directions for future work.

Deformable objects. Our data augmentation pipeline relies on mesh-enhanced geometric reconstruction that assumes rigid objects with stable geometries. For deformable or articulation-enabled objects such as cloth, ropes, or spring-loaded mechanisms, the ICP-based alignment and rigid transformation

Table 14: Change in success rate (%) under random keyframe shifts.

Task	Goal Blocking	Object Catching	Mole Whacking
Δ s.r. (%)	-5	$+2$	$+1$

Table 15: Bottle-catching success under injected point-cloud noise.

Setting	Std. dev. (cm)	Noise-point ratio	Success
No noise	–	0%	7/10
Small-range noise	(2.5, 5, 5)	5.96%–7.76%	6/10
Large-range noise	(5, 10, 10)	5.96%–7.76%	6/10

Table 16: Success under different surface conditions (successful trials).

Surface condition	Successful trials
Dry tabletop	9/10
Wet tabletop	8/10
Towel-covered tabletop	9/10

Table 17: Transfer to π_0 (success rate, %).

Policy	Goal Blocking	Object Catching
π_0	22%	28%
π_0 + ours	62%	40%

Table 18: Comparison with a FoundationPose-based tracking-and-planning pipeline (success rate, %).

Method	Belt Picking	Object Catching	Goal Blocking
Ours	82	68	87
FoundationPose + planner	50	32	40

assumptions break down. Editing the trajectory of a deformable object would require modeling its evolving shape and physical parameters, which remains an open challenge.

Rigid grasp assumption. The current pipeline attaches the grasped object to the end-effector via a fixed relative transform $\mathbf{T}_{\text{attach}}$ after contact. This assumption holds for tasks where the object does not slip or reorient relative to the gripper, but fails in scenarios requiring in-hand manipulation, rolling, or sliding contacts. Extending our method to handle non-rigid attachments or dynamic grasp maintenance would be a valuable extension.

Lack of dexterous tasks. Our benchmark tasks primarily involve parallel-jaw grippers and gross manipulation primitives such as tapping, catching, and blocking. We have not explored dexterous, multi-fingered tasks that require fine finger coordination or adaptive grasping strategies. Applying DynamicManip to dexterous manipulation would likely require augmenting the action space and rethinking the trajectory editing pipeline for multi-contact interactions.