

ScheduleGen: Demonstration Generation via Skill Scheduling for Multi-Arm Imitation Learning

Neel Jawale¹, Stephan Pleines¹, Soha Pouya¹, Yuke Zhu^{1,2}, Ajay Mandlekar^{1,†}, Caelan Garrett^{1,†}

¹NVIDIA ²The University of Texas at Austin [†]Equal advising

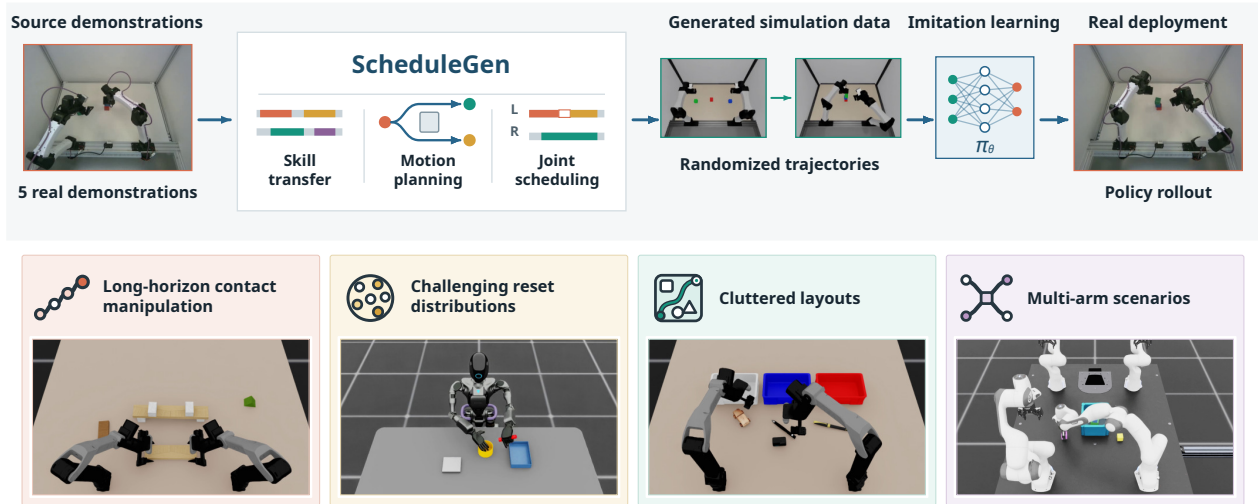


Figure 1: **ScheduleGen overview.** *Top:* From a few source demonstrations, ScheduleGen adapts contact-rich skill demonstrations, plans collision-free connecting motions, and jointly schedules multiple arms to generate simulation data for multi-arm imitation learning that transfers to the real world. *Bottom:* ScheduleGen can handle long-horizon contact manipulation, challenging reset distributions, cluttered layouts, and multi-arm scenarios.

Abstract

Multi-arm robot platforms, including bimanual and humanoid robots, have the capability to complete tasks with lower cycle times than single-arm systems, but they are more difficult to safely control due to the presence of multi-arm constraints. The paradigm of training generalist robot policies from massive datasets is a promising approach to address general-purpose multi-arm manipulation, but the acquisition of large multi-arm manipulation datasets remains a bottleneck. Automated demonstration generation can alleviate this burden by synthesizing new demonstrations from a handful of source demonstrations, but existing algorithms ignore multi-arm constraints and as a result often induce unsafe and unsuccessful behaviors. We address this limitation by introducing ScheduleGen, an integrated planning & scheduling algorithm for multi-arm demonstration generation. We evaluate ScheduleGen on 28 simulation task variants, including seven externally authored RoboDojo tasks and settings inaccessible to prior systems; it achieves 79.6% mean data-generation success and trains policies with 65.5% mean success. ScheduleGen policies outperform those trained on a state-of-the-art data generation baseline, and achieve 50-80% success across eight real-world variants through zero-shot sim-to-real and real-to-sim-to-real transfer. Videos and more at: [ScheduleGen Website](#)

Keywords: Imitation Learning, Demonstration Generation, Multi-Arm Manipulation, Planning and Scheduling

1. Introduction

Generalist robot policies [1, 24] hold the promise of seamlessly solving a wide range of manipulation tasks across diverse robot embodiments. There is substantial interest in applying this paradigm to multi-arm embodiments, especially bimanual and humanoid robots, due to their ability to address a wider range of tasks compared to single-arm embodiments. Such embodiments make it possible to address tasks requiring multi-arm coordination and larger manipulation workspaces, and also have the potential to carry out tasks more efficiently due to simultaneous operation.

However, the predominant method for training generalist robot policies requires large-scale data collection, usually obtained through months of human teleoperation effort [8, 16]. Collecting such datasets is already painful and costly for single-arm embodiments, but these problems are exacerbated in the multi-arm setting, due to the need for special teleoperation interfaces [5, 29, 32], and the resulting increase in operator burden. It can furthermore be difficult to capture desirable multi-arm coordination characteristics due to the limitations of the teleoperation interface [15].

Synthetic data generation in simulation is a compelling method to address this need. Prior methods allow for the generation of large-scale datasets with minimal human effort [6, 21] but fail to address some key challenges of generating data for multi-arm scenarios. Some works use planning-based approaches [4, 6, 22], but contact-rich tasks remain difficult for such methods, and generating data for diverse multi-arm coordination characteristics remains challenging. DexMimicGen [15] is able to handle different contact-rich tasks, but the coordination behaviors that it can generate are entirely dictated by the human demonstrations used to seed the process, limiting data diversity. Furthermore, the absence of planning means that it cannot handle situations where arm-arm or arm-object collisions are prevalent, such as wide object distributions or cluttered workspaces.

To address these shortcomings, we present ScheduleGen, a multi-arm manipulation data generation method that can encompass diverse multi-arm coordination strategies, long-horizon and contact-rich manipulation, and a wide range of workspaces and embodiments, including challenging object initializations and cluttered scenes. Given a small set of annotated demonstrations, ScheduleGen preserves object-relative contact skills, replans collision-free transitions, and jointly schedules the arms under task and collision constraints.

The contributions of this paper are:

- ScheduleGen, a new automated demonstration generation algorithm for multi-arm systems that integrates planning, scheduling, and demonstration adaptation.
- ScheduleGen utilizes a novel planning formulation that first plans single-robot behavior to adapt object-centric skill demonstrations and then safely coordinates the robots by framing scheduling as a Mixed Integer Linear Program (MILP) with constraints evaluated using GPU-accelerated collision checking.
- We demonstrate ScheduleGen’s applicability to long-horizon contact-rich tasks, challenging re-sets, cluttered workspaces, and configurable coordination strategies across bimanual, four-arm, and humanoid systems. We evaluate ScheduleGen on 28 task variants across 3 simulation frameworks. On comparable tasks, ScheduleGen-trained policies outperform prior work DexMimicGen [15]. In zero-shot sim-to-real and real-to-sim-to-real evaluation, the learned policies achieve 50–80% success across eight real-world variants.

2. Related Work

Imitation Learning from Human Demonstrations. Manipulation demonstrations are often collected via robot teleoperation [20, 31], and Behavioral Cloning (BC) [25] is a common method for training manipulation policies on such datasets. Recent efforts have scaled this paradigm to months of demonstration data using large teams of human teleoperators [8, 16], and train expressive policy architectures

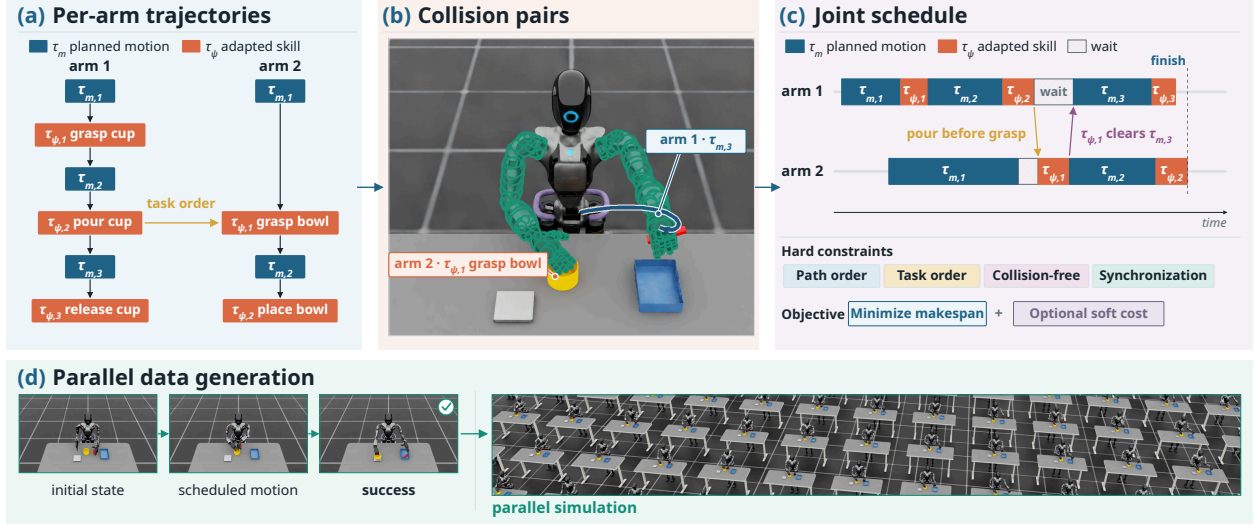


Figure 2: **ScheduleGen methodology.** (a) Contact-skill segments τ_{ψ} are adapted to the initial state and connected by planned motion τ_m , producing partially ordered paths for each arm. (b) Collision checking identifies conflicting path segments. (c) ScheduleGen implicitly inserts waits to form a joint schedule that preserves task order, avoids collisions, and enforces synchronization when required. It minimizes makespan with optional soft costs, like summed robot execution times. (d) Scheduled motions are replayed across parallel simulation environments, and successful rollouts are retained as demonstrations.

on this data, resulting in capable generalist robot policies [1, 24]. However, data sourcing remains a key bottleneck for this paradigm. The problem is exacerbated for bimanual robots and humanoids, which can require specialized teleoperation interfaces to support multi-arm teleoperation [5, 29, 32].

Automated Demonstration Generation. Synthetic data generation in simulation is a promising way to address the data bottleneck. TAMP-based approaches [4, 10, 22] fully automate generation but are limited to single-robot prehensile manipulation. MimicGen [21] and DexMimicGen [15] instead adapt a few teleoperated demonstrations to new initial states for single-arm and bimanual tasks, but adapting full demonstrations without accounting for collision or kinematic constraints limits their robustness to substantial initial-state variation. SkillGen combines demonstration adaptation with motion planning to preserve contact-rich behavior while accommodating environmental and state changes [11], but is limited to single-arm systems. GLIDE [17] and HumanoidMimicGen [18] also incorporate motion planning and support some bimanual tasks, but independently planned arm motions compromise safety and success on complex bimanual tasks. In contrast, ScheduleGen frames demonstration generation as a collision-aware scheduling problem to directly plan safe and efficient multi-arm manipulation.

Multi-Arm Manipulation. Planning for multiple arms as a single composite robot can be intractable due to high-dimensional, combinatorial search spaces. Multi-robot planners therefore factor the problem into interacting single-robot subproblems [7], reconciling pairwise constraints through backtracking over action timing [12]. Prior work includes domain-specific planners for assembly [2] and construction [13], as well as single-robot TAMP frameworks extended to simultaneous multi-robot manipulation [9, 28, 30]. ScheduleGen also uses factorization for efficient multi-arm planning, but it introduces demonstrations as schedulable skills and analyzes how scheduling decisions affect end-to-end visuomotor policy learning.

3. Problem Description

Our objective is to develop an automated planning & scheduling system that generates demonstration datasets for training multi-robot visuomotor policies using imitation learning. We start by formulating our multi-robot control problem (Section 3.1). Then, we define the notion of an object-centric single-

robot manipulation skill (Section 3.2). Finally, we introduce temporal constraints among skills, which model the minimal set of temporal dynamics that skills must satisfy to correctly satisfy the task (Section 3.3).

3.1. Prerequisites

We are interested in controlling a set of robot arms $\mathcal{R}$ simultaneously to complete a task in an environment with objects represented by coordinate frames $\mathcal{F}$. Let w denote the world frame and e_r denote robot r 's end-effector frame. The state of each robot r is represented by the *configuration* $q \in \mathbf{R}^d$ of its d degrees of freedom (DOFs). Let $T_{f_1}^{f_2} \in \text{SE}(3)$ denote the rigid transform from frame f_1 to frame f_2 ; thus, T_f^w is the *world pose* of frame f . Let the pose of robot r 's end-effector frame e_r be $T_{e_r}^w$.

We model multi-robot manipulation as a Partially Observable Markov Decision Process (POMDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, initial states $s_0 \sim p_0$, and goal states $S_* \subseteq \mathcal{S}$. Each state s encodes the configuration of each robot arm $s[r] = q$, pose of each end effector $s[e_r] = T_{e_r}^w$, and pose of each object frame $s[f] = T_f^w$. Actions specify arm and finger targets. Here, we use joint targets, but ScheduleGen also supports relative and absolute end-effector pose targets (Fig. 2). Actions are executed by a composite controller at a constant rate of $1/\Delta t$. Observations include external and wrist RGB camera images as well as centralized robot proprioception. We wish to learn policies $\pi : \mathcal{O} \rightarrow \mathcal{A}$ that map observations to actions.

3.2. Skill Adaptation

Our high-level demonstration generation strategy is to consume a set of object-centric skills per robot, adapt the skills given the current state of the objects, and schedule safe and correct execution of the skills among the robots. Unlike prior work [11, 18], we support different methods for acquiring skills including 1) segmenting teleoperated demonstrations, 2) scripting waypoints, 3) using pre-grasp and grasp waypoints, and 4) agentially authoring skills. A robot arm *skill* $\psi = \langle r, f, \vec{s}, \vec{a} \rangle$ is defined by a robot $r \in \mathcal{R}$, reference object frame $f \in \mathcal{F}$, sequence of states $\vec{s} \in \mathcal{S}^\infty$, and sequence of actions $\vec{a} \in \mathcal{A}^\infty$.

As a running example, consider the humanoid task in Figure 2 where a humanoid with two robot arms is tasked with pouring from a red cup into a yellow bowl and then placing the bowl on a plate. This example has five skills: three for arm 1 (grasp, pour, and release cup) and two for arm 2 (grasp and place bowl). For this task, we acquired these skills by collecting bimanual demonstrations and then annotating intervals that correspond to each skill.

We adopt the procedure in MimicGen [21] for adapting skills to new states by assuming approximate pose invariance. Given a new state s_0 , we adapt each task-space end-effector state $s[e_r]$ and action $a[e_r]$ relative to object frame f using:

$$s'[e_r] \leftarrow (s_0[f]s[f]^{-1})s[e_r], \quad a'[e_r] \leftarrow (s_0[f]s[f]^{-1})a[e_r].$$

Section 4.1 converts adapted poses to joint targets through inverse kinematics or replays them directly in task space. Adaptation does not modify finger states or actions. Appendix A.1 explains how adapted states and actions are used during planning and replay.

3.3. Temporal Constraints

In order to complete manipulation tasks, the team of robots will generally need to execute multiple skills. When there's only a single robot, skills can only be executed sequentially [11]; however, multiple robots expose opportunities for parallelism, which enable quicker task completion. In the humanoid task in Figure 2, arm 1 and arm 2 can move simultaneously in order to pick their respective objects. Still, in order to correctly complete many tasks, certain skills need to be performed in sequence. In this task, arm 1 needs to pick the cup before it can pour. The pour must occur before arm 2 grasps the

bowl in order to ensure it receives the contents of the cup. In other tasks, such as the coordinated lift in the Build Tower example in Figure 1 (bottom left), both arms must execute a synchronized skill on the same object.

To model these dynamics, we introduce temporal skill constraints, which specify partial orders between specific steps of two skills, generalizing past notions [15, 18] in a manner aligned with mathematical programming. A temporal precedence constraint $p = \langle \psi, i, \psi', j \rangle$, where $0 \leq i < |\vec{s}_\psi|$ and $0 \leq j < |\vec{s}_{\psi'}|$, asserts that step i of skill ψ must execute no later than step j of skill ψ' . Let a skill plan $\langle \Psi, P \rangle$ be a set of robot skills $\psi \in \Psi$ and a set of temporal precedence constraints $p \in P$. Temporal constraints can represent many different dynamics. Consider the following examples¹ involving two skills ψ, ψ' . We represent *sequential* relationships using: $P = \{\langle \psi, -1, \psi', 0 \rangle\}$. If the skills have no ordering, then they can be executed in *parallel*: $P = \emptyset$. A coordination relationship between the two skills can be expressed as:

$$P = \{\langle \psi, 0, \psi', 0 \rangle, \langle \psi', 0, \psi, 0 \rangle, \langle \psi, -1, \psi', -1 \rangle, \langle \psi', -1, \psi, -1 \rangle\},$$

where two inequalities at the start $i = j = 0$ and end $i = j = -1$ of the skills result in an equality constraint.

4. ScheduleGen

ScheduleGen solves a mixed discrete-continuous planning & scheduling problem to generate demonstrations. Figure 2 outlines ScheduleGen’s demonstration generation phases. ScheduleGen takes as input the initial state s_0 and a skill plan $\langle \Psi, P \rangle$ (Section 3). Then, it operates in two phases. First, it plans joint-space trajectories per robot by adapting each skill ψ to the current state and connecting consecutive robot skills using motion planning (Section 4.1). Then, it schedules execution of each action among the robots, satisfying temporal and collision constraints as well as minimizing cost functions like schedule duration (Section 4.2). Finally, we describe some GPU acceleration considerations in Section 4.3.

Algorithm 1 ScheduleGen

```

1: procedure schedule-gen( $\mathcal{R}, p_0, S_*, \mathcal{D}; n$ )
2:    $D \leftarrow \emptyset$  ▷ Initialize the successful demo dataset
3:   while  $|D| < n$  do ▷ Loop until the dataset is complete
4:      $s_0 \sim p_0; d \sim \mathcal{D}$  ▷ Sample the initial state and demo
5:      $\langle \Psi, P \rangle \leftarrow \text{segment-skills}(d)$  ▷ Segment the demo
6:      $\mathcal{T} \leftarrow \text{serialized-planner}(\mathcal{R}, s_0, \langle \Psi, P \rangle)$ 
7:     if  $\mathcal{T} \neq \text{None}$  then
8:        $\tau \leftarrow \text{schedule-robots}(\mathcal{R}, \mathcal{T}, P)$ 
9:       if  $\tau \neq \text{None}$  then
10:         $\langle \vec{s}, \vec{a}, \vec{o} \rangle \leftarrow \text{execute}(s_0, \tau)$  ▷ Execute the demo
11:        if  $\vec{s}[-1] \in S_*$  then ▷ Check for demo success
12:           $D \leftarrow D \cup \{\langle \vec{o}, \vec{a} \rangle\}$  ▷ Add the successful demo
13:   return  $D$  ▷ Return the generated demos

```

Algorithm 1 gives the high-level pseudocode for `schedule-gen`. It takes in the set of robots $\mathcal{R}$, initial state distribution p_0 , goal state set S_* , set of source demonstrations $\mathcal{D}$, and a target number of demonstrations n . Until n successful demonstrations in D are generated, it randomly samples an initial state s_0 and demonstration d . The demonstration d is segmented into a skill plan $\langle \Psi, P \rangle$ (done a priori in practice). Then, `schedule-gen` computes a collection of trajectories $\mathcal{T}$ for the robots $\mathcal{R}$ that independently perform each skill (Section 4.1). This collection is safely and correctly scheduled into a single multi-robot trajectory τ by `schedule-robots`. The trajectory is executed in simulation producing states $\vec{s}$, actions $\vec{a}$, and observations $\vec{o}$. If terminal state $\vec{s}[-1]$ is in S_* , the demonstration is deemed a success and added to the successful demo dataset D .

¹Let an index value of -1 be shorthand for the last step on a skill.

4.1. Skill Planning

ScheduleGen’s first phase samples manipulation plans per robot that perform and connect the skills. Algorithm 2 presents the pseudocode for `serialized-planner`, which computes a manipulation plan for each robot that performs its associated skills. `serialized-planner` takes as input the set of robots $\mathcal{R}$, initial state s_0 , and skill plan $\langle \Psi, P \rangle$. It returns a mapping $\mathcal{T}$ from each robot $r \in \mathcal{R}$ to a sequence of robot configuration states and controller targets. First, `serialized-planner` topologically sorts Ψ into a sequence of skills by performing a Depth-First Search (DFS) using the sequential precedence constraints in P . Then, it processes each skill $\psi = \langle r, f, \vec{s}_\psi, \vec{a}_\psi \rangle$ in resulting order. Using the current pose of $s_0[f]$ for object frame f , it adapts the states $\vec{s}_\psi$ and actions $\vec{a}_\psi$ on the skill demonstration (Section 3.2). Each adapted end-effector pose $s'_\psi[e_r]$ serves as the target for iterative inverse kinematics, inducing a skill trajectory τ_ψ . Then, `serialized-planner` plans a motion trajectory τ_m to the skill’s initial robot configuration. The motion and skill trajectories τ_m and τ_ψ are logged in $\mathcal{T}$ for robot r , and the robot and object states are updated in s_0 . Inverse kinematics and motion planning can condition their behavior on the current state s_0 ; however, collision constraints with other moving robots are deferred until Section 4.2. If inverse kinematics or motion planning fails, `serialized-planner` returns **None** and the candidate is rejected. Later attempts resample the initial state and source demonstration.

Algorithm 2 Serialized Multi-Robot Planning

```

1: procedure serialized-planner( $\mathcal{R}, s_0, \langle \Psi, P \rangle$ )
2:    $\mathcal{T} \leftarrow \{r : [] \text{ for } r \in \mathcal{R}\}$  ▷ Initial robot trajectories
3:   for  $\langle r, f, \vec{s}_\psi, \vec{a}_\psi \rangle \in \text{topological-sort}(\Psi, P)$  do
4:      $\tau_\psi \leftarrow []$  ▷ Initialize skill trajectory
5:     for  $\langle s'_\psi, a'_\psi \rangle \in \text{adapt-demo}(\vec{s}_\psi, \vec{a}_\psi, s_0[f])$  do
6:        $s'_\psi[r] \leftarrow \text{inverse-kinematics}(r, s'_\psi[e_r]; s_0)$ 
7:        $a'_\psi[r] \leftarrow \text{inverse-kinematics}(r, a'_\psi[e_r]; s_0)$ 
8:       if  $s'_\psi[r] = \text{None}$  or  $a'_\psi[r] = \text{None}$  then
9:         return None
10:       $\tau_\psi \leftarrow \tau_\psi + [\langle s'_\psi, a'_\psi \rangle]$  ▷ Add skill state and action
11:       $\langle s_\psi, - \rangle \leftarrow \tau_\psi[0]$  ▷ Initial skill state
12:       $\tau_m \leftarrow \text{motion-planner}(r, s_0[r], s_\psi[r]; s_0)$ 
13:      if  $\tau_m = \text{None}$  then
14:        return None
15:       $\mathcal{T}[r] \leftarrow \mathcal{T}[r] + \tau_m + \tau_\psi$  ▷ Concatenate new trajectories
16:       $s_0 \leftarrow s'_\psi$ ; ▷ Update the state
17:   return  $\mathcal{T}$ 

```

4.2. Skill Scheduling

The output of Section 4.1 is a collection of trajectories $\mathcal{T} = \{r : \tau_r \mid r \in \mathcal{R}\}$ for robots $\mathcal{R}$. At this point, prior work would execute the trajectories sequentially [11] or simultaneously but myopically [18], but these schemes result in inefficient or unsafe demonstrations because they don’t reason about collision-free simultaneous motion. For example, in Figure 2, even if arm 2 doesn’t initiate its pick bowl skill, moving immediately to the bowl obstructs arm 1 from safely pouring the cup. As a result, long-horizon scheduling is required to coordinate this behavior.

To produce high-quality multi-arm demonstrations, we propose *scheduling* the skills in order to minimize execution time while respecting task and collision constraints. We accomplish this by implementing `schedule-robots`($\mathcal{R}, \mathcal{T}, P$) using mathematical programming. We define nonnegative time decision variables $\mathbf{t}_r$, where $|\mathbf{t}_r| = |\mathcal{T}[r]| = n_r$ for each robot r . The objective is to minimize the *makespan* of the schedule $\mathbf{t}_*$, the maximum robot execution duration, subject to constraints: $\mathbf{t}_* = \max_{r \in \mathcal{R}} \mathbf{t}_r[-1]$. We formulate safe scheduling as a mathematical program.

The objective minimizes makespan together with the sum of robot end times, weighted by $w \geq 0$.

The experiment in Table 2 ablates other secondary costs. Constraint one sets the start time $t_r[0]$ to be zero. Constraint two ensures the makespan is no less than each end time $t_r[-1]$. Constraint three gives the minimum time step Δt between waypoints. The minimum time step prevents scheduling from increasing waypoint-to-waypoint speed; acceleration is not explicitly constrained by this formulation. Constraint four enforces the temporal constraints P between robots $r, r' \in \mathcal{R}$. Constraint five is a disjunctive collision constraint: if configurations $q_r[i], q_{r'}[j]$ collide, robot r must finish i before robot r' starts j (plus a safety parameter $\epsilon > 0$) or vice versa. Critically, the disjunctive constraint makes this a Mixed-Integer Linear Program (MILP) as there’s a discrete decision for which order to impose whenever there’s a collision. We create binary decision variables indicating which clause is active and use the Big-M method to enforce that only one constraint is active. Appendix A.2 gives the resulting inequalities.

$$\begin{aligned}
 \min_{\mathbf{t}_*} \quad & \mathbf{t}_* + w \sum_{r \in \mathcal{R}} \mathbf{t}_r[-1] & (1) \\
 \text{s.t.} \quad & 0 = \mathbf{t}_r[0] & \forall r \in \mathcal{R} \\
 & \mathbf{t}_r[-1] \leq \mathbf{t}_* & \forall r \in \mathcal{R} \\
 & \mathbf{t}_r[i] + \Delta t \leq \mathbf{t}_r[i+1], & \forall r \in \mathcal{R}, i \in [n_r - 1] \\
 & \mathbf{t}_r[i] \leq \mathbf{t}_{r'}[j], & \forall \langle \psi_r, i, \psi_{r'}, j \rangle \in P \\
 & \mathbf{t}_r[i+1] + \epsilon \leq \mathbf{t}_{r'}[j] & \forall r \neq r' \in \mathcal{R}, i \in [n_r - 1], \\
 & \text{or } \mathbf{t}_{r'}[j+1] + \epsilon \leq \mathbf{t}_r[i], & \text{if collision}(q_r[i], q_{r'}[j])
 \end{aligned}$$

We extract a joint robot trajectory τ from the MILP by rounding the time variables t_r , bucketing them in indices $\lfloor t/\Delta t \rfloor$, and maintaining the same set point per robot until the next waypoint. If any robot start $q_r[0]$ or end $q_r[-1]$ states collide, then the MILP is infeasible.

4.3. GPU Acceleration

In order to formulate the MILP in section 4.2, we need to be able to evaluate the collision constraints using `collision`. This requires checking all configuration pairs across robot trajectories, inducing $\mathcal{O}(|\mathcal{R}|^2 n_{\max}^2)$ collision checks for $n_{\max} = \max_r n_r$. To combat this, we leverage GPU acceleration to evaluate these collisions in parallel, sometimes in a single batch call. We frame robot-robot collision checking as a pairwise self-collision check in a composite robot made up of the two robots [9]. We use cuRobo’s custom self-collision CUDA kernel [26, 27] for the sphere-sphere collision checks. We use a tangent sphere inflation [14] strategy to compute spherical decompositions of the robots. We represent grasped objects during scheduling also as a set of spheres that temporarily augment the collision geometry of the robot.

5. Experiments

We evaluate ScheduleGen on bimanual, four-arm, humanoid, benchmark-transfer, and real-world tasks.² Section 5.1 defines the common settings and task suite, Section 5.2 presents the capabilities enabled by ScheduleGen, Section 5.3 analyzes its data-generation and policy behavior, and Section 5.4 reports real-world transfer.

5.1. Experimental Setup

Data-Generation and Policy Training Settings. Unless otherwise stated, we use five annotated source demonstrations per task, share them across variants and baselines, and collect 1,000 successful

²Experiment videos are available at <https://schedulegen.github.io/>.

demonstrations per variant. We train flow-matching visuomotor [19] policies on the demonstrations and evaluate them on held-out resets. We report data-generation success, policy success, and mean steps to completion over successful rollouts. Appendix D gives the policy training settings.

Baselines. We compare ScheduleGen with DexMimicGen [15], which transforms and composes per-arm demonstration segments independently, and DexMimicGen-Tuned, which adds manually specified task-level holds and ordering constraints to reduce collisions caused by arm-arm overlap for the defined reset distributions.

Tasks and Variants. We define five task families in Isaac Lab [23]: YAM Bowl and Dishware Stacking, DROID Two- and Four-Arm Cube Stacking, and fixed-waist GR1T2 Humanoid Nut Pouring. *Normal* denotes the source layout, *Cross* moves the arm-object assignments across the robot centerline while reusing the same source demonstrations, and *Bin* adds a shallow bin as an obstacle/clutter condition on both Normal and Cross. Four-Arm Cube Stacking builds a five-object tower on a table or in a bin under a base reset support (Distribution I) and an expanded support (Distribution II). We additionally evaluate four robosuite variants in MuJoCo and seven released RoboDojo tasks on the X5 bimanual setup [3] (Fig. 4). For RoboDojo policy evaluation and the aggregate, *Nominal* uses held-out layouts from the training reset distribution, while *Perturbed* uses the same normalized draws with object-pose perturbations doubled.

5.2. ScheduleGen Features

ScheduleGen generates demonstrations across diverse tasks and robot embodiments. Across 17 Isaac Lab variants, ScheduleGen achieves 61.1-98.0% data-generation success, exceeding 85% on 10 variants. All policies trained on this data reach 30-100% success (Fig. 3). These results span bimanual YAM and DROID [16], a four-arm DROID setup, and a GR1T2 humanoid. Data-generation success also remains high (75.5-100.0%) across seven RoboDojo tasks on X5 (Fig. 5). On four robosuite variants in MuJoCo, BC-RNN-GMM policies reach 60-100% success over 50 episodes from data generated at 24.7-84% success (Table 1).

ScheduleGen generates collision-free demonstrations even in scenarios with inter-arm overlap, unlike prior work. The Cross-Arm variants introduce overlapping arm workspaces and broader object resets while reusing the Normal task demonstrations. DexMimicGen transfers each arm independently, while the manually selected holds in DexMimicGen-Tuned apply fixed timing across resets. ScheduleGen instead replans connecting motions and schedules collision-free waits for each reset. Across the four bimanual Cross and Cross-Bin variants, it reaches 64-88% data-generation success and 70-86% policy success. Two-Arm Cube Stacking uses a state-based BC-RNN, while Four-Arm Cube Stacking uses state-based flow-matching policies. DexMimicGen generates no valid trajectories on these variants, while DexMimicGen-Tuned remains below ScheduleGen throughout (Fig. 3).

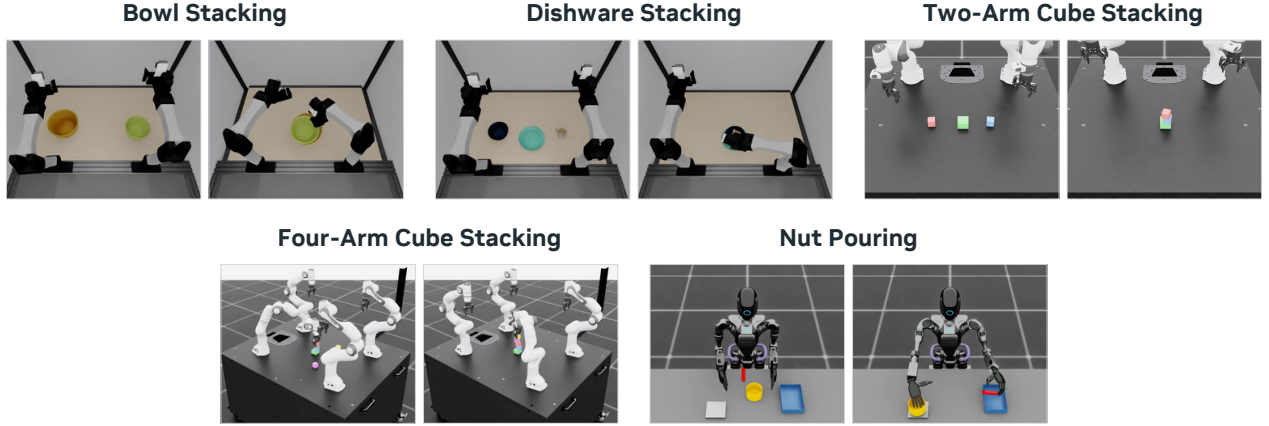
ScheduleGen scales data generation and policy learning to multi-arm systems. Four-Arm Cube Stacking uses motion-planned pick-and-place segments as source skills rather than teleoperation. ScheduleGen jointly schedules them across four arms, checking all six arm pairs while building a five-object tower on a table or in a narrow bin. We train a separate state-based flow-matching policy for each layout and reset distribution. Across eight settings, data-generation success is 63.4-95.1% and policy success is 30-90% (Fig. 3). DexMimicGen is not applicable because its interface supports only bimanual generation.

ScheduleGen uses scheduling costs to generate different coordination strategies. On the RoboDojo Insert Tubes task, we compare Makespan with three secondary costs (Section 4.2) under the same collision, precedence, synchronization, and per-arm ordering constraints. All four modes generate 500 successful demonstrations from the same five sources while maintaining 92.08-94.16% data-generation success. Each dataset supports policy learning, with 40.0-50.0% success over 60 held-out Nominal episodes per mode and 237.8-240.3 mean steps over successful rollouts (Table 2).

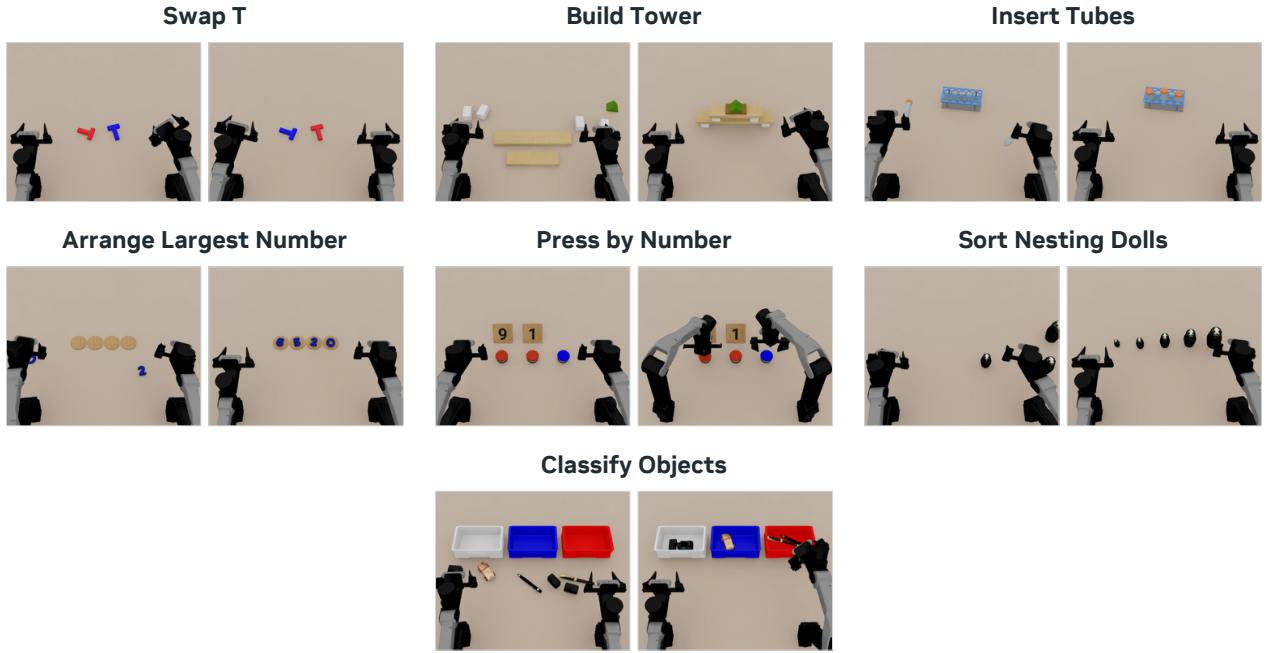


Figure 3: **Simulation data generation and policy evaluation.** Rows group tasks and embodiments; columns show data generation success, policy success, and mean steps over successful policy episodes. Methods share source demonstrations within each family. Policies use 50 evaluation episodes. Four-arm policies train separately for distributions I and II; DexMimicGen baselines are unsupported. Labels are rounded; bar heights retain the original values. Filled baseline markers denote zero; hollow markers denote unavailable steps. Unsupported four-arm baselines are omitted. Step axes use task-specific ranges.

Isaac Lab



RoboDojo



MuJoCo / robosuite

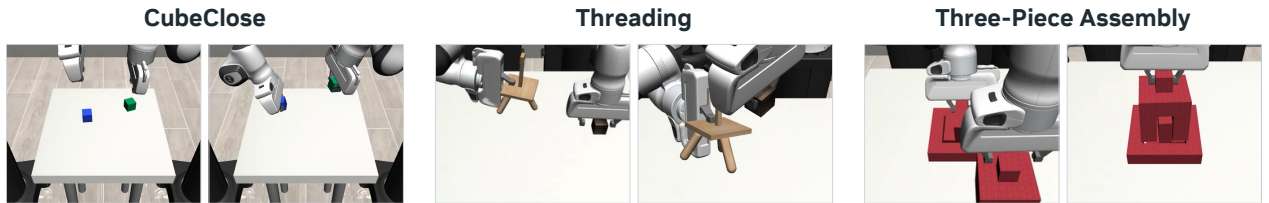


Figure 4: **Simulation task suite.** Five Isaac Lab, seven RoboDojo, and three robosuite simulated task families are shown. Paired views show task execution (*left*) and completion (*right*). CubeClose is evaluated under D0 and D1.

Table 1: **Robosuite experiments.** ScheduleGen improves data generation success rate on three of four variants, improves or matches policy success wherever DexMimicGen (DexMG) trains a policy, and alone generates successful data on CubeClose D1.

Task	Method	DGR (%)	Policy SR (%)
CubeClose D0	DexMG	87.6	98
	ScheduleGen	84.0	100
CubeClose D1	DexMG	0.0	—
	ScheduleGen	72.3	94
Threading	DexMG	20.7	60
	ScheduleGen	24.7	60
TPA	DexMG	14.0	66
	ScheduleGen	39.7	80

Table 2: **Scheduling cost ablation.** Effect of scheduling costs on data generation and policy learning. Each cost generates 500 successful demonstrations. Policy success is measured over 60 Nominal episodes per cost; mean policy steps are calculated over successful rollouts. Appendix B defines the secondary costs.

Scheduling cost	Data generation success (%)	Policy success (%)	Mean policy steps ↓
Makespan Minimize just the schedule duration.	94.16	43.3	239.8
Makespan + continuity Minimize the longest internal pause.	93.46	43.3	240.2
Makespan + max-min velocity Maximize the slowest moving-edge velocity.	92.08	40.0	237.8
Makespan + minimum active span Minimize summed arm-active spans.	93.46	50.0	240.3

ScheduleGen scales released demonstrations to externally authored tasks. Agentic annotation converts five released RoboDojo demonstrations per task into seeds with per-arm contact boundaries, object-relative frames, and task constraints; ambiguous cases retain human review, and all annotations must pass replay and task-success checks. ScheduleGen then generates 500 successful training demonstrations per task at 75.5-100.0% success (89.1% mean). The resulting flow-matching policies complete all seven tasks under both conditions, averaging 47.9% success on held-out Nominal layouts and 25.0% under doubled pose perturbations over 20 episodes per task and condition (Fig. 5).

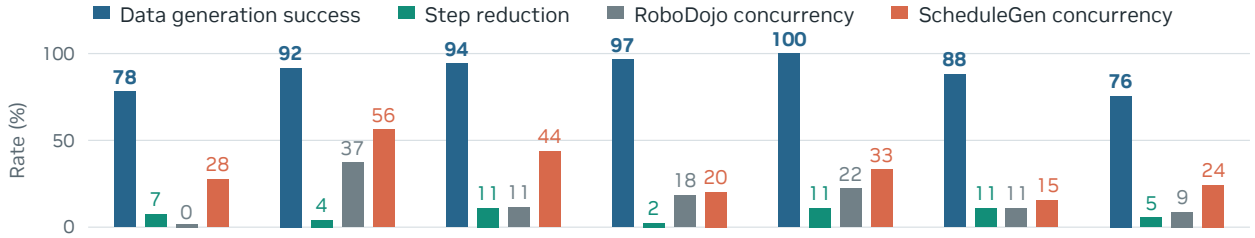
5.3. ScheduleGen Analysis

ScheduleGen reduces the task-specific data collection and tuning needed for difficult layouts. We separately collect Cross-Arm demonstrations for YAM Bowl and Dishware Stacking. Each set takes the operator about 20 minutes, with additional work for annotation and hold tuning. To generate 1,000 demonstrations, DexMimicGen reaches 32.1% and 11.2% data-generation success and 30% and 20% policy success evaluated over 50 episodes. ScheduleGen instead reuses the five Normal demonstrations, reaching 66.7% and 68.0% data-generation success and 80% and 76% policy success. Task-matched teleoperation alone therefore does not close the gap: ScheduleGen retains the contact skills while replanning and scheduling free-space motion for each reset.

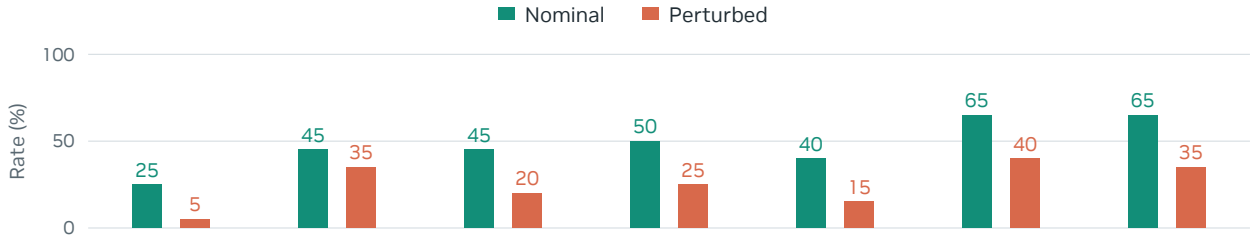
Policies trained on ScheduleGen data outperform the best DexMimicGen baseline across tasks and embodiments. Across nine variants supported by DexMimicGen, ScheduleGen policies improve success on eight and match the best baseline at 94% on Normal Two-Arm Cube Stacking. Policy success rises from 42-78% to 76-100% on Bowl and Dishware Stacking, 0-74% to 70-86% on the Cross-Arm and Cross-Bin cube variants, and 20-90% to 48-96% on Humanoid Nut Pouring. ScheduleGen policies also use fewer steps whenever a baseline policy succeeds (Fig. 3).

ScheduleGen shortens RoboDojo trajectories by scheduling more arm motion in parallel. Across the 3,500 trajectories used for policy training, ScheduleGen reduces mean execution length by 2.3-11.1% per task relative to the released RoboDojo trajectories, with a 7.5% average reduction across tasks. Both arms are active together for 31.5% of execution steps, compared with 15.5% in the released demonstrations. To measure scheduling alone, we also apply the scheduler to all 700 released trajectories while keeping their paths and task order fixed. It finds valid schedules for 98.3% of these trajectories and reduces total steps by 14.98% without replanning or physics replay. The mean reduction per episode is 16.44% (Fig. 5). Appendix C.5 gives the per-task measurements.

(a) Generated training data



(b) Learned policies



(c) Source-only scheduling

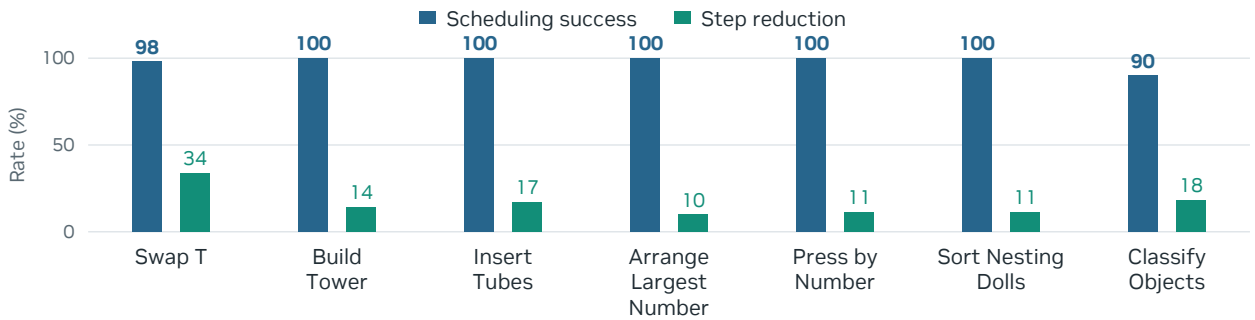


Figure 5: **RoboDojo results.** (a) Data generation success, reduction in mean steps relative to released demonstrations, and concurrent arm motion for 500 generated trajectories per task. (b) Policy success over 20 Nominal and 20 Perturbed episodes per task. (c) Scheduling success and mean per-trajectory step reduction for 700 fixed released trajectories (Table 4). Concurrency is the fraction of steps when both arms move; labels are rounded. A short baseline marker denotes zero.

5.4. Real-World Evaluation

We evaluate two transfer routes on the real YAM setup. In sim-to-real, policies are trained only on ScheduleGen simulation data. In real-to-sim-to-real, calibrated real demonstrations seed simulation data generation for Cube Stacking and Trash, and the resulting policies are returned to the same robot. All policies are trained in simulation without real-world fine-tuning. We use calibrated robot, camera, workspace, and object geometry, and randomize object pose, appearance, lighting, camera intrinsics and extrinsics, and background. Policies observe three RGB views and both arms’ joint and gripper states, and predict joint-position and gripper targets.

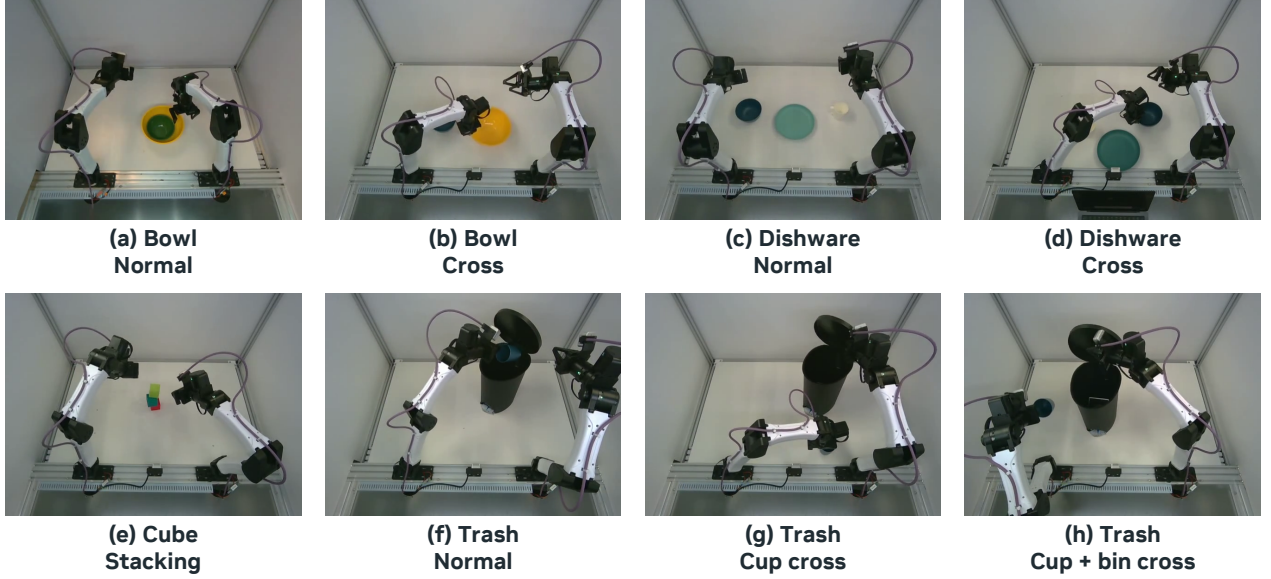


Figure 6: **ScheduleGen policies transfer to the real world.** Top: successful sim-to-real Bowl and Dishware Stacking rollouts (Normal and Cross). Bottom: real-to-sim-to-real Cube Stacking and Trash rollouts (Normal, Cup cross, and Cup + bin cross). Each policy is evaluated over 20 real-world episodes; Table 3 reports success rates.

Table 3: **Simulation data-generation and real-world policy success.** Each policy is evaluated over 20 real-world episodes without real-world fine-tuning. Each variant uses 1,000 ScheduleGen-generated simulation demonstrations. Sim-to-real uses no real sources; real-to-sim-to-real uses calibrated real sources, collected only in the Normal layout for Trash. DGR: data generation success rate in simulation. SR: real-world policy success rate.

Experiment	Task	Sim DGR (%) ↑	Real SR (%) ↑
Sim-to-real	Bowl Stacking (Normal)	96.5	75
	Bowl Stacking (Cross-Arm)	66.7	80
	Dishware Stacking (Normal)	71.2	80
	Dishware Stacking (Cross-Arm)	68.0	75
Real-to-sim-to-real	Cube Stacking	64.6	65
	Trash (Normal)	74.9	75
	Trash (Cup cross)	59.5	70
	Trash (Cup + bin cross)	63.6	50

ScheduleGen’s simulation data train policies that transfer directly to the real YAM. The sim-to-real policies use no real demonstrations. Over 20 episodes per task, they reach 75% success on Normal Bowl Stacking, 80% on Cross-Arm Bowl Stacking, 80% on Normal Dishware Stacking, and 75% on Cross-Arm Dishware Stacking (Fig. 6).

ScheduleGen expands a small set of real demonstrations into simulation datasets that train policies for real deployment. After reconstructing and physics-validating five real Cube Stacking demonstrations, ScheduleGen generates a randomized simulation dataset whose policy reaches 65% success over 20 real-world episodes.

For Trash, real source demonstrations collected only in the Normal layout seed all three variants. ScheduleGen generates 1,000 demonstrations per variant, with data generation success rates of 74.9%, 59.5%, and 63.6% for Normal, Cup cross, and Cup + bin cross, respectively. The resulting policies reach 75%, 70%, and 50% success over 20 real-world episodes per variant. Appendix C.1 gives the metric definitions.

6. Conclusion

We presented ScheduleGen, an automated demonstration generation system for multi-arm imitation learning. It combines object-centric skill adaptation, motion planning, and joint scheduling to preserve contact-rich behavior while satisfying task order, synchronization, and collision constraints. Experiments across 28 simulation variants show that ScheduleGen supports diverse embodiments and challenging layouts, and trains policies that outperform prior bimanual methods on most comparable tasks. The learned policies also transfer to eight real-world variants without real-world fine-tuning. These results show that adapting both motion and coordination can reduce task-specific data collection while supporting long-horizon, contact-rich multi-arm manipulation.

7. Acknowledgments

The authors would like to thank Desmond Kyu and Ramanpreet Singh for data collection and model evaluation; Matin Nikoui and Lion Park for hardware support; Tianyuan Dai for help with simulation and domain randomization infrastructure; and Tuur Stuyck, Peter Du, Yashraj Narang, and Fabio Ramos for feedback and valuable discussions.

References

- [1] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 2, 3
- [2] Jingkai Chen, Jiaoyang Li, Yijiang Huang, Caelan Garrett, Dawei Sun, Chuchu Fan, Andreas Hofmann, Caitlin Mueller, Sven Koenig, and Brian C Williams. Cooperative task and motion planning for multi-arm assembly systems. *arXiv preprint arXiv:2203.02475*, 2022. 3
- [3] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Weijie Wan, Baijun Chen, Haoran Lu, Haowen Yan, Honghao Su, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026. 8
- [4] Murtaza Dalal, Ajay Mandlekar, Caelan Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. *arXiv preprint arXiv:2305.16309*, 2023. 2, 3
- [5] Kourosh Darvish, Luigi Penco, Joao Ramos, Rafael Cisneros, Jerry Pratt, Eiichi Yoshida, Serena Ivaldi, and Daniele Pucci. Teleoperation of humanoid robots: A survey. *IEEE Transactions on Robotics*, 39(3):1706–1727, 2023. 2, 3
- [6] Abhay Deshpande, Maya Guru, Rose Hendrix, Snehal Jauhri, Ainaz Eftekhari, Rohun Tripathi, Max Argus, Jordi Salvador, Haoquan Fang, Matthew Wallingford, et al. Molmob0t: Large-scale simulation enables zero-shot manipulation. *arXiv preprint arXiv:2603.16861*, 2026. 2
- [7] Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 international symposium on multi-robot and multi-agent systems (MRS)*, pages 120–127. IEEE, 2017. 3

- [8] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets. In *Robotics: Science and Systems*, 2022. 2
- [9] Caelan Garrett and Fabio Ramos. Schedulstream: Temporal planning with samplers for gpu-accelerated multi-arm task and motion planning & scheduling. *arXiv preprint arXiv:2511.04758*, 2025. 3, 7
- [10] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021. 3
- [11] Caelan Reed Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. Skillgen: Automated demonstration generation for efficient skill learning and deployment. In *8th Annual Conference on Robot Learning*, 2024. 3, 4, 6
- [12] Weihang Guo, Zachary Kingston, Kaiyu Hang, and Lydia E Kavraki. Efficient multi-robot motion planning for manifold-constrained manipulators by randomized scheduling and informed path generation. *arXiv preprint arXiv:2412.00366*, 2024. 3
- [13] Valentin N Hartmann, Andreas Orthey, Danny Driess, Ozgur S Oguz, and Marc Toussaint. Long-horizon multi-robot rearrangement planning for construction assembly. *IEEE Transactions on Robotics*, 39(1):239–252, 2022. 3
- [14] Masatomo Inui, Nobuyuki Umezu, and Ryohei Shimane. Shrinking sphere: A parallel algorithm for computing the thickness of 3d objects. *Computer-Aided Design and Applications*, 13(2):199–207, 2016. 7
- [15] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. *arXiv preprint arXiv:2410.24185*, 2024. 2, 3, 5, 8
- [16] 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. 2, 8
- [17] Xuanlin Li, Tong Zhao, Xinghao Zhu, Jiuguang Wang, Tao Pang, and Kuan Fang. Planning-guided diffusion policy learning for generalizable contact-rich bimanual manipulation. *arXiv preprint arXiv:2412.02676*, 2024. 3
- [18] Kevin Lin, Ajay Mandlekar, Caelan Reed Garrett, Nikita Chernyadev, Yu Fang, Runyu Ding, Yuqi Xie, Linxi Fan, and Yuke Zhu. Humanoidmimicgen: Data generation for loco-manipulation via whole-body planning and adaptation. In *ICRA 2026 Workshop on Synthetic Data for Robot Learning*, 2026. 3, 4, 5, 6
- [19] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. 8
- [20] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conf on Robot Learning*, 2018. 2
- [21] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Ireteyio Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *7th Annual Conference on Robot Learning*, 2023. 2, 3, 4
- [22] Michael James McDonald and Dylan Hadfield-Menell. Guided imitation of task and motion planning. In *Conference on Robot Learning*, pages 630–640. PMLR, 2022. 2, 3
- [23] Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Munoz, Xinjie Yao, René Zurbrugg, Nikita Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025. 8
- [24] NVIDIA, 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. 2, 3
- [25] Dean A Pomerleau. Alvin: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 2
- [26] 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 robot motion generation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. 7

- [27] Balakumar Sundaralingam, Adithyavairavan Murali, and Stan Birchfield. curoboV2: Dynamics-aware motion generation with depth-fused distance fields for high-dof robots. *arXiv preprint arXiv:2603.05493*, 2026. 7
- [28] Marc Toussaint and Manuel Lopes. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4044–4051, 2017. ISBN 9781509046331. doi: 10.1109/ICRA.2017.7989464. 3
- [29] Albert Tung, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Learning multi-arm manipulation through collaborative teleoperation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9212–9219. IEEE, 2021. 2, 3
- [30] Hejia Zhang, Shao-Hung Chan, Jie Zhong, Jiaoyang Li, Sven Koenig, and Stefanos Nikolaidis. A mip-based approach for multi-robot geometric task-and-motion planning. In *2022 IEEE 18th international conference on automation science and engineering (CASE)*, pages 2102–2109. IEEE, 2022. 3
- [31] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE Int’l Conf on robotics and automation (ICRA)*, 2018. 2
- [32] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems*, Daegu, Republic of Korea, 2023. 2, 3

Appendix

- A. Planning and Scheduling Details
- B. Scheduling Costs
- C. Additional Results and Evaluation
- D. Policy Training

A. Planning and Scheduling Details

Algorithms 1 and 2 give the generation loop and skill planner. Here we describe how states and actions form the arm paths and expand the collision constraints in Section 4.2.

A.1. Path Construction

Algorithm 2 processes skills in a topological order of their sequential precedence constraints, updating object poses for later skills. This order is used to construct the planning scene; the arms can still execute in parallel after scheduling.

The adapted state sequence defines the configurations used for planning and collision checking, while the adapted action sequence supplies the controller targets for replay. We solve inverse kinematics for both sequences and retain the demonstrated finger states and actions. Connecting motions are inserted between skills as in Fig. 2(a). We then map each skill step to its index in the full arm path so that the scheduler can enforce the original temporal constraints.

A.2. Collision Constraints and Execution

Let C be the set of waypoint pairs (r, i, r', j) for which $\text{collision}(q_r[i], q_{r'}[j])$ holds in Eq. 1. For each pair, a binary variable $b_{r,i,r',j}$ selects which arm moves through the conflict first. The Big-M constraints are

$$\begin{aligned} \mathbf{t}_r[i+1] + \epsilon &\leq \mathbf{t}_{r'}[j] + Mb_{r,i,r',j}, \\ \mathbf{t}_{r'}[j+1] + \epsilon &\leq \mathbf{t}_r[i] + M(1 - b_{r,i,r',j}). \end{aligned} \quad (2)$$

If $b_{r,i,r',j} = 0$, the first inequality is enforced; if it is 1, the second is enforced. M must be large enough to make the other inequality inactive over the scheduling horizon. These constraints preserve waypoint order and the task’s precedence and synchronization constraints while changing path timing. They apply to both connecting motions and contact skills, including the approach conflict in the Nut Pouring example (Fig. 2(b–c)). A feasible schedule is replayed in simulation to check contact behavior and task completion before the demonstration is accepted.

B. Scheduling Costs

Equation 1 minimizes makespan together with the weighted sum of arm end times. This section defines the secondary costs compared in Table 2. They change path timing while preserving the task, collision, and per-arm ordering constraints.

B.1. Primary and Secondary Objectives

Let $\mathbf{t}$ collect the waypoint times $\mathbf{t}_r[i]$, and let $\mathcal{S}_{\text{feas}}$ be the schedules satisfying the constraints in Eq. 1. As in the main formulation, makespan is the latest arm end time, $\max_{r \in \mathcal{R}} \mathbf{t}_r[n_r - 1]$.

For the cost ablation, we minimize makespan first and then optimize a secondary cost among schedules with that same makespan:

$$\begin{aligned} \mathbf{t}_*^{\min} &= \min_{\mathbf{t} \in \mathcal{S}_{\text{feas}}} \max_{r \in \mathcal{R}} \mathbf{t}_r[n_r - 1], \\ \mathbf{t}_c^* &\in \arg \min_{\substack{\mathbf{t} \in \mathcal{S}_{\text{feas}} \\ \max_{r \in \mathcal{R}} \mathbf{t}_r[n_r - 1] = \mathbf{t}_*^{\min}}} c(\mathbf{t}). \end{aligned} \quad (3)$$

Makespan alone uses the first stage. The other modes also use the second stage, so their secondary objective cannot increase the minimum makespan. This differs from the weighted objective in Eq. 1, which balances makespan against arm end times. Schedules with the same makespan can still differ in when each arm moves, waits, and finishes; the secondary costs distinguish these timings.

B.2. Secondary Cost Definitions

For consecutive waypoints, define the interval in control ticks and the joint-space distance as

$$g_r[i] = \frac{\mathbf{t}_r[i+1] - \mathbf{t}_r[i]}{\Delta t} \geq 1, \quad \ell_r[i] = \|q_r[i+1] - q_r[i]\|_2.$$

The lower bound on $g_r[i]$ follows from the minimum time step in Eq. 1.

Continuity.

The set $\mathcal{E}_{\text{int}}$ contains edges within an arm’s active interval, excluding its launch delay and terminal hold. The continuity cost is

$$c_{\text{cont}}(\mathbf{t}) = \max_{(r,i) \in \mathcal{E}_{\text{int}}} (g_r[i] - 1). \quad (4a)$$

The quantity $g_r[i] - 1$ measures the interval beyond an edge’s minimum one-tick duration. Minimizing its largest value discourages long internal pauses. Excluding launch delays and terminal holds allows an arm to wait before starting or after finishing. In Nut Pouring (Fig. 2), the bowl arm can delay its approach until the other arm finishes pouring, then move without a long internal pause.

Max-min velocity.

The set $\mathcal{E}_{\text{move}}$ contains edges with $\ell_r[i] > \delta_q$, where δ_q is a joint-motion threshold, separate from the collision time margin ϵ . The cost is

$$c_{\text{vel}}(\mathbf{t}) = \max_{(r,i) \in \mathcal{E}_{\text{move}}} \frac{g_r[i]}{\ell_r[i]}. \quad (4b)$$

For a moving edge, joint-space speed is $\ell_r[i]/(g_r[i]\Delta t)$. Since Δt is fixed, minimizing the largest $g_r[i]/\ell_r[i]$ maximizes the slowest moving-edge speed. Stationary edges are excluded to avoid division by zero, so this cost does not directly penalize their duration. Faster moving edges can therefore coexist with long pauses (Appendix C.4).

Minimum active span.

Let u_r^{first} and u_r^{last} be the first and last control ticks that advance a moving edge. The cost is

$$c_{\text{span}}(\mathbf{t}) = \sum_{r \in \mathcal{R}} (u_r^{\text{last}} - u_r^{\text{first}} + 1). \quad (4c)$$

The span $u_r^{\text{last}} - u_r^{\text{first}} + 1$ counts control ticks from an arm’s first movement through its last. Minimizing the sum favors shorter active intervals and discourages waits between movements. Launch delays and terminal holds do not contribute to this cost. Continuity targets the longest internal pause; minimum active span also accounts for the accumulated duration of several shorter pauses.

C. Additional Results and Evaluation

This section expands the experiments in Section 5. We examine source demonstration reuse, explain how released demonstrations become skill seeds, and show how scheduling changes coordination and execution length.

C.1. Evaluation Metrics

Data generation success rate is the fraction of attempts that produce a successful demonstration after simulation replay. Policy success rate is measured over held-out evaluation episodes; mean policy completion steps use successful rollouts only. In real-world transfer, generation is evaluated in simulation and policy success is evaluated on the robot. Aggregate simulation rates average the 28 variants equally and use Nominal evaluation for RoboDojo.

In Table 1, the CubeClose D1 dash means that no DexMimicGen policy was trained because generation produced no successful demonstrations. On CubeClose D0, ScheduleGen has lower data generation success but higher policy success. The two rates measure different stages and need not rank methods identically.

C.2. Source Demonstration Reuse

The Cross-Arm comparison in Section 5.3 tests whether collecting demonstrations directly in the target layout closes the gap to ScheduleGen. DexMimicGen uses newly collected Cross-Arm demonstrations for Bowl and Dishware Stacking, while ScheduleGen reuses the five Normal demonstrations. Both methods are evaluated on the Cross-Arm variants, where the arms reach across the robot centerline.

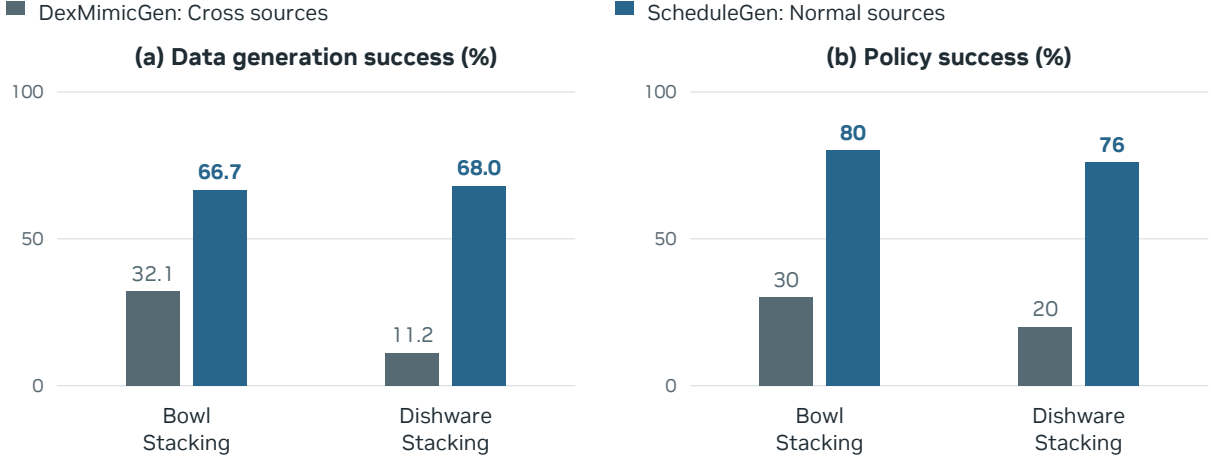


Figure 7: **Source layout and Cross-Arm performance.** DexMimicGen receives newly collected Cross-Arm sources; ScheduleGen reuses Normal sources. The comparison reports data generation success and learned-policy success on the same two task variants. Each policy is evaluated over 50 episodes.

Figure 7 shows that task-matched sources alone do not recover the performance of ScheduleGen. A source demonstration provides the contact skills, but its connecting motions and timing may no longer be suitable when objects move. ScheduleGen plans those motions and coordinates the arms for each sampled scene, reducing the need to recollect demonstrations for a new layout.

C.3. RoboDojo Demonstration Transfer

RoboDojo demonstrations provide RGB observations and joint actions. We use calibrated object geometry to reconstruct the scene in simulation and check the transferred motion through physics replay. Skill annotations specify per-arm contact boundaries, object-relative frames, and temporal constraints. A seed is used for generation only after replay and task-success checks.

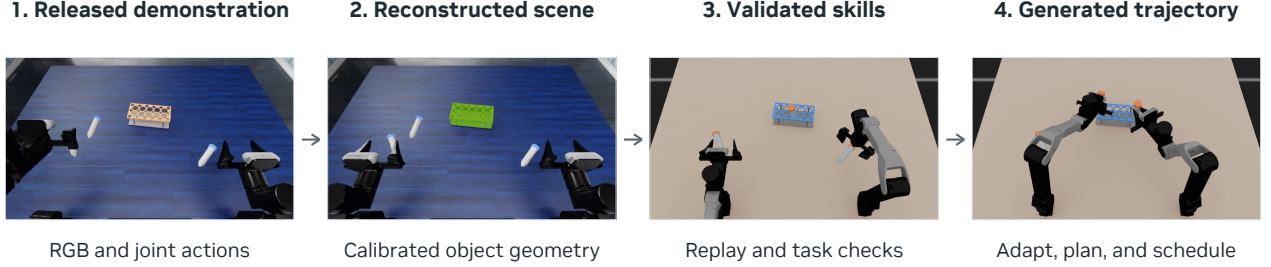


Figure 8: **RoboDojo demonstration transfer.** Insert Tubes illustrates scene calibration, skill validation through replay, and trajectory generation. The same procedure supplies the five source demonstrations used for each RoboDojo task.

Algorithms 1 and 2 reuse these contact skills and task constraints in new scenes. ScheduleGen adapts the skills, plans connecting motions, and schedules the arms; generated demonstrations must pass replay before entering the training set.

C.4. Effects of Scheduling Costs

Figure 9 compares scheduling costs on the same Insert Tubes skills and target scene. All modes finish their paths in 10.44 s (261 control ticks at 25 Hz); a shared 25-tick final hold brings each successful replay to 11.44 s.

An arm’s active span runs from its first movement through its last, including internal pauses; panel (b) sums the spans over both arms. Continuity and minimum active span remove internal pauses and shorten this sum from 17.00 to 15.40 s. Max-min velocity permits a longer pause, showing that improving moving-edge speed need not give continuous motion. Appendix B defines the costs; Table 2 reports policy results.

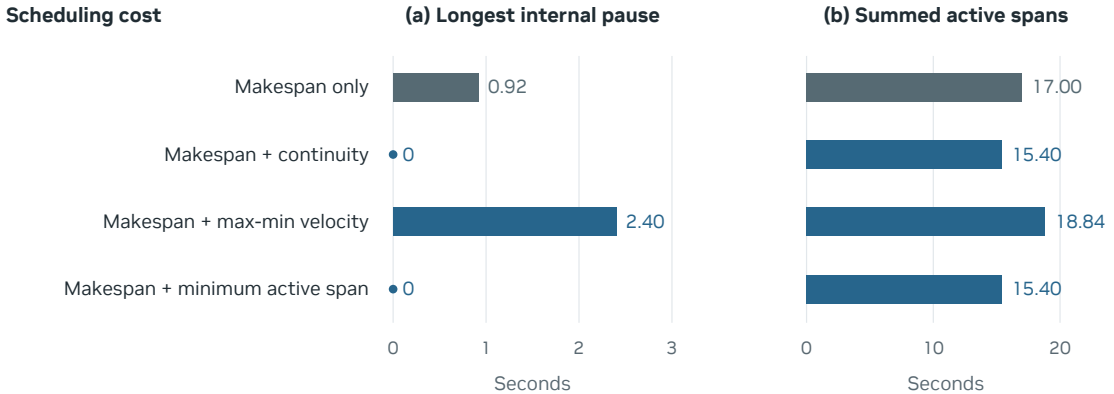


Figure 9: **Scheduling costs change motion timing at the same completion time.** Both measures use seconds; lower is better. Summed spans can exceed completion time because arms move in parallel. Pauses and spans use a 10^{-3} -rad joint-motion threshold.

C.5. Retiming Released RoboDojo Demonstrations

We use the released RoboDojo demonstrations to ask whether ScheduleGen can improve coordination when the arm paths are already available. To isolate scheduling, we keep the per-arm paths and task order fixed for 100 demonstrations per task and optimize when the arms advance along those paths. Independent motions can overlap when task and collision constraints allow it; dependent motions must still wait. No new geometric paths are planned.

The scheduler finds feasible timings for 688 of the 700 trajectories (98.3%). Here, feasibility means satisfying Eq. 1; task completion is not checked through physics replay. Table 4 compares each successfully scheduled trajectory with its original released version, so both step counts refer to the same underlying paths.

Table 4: **Retiming released RoboDojo demonstrations.** Mean step counts use the source and scheduled versions of each successfully scheduled trajectory. Paths and task order are fixed. The last column gives the mean percentage reduction per trajectory.

Task	Scheduled/ attempted	Source steps	Scheduled steps	Mean reduction (%)
Swap T	98/100	276.88	183.55	33.71
Build Tower	100/100	726.78	622.51	14.34
Insert Tubes	100/100	322.39	267.66	16.99
Arrange Largest Number	100/100	583.33	523.59	9.92
Press by Number	100/100	389.21	346.87	11.14
Sort Nesting Dolls	100/100	590.13	522.82	11.20
Classify Objects	90/100	689.62	572.87	18.31
All tasks	688/700	—	—	16.44

Retiming reduces mean step counts for every task. The released timings can already include concurrent arm motion, so these reductions show improvements over the source coordination rather than gains relative to fully serial execution.

The scheduled versions use 14.98% fewer steps in total, with a mean reduction per trajectory of 16.44%. These summarize the same paired comparisons in different ways. Let $\mathcal{J}$ contain the successfully scheduled trajectories, with source and scheduled step counts L_k and $\tilde{L}_k$ for $k \in \mathcal{J}$:

$$\begin{aligned}
 R_{\text{pooled}} &= 100 \left(1 - \frac{\sum_{k \in \mathcal{J}} \tilde{L}_k}{\sum_{k \in \mathcal{J}} L_k} \right), \\
 R_{\text{episode}} &= \frac{100}{|\mathcal{J}|} \sum_{k \in \mathcal{J}} \left(1 - \frac{\tilde{L}_k}{L_k} \right).
 \end{aligned} \tag{5}$$

The pooled reduction weights trajectories by source length, so longer demonstrations contribute more to the result. The mean per-trajectory reduction gives each demonstration equal weight, regardless of its original length. Both use the same 688 trajectories. Failed schedules count toward the scheduling success rate but are excluded from length comparisons.

In Fig. 5(a), ScheduleGen adapts source skills to sampled scenes and checks generated demonstrations through replay. Here, each released trajectory is compared with a retimed version of itself. Together, the experiments show how scheduling contributes within the generation pipeline and when applied directly to existing arm paths.

D. Policy Training

YAM visuomotor policies.

YAM policies use three RGB views (left wrist, top, and right wrist) and a 14-dimensional absolute joint and gripper state to predict 14-dimensional absolute joint and gripper targets. Table 5 lists the training settings. Scene randomization for real-world transfer is described in Section 5.4.

Table 5: **YAM visuomotor policy hyperparameters.**

Component	Setting
Image encoder	DINOv3 ViT-L/16, fine-tuned
Inputs	Three RGB views and 14-D state
Output	14-D absolute action, horizon 50
DiT	32 layers; 16 heads; head dim. 64; hidden dim. 1024; dropout 0.2
Flow noise	Beta(1.5, 1.0); $s = 0.999$; 1,000 time buckets
Inference	Four flow integration steps
Optimizer	Fused AdamW; learning rate 10^{-4} ; weight decay 10^{-5} ; max grad. norm 1
Schedule	Cosine decay; 5% warmup; 50,000 updates
Batch	32 per GPU; 8 GPUs; accumulation 1 (global 256)
Precision	bfloat16, TF32, gradient checkpointing
Checkpoints	Every 1,000 updates
Trainable modules	Vision tower, DiT, and state/action adapters; language self-attention disabled
Image augmentation	Resize shortest edge to 224; random 0.95 crop; brightness 0.3, contrast 0.4, saturation 0.5, hue 0.12
Seed	42

BC policies.

State-based BC uses a two-layer LSTM (hidden size 1,000, sequence length 10) with a five-mode Gaussian-mixture (GMM) or deterministic MSE head. Both use Adam, batch size 100, and a nominal 1,500-epoch budget. Initial learning rates are 5×10^{-4} for GMM and 10^{-3} for MSE, halved at epochs 800 and 1,200.

Image BC uses the same recurrent core with one ResNet-18 and SpatialSoftmax encoder per view, a batch size of 16, and a nominal 600-epoch training budget. GMM policies use learning rate 10^{-4} ; MSE policies use 2×10^{-4} . Learning rates are halved at epochs 400 and 500. Task-specific checkpoints are selected using held-out rollout evaluation.