

cuArms: GPU-Accelerated Conflict-Based Planning for Multi-Arm Manipulation

Anonymous Authors

Abstract—Multi-arm manipulation has the potential to dramatically improve the throughput, efficiency, and dexterity of many manipulation tasks. Recent work coordinates multiple robot arms in a shared workcell by allocating and scheduling tasks for greater parallelism, jointly planning multi-arm motions, or directly learning multi-arm policies. So far, these approaches are either too slow, rely on domain-specific assumptions, or scale poorly with the number of robots and task horizon. Thus, we introduce *cuArms*, a conflict-based planner co-designed with GPU parallelism that efficiently decomposes the search space using reusable multi-modal roadmaps for individual robots and resolves conflicts and deadlocks through a lazy search. To jointly optimize task assignment, schedules, motions, and manipulation skills, we integrate a high-level scheduler that proposes candidate skeletons for *cuArms* to ground into coordinated manipulation plans. Experimental results show that our method establishes a new state of the art, with real-time initial solutions and anytime improvements in both motion quality and scheduling, without days of domain-specific training required by learning approaches. Videos of both simulated and real-world industrial tasks are available on <https://cuarms.github.io/>.

I. INTRODUCTION

As robotic manipulator hardware becomes more widely available and less expensive, multi-arm robotic manipulation—using multiple arms to complete tasks in a shared work cell—becomes much more affordable and accessible. A multi-arm system improves task throughput with greater parallelization, increases space utilization on factory floors, and becomes necessary for tasks that require more dexterous solutions and bimanual coordination, or the handling of bulky parts.

Realizing this promise requires tightly coordinating the motions and manipulation skills of multiple robots that share the same workspace to avoid collisions. Specifically, there are several technical challenges: task allocation (which robot is assigned to each task), scheduling (the order and duration of each robot’s actions), safety (robots must not collide), and reasoning over hybrid action spaces (e.g., contacts, grasps, kinematic modality changes, reachability). Current research in this area is often domain-specific [1], [2], time-consuming [3], [4], and cannot plan bimanual manipulation skills [3], [5].

To make these increasingly difficult problems tractable, we introduce *cuArms*, a new planner that tightly couples GPU acceleration with a conflict-based multi-agent planner [6] to find long-horizon, multi-modal, and multi-arm manipulation plans. This combination attacks the computational bottleneck from both sides: algorithmic decompositions reduce the search complexity, while GPU parallelism accelerates expensive geometric validations. *cuArms* constructs a multi-modal roadmap [7] for each robot, searches the joint configuration space to resolve conflicts among robot motions and manipulation skills, and

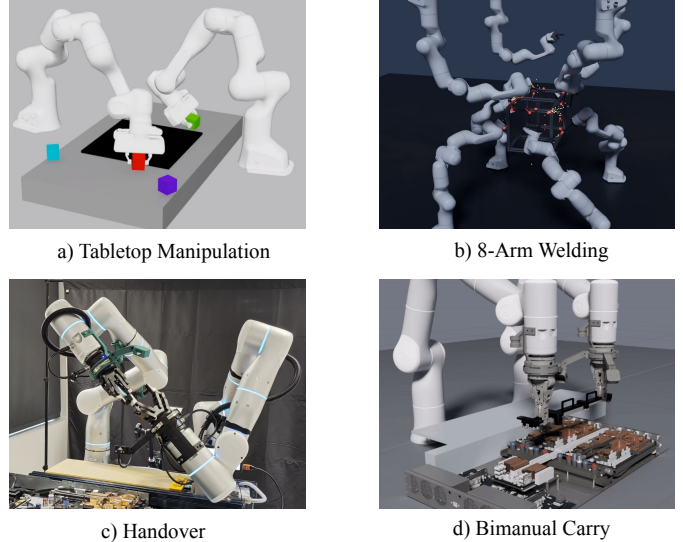


Fig. 1: Applications of *cuArms*: a) tabletop manipulation, b) welding, c) real object reorientation, and d) bimanual carry.

iteratively improves the resulting trajectory through shortcutting [8], [9]. A GPU-native implementation accelerates every stage using large batches of parallel geometric validity checks. At the same time, our algorithm reuses computation and avoids excessive CPU-GPU communication by decomposing the geometry of different robots, movable objects, and environments. Simulation and real-robot experiments show that *cuArms* finds initial plans in about 1 second across a variety of domains (e.g., Fig. 1), including an 8-arm, 40-task welding task, and improves solution quality up to 57% with additional computations. Furthermore, this efficiency makes full multi-arm task and motion planning (TAMP) tractable. As shown in Fig. 2, we use a hierarchical *schedule-then-coordinate* approach that sequentially generates many candidate schedule skeletons and plans the full multi-arm trajectory with *cuArms*.

Our contributions are:

- *cuArms*, a multi-arm manipulation planner that combines per-arm roadmaps, lazy conflict-driven joint search, and anytime trajectory refinement in a disaggregated architecture co-designed for GPU-batched computation.
- Domain-agnostic, multi-arm task and motion planning by integrating *cuArms* with a high-level scheduler with support for collaborative bimanual manipulation skills, such as handover and carry.
- Simulation and real-robot experiments for multi-arm manipulation, welding, and assembly tasks demonstrating the algorithm’s efficiency, quality, and scalability.

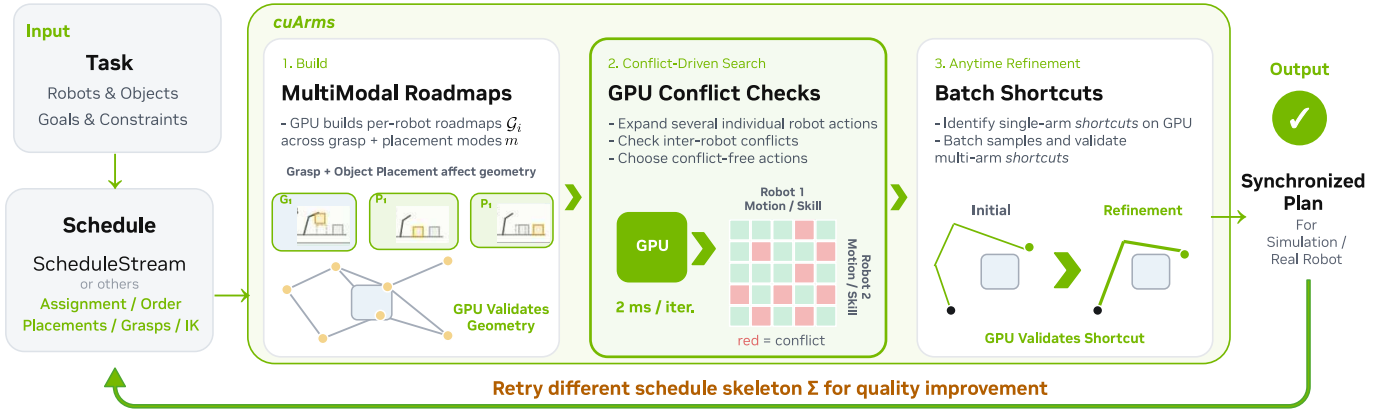


Fig. 2: System and *cuArms* overview: A human or agent defines the task: the robots and their kinematics, the objects, goals, and constraints. The domain also supplies samplers that can generate feasible manipulation choices. A task scheduler, such as ScheduleStream, proposes a candidate schedule. This includes task assignment and ordering, along with satisfying samples such as placements, grasps, and inverse kinematic solutions. *cuArms* builds a per-robot multi-modal roadmap, searches for conflict-free robot movements and skills with GPU-batched conflict tables, and refines the solution trajectory with shortcutting. *cuArms* can iteratively replan with different schedule skeletons to improve upon infeasible or poor skeletons.

II. RELATED WORK

Our work builds on prior work in multi-agent pathfinding (MAPF), multi-arm manipulation, and accelerated planning.

A. Multi-Agent Path Finding (MAPF)

MAPF algorithms [10] coordinate thousands of robots on discrete space-time graphs to find collision-free paths. Priority inheritance with backtracking (PIBT) [11] is a seminal algorithm that significantly improves speed and scalability in MAPF by resolving conflicts for single-step actions. However, PIBT is incomplete and can livelock. Lazy constraints addition search for MAPF (LaCAM) [6] embeds PIBT as a lazy expansion step in a complete depth-first search of the joint configuration space. db-LaCAM [12] extends LaCAM to kinodynamic multi-robot motion planning with motion primitives. However, LaCAM-style search has only been shown for mobile robots, as building graphs and robot collision checking for arms is much more expensive. *cuArms* uses LaCAM-style search for contact-rich multi-arm manipulation tasks with GPU-accelerated search and roadmap construction.

B. Multi-Arm Manipulation Planning

Recent work has explored how sampling-based motion planning methods [13] or MAPF techniques [3] can find efficient and collision-free motions for multi-arm systems. However, these are limited to single-query start-to-goal motion planning. Several approaches [4], [14] extend sampling-based planners to long-horizon, multi-arm pick-and-place and handover tasks. These methods are limited by collision checking as the dominant computational bottleneck.

Task and Motion Planning (TAMP) [15] offers a general model-based hybrid planning framework for long-horizon manipulation planning. ScheduleStream [16] searches over temporal, durative actions with a code-based domain description to find multi-arm schedule skeletons. However, ScheduleStream

samples conflict-free motions with rejection sampling, which is ineffective if the paths are highly intertwined. Other approaches for multi-arm TAMP also depend on postprocessing [17] or synchronization [18] to plan multi-arm motions. In contrast, *cuArms* couples individual robot roadmaps with a conflict-driven search in their composite space to find asynchronous trajectories and skills jointly. Application-driven systems exploit task structure to scale planning for multi-arm assembly [1], [2], but task-specific hierarchies and constraints make them difficult to generalize to new tasks and objectives. In contrast, *cuArms* is domain-agnostic, and a human or AI agent can define new problems without extending the planner.

C. Accelerated Motion Planning

One trend to improve motion planning is to accelerate computation with hardware such as CPU SIMD (e.g., VAMP [19]) or GPUs (e.g., cuRobo [20]). GPU kinematic solvers and planners often directly use optimized libraries such as PyTorch [20] or JAX [21], making a Python-based codebase possible for better user and agentic experiences. A similar trend appears in task and motion planning: cuTAMP [22] uses a GPU to evaluate many task skeletons in parallel. For multiple robot arms, VAMP-MR [5] is a recent multi-arm motion planner that vectorizes collision checking with CPU AVX2 instructions without algorithmic modification, but collision checking remains the computational bottleneck. To reduce this bottleneck using a GPU while avoiding excessive CPU-to-GPU communication, we design *cuArms* to evaluate collisions early and in a few large batches rather than naively moving an existing CPU-based planner onto the GPU.

III. MULTI-ARM MANIPULATION PLANNING

Robot Workspace. We assume the workspace $\mathcal{W}$ contains n robot arms $\mathcal{R} = \{r_1, \dots, r_n\}$, static obstacles, and a set of movable rigid objects $\mathcal{O}$. Each robot r_i moves in a

configuration space C_i subject to its joint position and velocity limits, and has a known base-to-world transform. Robots, grippers, and objects in the hand are modeled as unions of collision spheres [20].

Modes. A mode $m \in \mathcal{M}$ specifies the discrete grasp, attachment, or placement state of every object $o \in \mathcal{O}$ relative to a robot frame or the world frame [23]. *Modes* determine the state of all objects, and combined with the robot configuration fully define the state of the system. Only manipulation skills can change *modes*. For example, in Fig. 3, robot skills can change the object-robot attachments and thus *modes* via pick and place (Fig. 3 (a)) or pick and handover (Fig. 3 (c)). In Fig. 3 (d), the robots become kinematically coupled through the shared object during the coordinated pick-carry-place sequence.

Skills and Motions. Let $\mathcal{A}$ denote the set of manipulation skills. Each skill $a_j \in \mathcal{A}$ is a parametrized, contact-rich interaction with one or two arms. A skill a_j has a set of parameters $x_j = (q_j^{\text{in}}, q_j^{\text{out}}, d_j, \mathcal{B}_j)$, where q_j^{in} and q_j^{out} are its entry and exit joint configurations, $d_j > 0$ specifies its minimum execution duration, and $\mathcal{B}_j$ is a bounding volume enclosing the collision geometry of all participating arms and manipulated objects during execution. Each skill a_j has a known mode-transition function $m^{\text{out}} = T_{a_j}(m^{\text{in}})$ that specifies the exit mode given input mode. Each robot can also move in free space with or without an attached object. A *move* is a single-robot, collision-free motion within a fixed mode, specified by one or more target joint configurations provided directly or sampled via inverse kinematics.

Schedule Skeleton. Let a schedule skeleton $\Sigma = (\{\hat{a}_j, \mathcal{R}_j, x_j\}_{j=1}^M, \prec_\Sigma)$ define the high-level symbolic plan and be an input to our multi-arm manipulation planner *cuArms*. There are M total manipulation skills and *move* segments among all robots. Each $\hat{a}_j \in (\mathcal{A} \cup \{\text{move}\})$ is either a skill a_j or a *move* motion with participating robot set $\mathcal{R}_j$ and fixed input parameters x_j . The precedence relation $\prec_\Sigma$ imposes a total order for each robot and additional task-level constraints between actions from different robots. Skills and movements on different robots that are unordered may execute asynchronously, such as the *move* and *weld* skills in Fig. 3 (b). The skeleton fixes each skill's mode transition function T_{a_j} , and the current global mode is determined by the current skills completed.

Fig. 3 shows four skeletons. In (a), each robot alternates between *move* and single-arm *pick* and *place* skills. In (b), each arm completes a sequence of *move* and *weld* skills with no cross-robot precedence constraints. In (c), *handover* #A and *receive* #A are two single-arm skills coupled by a precedence constraint requiring R1 to hold the object until R2 receives it. In (d), all manipulation skills are bimanual, so both robots must reach their entry configurations before the bimanual *pick-carry-place* skills and depart from their exit configurations.

A. Problem Formulation

Given a multi-arm workspace $\mathcal{W}$ and a schedule skeleton Σ , the multi-arm manipulation coordination problem jointly plans collision-free move trajectories and schedules all actions. The

skeleton fixes the actions, participating robots, skill parameters, and precedence constraints. Each *move* trajectory must connect the appropriate skill exit and entry configurations and respect the mode sequence induced by the skills. A solution is a timed manipulation plan $\tau = (\{\xi_j\}_{\hat{a}_j=\text{move}}, \{[t_j, t'_j]\}_{j=1}^M)$, where ξ_j is the trajectory of a *move* action and $[t_j, t'_j]$ is the interval assigned to each *move* and skill. A plan is feasible if it is collision-free and satisfies the ordering and duration constraints of the schedule skeleton. At every time t , the geometries of moving or waiting robots and the enclosing volumes $\mathcal{B}_j$ of active skills must be collision-free with workspace obstacles and with other robots. A skill a_j is active during its scheduled interval $[t_j, t'_j]$, whose duration must satisfy $t'_j - t_j \geq d_j$. The schedule must respect precedence constraints: $\hat{a}_j \prec_\Sigma \hat{a}_k$ implies $t'_j < t_k$. Robots may wait between actions at their current configurations while maintaining their object attachments and satisfying the same collision constraints.

Cost. To support domain-specific preferences such as minimal task completion time, or more Cartesian-like movement, the objective of *cuArms* is to minimize

$$J(\tau) = w_{\text{mk}} C_{\text{mk}}(\tau) + w_{\text{soc}} C_{\text{soc}}(\tau) + w_{\text{sod}} C_{\text{sod}}(\tau),$$

where C_{mk} , C_{soc} , and C_{sod} are the makespan, sum of robot completion times, and sum of motion-distance costs. Each robot's completion time is the time it finishes all assigned *moves* and skills and thereafter remains stationary. The makespan is the maximum robot completion time. The distance cost may be measured using joint-space path length, task-space path length, volume-weighted distances traveled by collision spheres, or a weighted combination of these metrics.

IV. CUARMS ALGORITHM

cuArms disaggregates the multi-arm manipulation coordination problem into three phases: roadmap construction, joint coordination, and anytime refinement. The key insight is to separate the reusable single-robot geometry from inter-robot interactions and motion refinement, then co-design each algorithmic phase around GPU-batched geometric computations to maximize efficiency. Additionally, *cuArms* jointly coordinates skills and motions for long-horizon tasks through a conflict-based multi-agent planning method. This differs from rescheduling solutions that are generated independently [16] or sequentially [1], or using composite-space sampling and prioritized planning [4]. Table I summarizes these distinctions. Alg. 1 provides an overview of our algorithm.

A. Multi-Modal Roadmap Construction

To simultaneously search over each robot's *move* and skills, *cuArms* builds a multi-modal roadmap (Alg. 1, line 2) with both free-space motions and skill-specific edges. *cuArms* constructs a probabilistic roadmap $\mathcal{G}_i$ [7] for each robot. The robot configurations of each node v_i and the connecting edges of $\mathcal{G}_i$ must be collision-free with the static environment, and *cuArms* samples and stores valid nodes and edges in GPU memory. v_i^{init} is the initial node. Roadmaps are well suited to large GPU batches because sampling can be parallelized

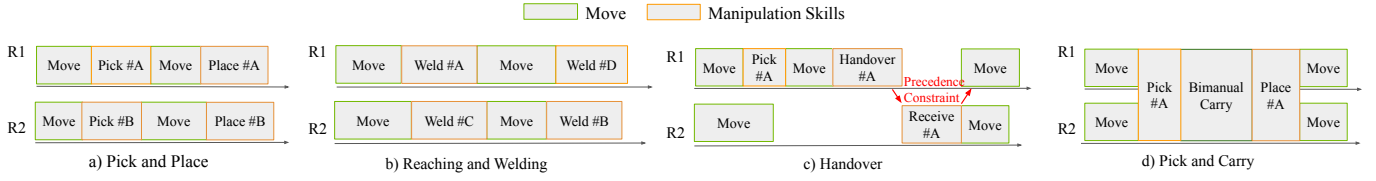


Fig. 3: Example schedule skeletons in the four domains from Fig. 1: (a) tabletop pick-and-place, (b) multi-robot reaching for welding, (c) bimanual handover, and (d) bimanual pick and coordinated carry. The high-level scheduler produces the skeleton with corresponding skill parameters such as handover poses, object placements, and grasp poses. *cuArms* then computes the coordinated multi-arm motions and skills based on the schedule skeleton.

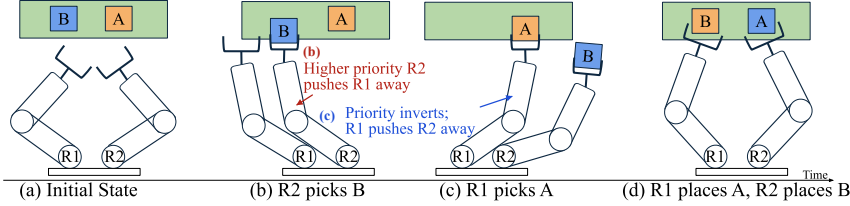


Fig. 4: A 2D toy object-swapping task. R1 must move A and R2 must move B to exchange their placements. (a) The arms initially block each other. (b) *cuArms* selects R1 to yield so R2 can pick B. (c) Priority reverses: R2 retreats while holding B so R1 can pick A. (d) Both objects reach their target placements.

and cached geometry can be reused across different schedule skeletons.

To account for movable objects $\mathcal{O}$ and modality switches in the skeleton Σ without rebuilding the roadmap, *cuArms* constructs a mode-validity mask v_i^m that marks its valid nodes and edges in each possible mode m under the skeleton Σ . The GPU batch-checks cached robot configurations against objects at their mode-specific placements, and attached objects against the environment and other objects. This decomposition reuses forward kinematic calculations and robot–environment collision results across modes, while evaluating the changing robot-object and attached-object interaction in GPU batches without rebuilding roadmaps. Intuitively, the decomposition works because objects only sparsely impact the validity of a uniformly sampled roadmap, so *cuArms* can reuse the same set of sampled nodes and edges for all modes.

cuArms then augments the roadmap $\mathcal{G}_i$ with special skill nodes and edges to connect motions and manipulation skills. For each skill’s parameter tuple x_j , *cuArms* adds the entry q_j^{in} and exit q_j^{out} configurations as nodes and connects them with an edge representing the skill execution and associated mode transitions. These entry and exit configurations are also initial and target nodes for neighboring *moves* in the skeleton Σ . The skill’s bounding volume $\mathcal{B}_j$ encloses the skill edge for batched collision checking. Execution durations are rounded up to a whole-number multiple of the max edge interval to synchronize the skill execution with normal roadmap motions. Thus, planning skill execution and connecting motions becomes searching for valid edge transitions on $\mathcal{G}_i$.

To inform the conflict-driven search, *cuArms* computes a heuristic cost-to-go $h_i[v, \hat{a}]$ for each node on the roadmap and current action (Alg. 1, line 3). Using unit costs for

TABLE I: Comparison of multi-arm motion or manipulation planning approaches.

Method	Parallelization	Skills	Conflict-based
xECBS [3]	✗	✗	✓
Hartmann et al. [4]	✗	✓	✗
APEX-MR [1]	✗	✓	✗
VAMP-MR [5]	CPU SIMD	✗	✓
ScheduleStream [16]	GPU	✓	✗
cuArms	GPU	✓	✓

each motion and skill edge, *cuArms* runs Dijkstra’s algorithm backward from each action’s target set on the base roadmap without the mode validity mask, then adds an estimate of the motion and skill costs remaining after reaching a target. When known precedence constraints from the skeleton Σ block a robot’s next *move* or skill, *cuArms* can instead guide motions to a conflict-free home configuration. For this purpose, *cuArms* computes a separate cost-to-home heuristic from the home node.

B. GPU-Accelerated Conflict-Driven Search

Given the per-robot roadmaps, *cuArms* builds on LaCAM [6] to plan multi-arm, multi-task, and multi-modal motion and manipulation skills. The original LaCAM is a depth-first search algorithm that lazily expands the successor joint state guided by the single-step inter-robot conflict resolution algorithm PIBT [11]. Fig. 4 illustrates this on a toy example, where R1 must first yield for R2’s pick, then R2 retreats with B to make room for R1. Planning motion independently cannot resolve this deadlock via retiming alone, while sequential planning must explicitly order the inter-robot actions in the schedule skeleton. However, the original LaCAM does not track task-level progress for long-horizon manipulation skills. Resolving conflicts for arms with PIBT is also slow because the recursive priority inheritance repeatedly invokes expensive inter-robot collision checks.

Compared to LaCAM, *cuArms* augments the roadmap node v_i with its current symbolic action $\hat{a}$, and each search node s stores the joint roadmap and action states $\langle v_1, \hat{a}_1, \dots, v_{|R|}, \hat{a}_{|R|} \rangle$ (line 4). A constraint queue $s.K$ stores the action-constraint set κ that forces certain action selection in PIBT, allowing LaCAM to lazily explore alternative joint

successors without enumerating all action combinations upfront (line 16). The joint state determines the current manipulation mode and whether the schedule skeleton allows each robot’s next action. At each state, *cuArms* expands a fixed list $\mathcal{U}_i(s)$ of candidate motions, waits, and manipulation skills for each robot (line 13). Skills become available when all participating robots reach their entry nodes and have no blocking precedence constraints. Completing a skill advances the robots to their exit nodes, advances action stages, and applies any mode transitions. *cuArms* sorts the candidate list $\mathcal{U}_i(s)$ by first prioritizing manipulation skills, then actions that lead to low cost-to-go at the resulting state s'_i (line 14).

The key innovation that accelerates LaCAM is to compute the geometric conflicts upfront and in parallel on the GPU (line 15), before the recursive PIBT action selection (line 17). In the original PIBT, robots move on a shared graph, and each candidate next vertex has at most one current robot occupant. PIBT processes robots in descending order of dynamic priority so that higher-priority robots choose actions first, and each robot selects its next vertex based on the cost-to-goal heuristic while avoiding higher-priority robots. If a robot requests a vertex occupied by another robot whose next action is undecided, the occupant temporarily inherits its priority and is recursively asked to move; if it cannot, PIBT backtracks and tries another candidate. For robot arms, a candidate edge can conflict with several arms simultaneously, and the recursive geometry check to find conflicts with other arms is expensive. Running these recursive checks on the GPU requires repeated kernel launches and synchronization overhead, which limits GPU parallelism. Instead, *cuArms* gathers the GPU-resident collision geometry for all candidate edges and batch-checks all inter-robot candidate pairs, using aligned robot trajectories for motions and enclosing volumes for skills. It then runs PIBT by looking up the resulting Boolean conflict table during priority inheritance and backtracking. This design separates the recursive CPU control flow from batched GPU computation and avoids repeated GPU launches and CPU-GPU synchronization.

cuArms uses PIBT to generate conflict-free joint successors within the lazily constrained LaCAM search. For bimanual skills, *cuArms* commits and constrains participating robots’ actions in PIBT and LaCAM together. *cuArms* can also exploit precedence constraints from the skeleton Σ if a *move* is blocked by directly prioritizing yield motions with cost-to-home heuristics in candidate ranking. The search terminates when every robot completes its action sequence, and the solution path is converted into the timed plan τ^* .

C. Anytime Refinement

Because initial solutions found by *cuArms* can be suboptimal due to coarse roadmaps and the lack of an optimality guarantee in LaCAM-style search, *cuArms* refines solutions by identifying shortcuts for *move* trajectories in batches on the GPU. Recent work on multi-arm shortcuts [9] has proposed how multi-arm shortcuts can improve trajectory quality, but it is nontrivial how shortcuts can improve long-horizon manipulation plans where robots execute both motions and skills

Algorithm 1 *cuArms*: Multi-Arm Manipulation Planning

Require: Schedule skeleton Σ , workspace $\mathcal{W}$
 $\triangleright$ Green denotes GPU-accelerated steps.

```

1: for all robots  $i \in \mathcal{R}$  do
2:    $\mathcal{G}_i \leftarrow \text{MULTIMODALROADMAP}(i, \Sigma, \mathcal{W})$   $\triangleright$  Sec. IV-A
3:    $h_i \leftarrow \text{HEURISTICS}(\mathcal{G}_i, \Sigma)$   $\triangleright$  Backward Dijkstra
4:    $s^{\text{init}} \leftarrow \langle v_1^{\text{init}}, a_1^{\text{init}}, \dots, v_{|R|}^{\text{init}}, a_{|R|}^{\text{init}} \rangle$   $s^{\text{init}}.K \leftarrow [\emptyset]$ ;  $s^{\text{init}}.\text{parent} \leftarrow \perp$ 
5:    $S \leftarrow [s^{\text{init}}]$ ;  $\tau \leftarrow \perp$   $\triangleright$  Initialize DFS stack and solution
6:   while  $S \neq []$  do
7:      $s \leftarrow \text{TOP}(S)$ 
8:     if all actions in  $\Sigma$  are complete at  $s$  then  $\triangleright$  Solution found
9:        $\tau \leftarrow \text{BACKTRACK}(s)$ ; break
10:    if  $s.K = []$  then  $\triangleright$  Constraint queue exhausted
11:       $\text{POP}(S)$ ; continue  $\triangleright$  Skip  $s$ 
12:    for all robots  $i \in \mathcal{R}$  do
13:       $\mathcal{U}_i(s) \leftarrow \text{EXPAND}(s, \mathcal{G}_i)$   $\triangleright$  Expand skills, motions, waits
14:      Sort  $\mathcal{U}_i(s)$  by  $\begin{cases} (0, h_i[s'_i]) & \text{skill,} \\ (1, h_i[s'_i]) & \text{motion or wait.} \end{cases}$ 
15:       $C_s \leftarrow \text{GPUCONFLICTTABLE}(\{\mathcal{U}_i(s)\}, \{\mathcal{G}_i\})$   $\triangleright$  Sec. IV-B
16:       $\kappa \leftarrow \text{POP}(s.K)$ ; enqueue new constraints extending  $\kappa$  into  $s.K$ 
17:       $s' \leftarrow \text{PIBT}(s, \{\mathcal{U}_i(s)\}, C_s, \kappa)$   $\triangleright$  Conflict-resolution
18:      if  $s' \neq \perp$  and  $s'$  is new then
19:         $s'.K \leftarrow [\emptyset]$ ;  $s'.\text{parent} \leftarrow s$ ;  $\text{PUSH}(S, s')$ 
20:    if  $\tau = \perp$  then
21:      return  $\perp$ 
22: return  $\text{BATCHEDREFINEMENT}(\tau, \Sigma, \mathcal{W})$   $\triangleright$  Sec. IV-C
```

asynchronously. This is because when post-processing multi-arm plans, optimizing one arm’s motion must not introduce collisions with the other robot’s motion or skills.

cuArms improves its initial plan through batched multi-arm shortcutting co-designed to maximize GPU parallelism. First, for each *move* trajectory, *cuArms* enumerates a bounded set of straight-line interpolations in joint space as shortcuts and batch-evaluates whether these shortcuts are collision-free with the static environment and movable objects $\mathcal{O}$ under its current mode. From all valid single-robot shortcuts, *cuArms* randomly samples a large batch of multi-arm shortcuts. A multi-arm shortcut can be a single-arm path or prioritized shortcut, where all other arms’ trajectories remain the same [9] or a composite shortcut made of at least two concurrent single-arm shortcuts. *cuArms* then evaluates whether these multi-arm shortcuts have any inter-robot conflicts in parallel. After finding a set of valid multi-arm shortcuts, *cuArms* greedily combines non-overlapping shortcuts in descending order of cost improvement, replacing the original robot trajectory.

V. MULTI-ARM TASK AND MOTION PLANNING

The full multi-arm TAMP problem requires deciding which robots perform which tasks, in what order, and with which skill parameters, in addition to solving the multi-arm manipulation planning problem. Joint optimization of the schedule skeleton and coordinated manipulation plan can improve upon using the high-level scheduler or the manipulation planner alone. A high-level scheduler or temporal planner often solves task allocation and scheduling and evaluates manipulation modes—such as object placements and grasps—effectively, but is less effective at finding tightly coordinated motions. Optimizing for motions under a fixed schedule skeleton cannot correct poor

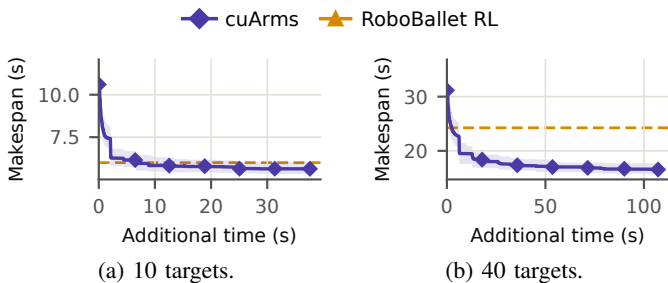


Fig. 5: **RoboBallet schedule search and refinement.** Execution makespan improves as *cuArms* shortcuts initial trajectories and searches alternative schedules.

task assignments or skill parameters that force unnecessary yielding and waiting.

We introduce a hierarchical *schedule-then-coordinate* approach in which a task-level scheduler generates candidate schedule skeletons while *cuArms* grounds them into tightly coordinated motion and manipulation trajectories that allow asynchronous execution. The scheduler may be a general temporal planner such as ScheduleStream [16], a domain-specific optimization program, or derived from a fixed-order assembly plan. If *cuArms* fails to find a feasible plan for any candidate skeletons, the system backtracks to the scheduler to generate additional skeletons. In practice, the scheduler can generate a batch of candidate skeletons upfront, which reduces the need for backtracking. *cuArms* also reuses the same multi-modal roadmap stored in GPU memory when the modes are shared; when new modes are introduced, it updates only the mode-validity masks without modifying the base roadmap. The resulting TAMP hierarchy evaluates many candidate schedule skeletons to improve from an initially infeasible or suboptimal manipulation plan.

VI. EXPERIMENTS

To evaluate *cuArms*, we examine three questions: 1) How quickly does it find an initial solution? 2) How does it improve solution quality with additional planning time? 3) How does it scale with the number of tasks and robots?

Environments and Baselines. We evaluate in two simulated benchmarks as well as real-world bimanual electronic assembly tasks. We first evaluate *cuArms* in a welding task proposed by RoboBallet [24], a state-of-the-art reinforcement learning (RL)-based method. We retrain the RL policies from scratch in about two days. Next, we analyze the performance of *cuArms* for TAMP problems in a set of tabletop manipulation tasks against ScheduleStream [16], a general temporal planner for multi-arm TAMP and scheduling with GPU acceleration. Lastly, we show how *cuArms* can be deployed in real-world bimanual electronics assembly tasks with integrated manipulation skill sampling and coordinated manipulation planning.

Metrics. For each set of experiments, we evaluate the time to initial solutions t_0 and initial and final makespan, denoted by J_0 and J_f , respectively. Unless stated otherwise, each simulation setting uses 15 random seeds; plots show the

TABLE II: **RoboBallet scheduling and planning.** t_{offline} is the median base roadmap construction time for *cuArms* and training time for RL. t_0 is the total inference time for RL.

Targets	Robots	Method	Solved	$t_{\text{offline}}(s)$	$t_0(s)$	$J_0(s)$	$J_f(s)$
10	4	<i>cuArms</i>	150/150	0.81	0.20	10.61	5.64
40	4	<i>cuArms</i>	150/150	0.81	0.41	31.16	16.56
40	8	<i>cuArms</i>	150/150	1.74	0.87	25.35	14.87
10	4	RL	29/150	1.6×10^5	0.11	–	6.00
40	4	RL	50/150	2.2×10^5	0.42	–	24.25

median and interquartile range (IQR). We use an RTX Pro 6000 GPU workstation to measure all results.

A. RoboBallet: Many-Arm and Long-Horizon Manipulation

RoboBallet [24] is an RL method for scaling the number of arms and tasks for multi-arm scheduling and motion planning. The evaluation environment requires 4 or 8 arms performing 10 or 40 spot welding tasks in a shared, tight workspace with obstacles, with the goal of reaching a set of Cartesian weld targets (Fig. 1 (b)). RoboBallet learns a single policy in simulation that jointly allocates and schedules tasks and generates coordinated motions. RoboBallet provides neither completeness nor formal safety guarantees and also assumes goal-reaching tolerances up to 2.5 cm and 15° .

Our system combines a greedy scheduler with *cuArms* to schedule and plan the full multi-arm reaching task and motions. The greedy scheduler repeatedly uses Hungarian matching to assign a target to each robot every round using relaxed travel-distance costs until all targets are assigned. The joint target configurations for each Cartesian weld target are sampled in batches using GPU-based inverse kinematic solvers [20] to achieve sub-millimeter-level precision. *cuArms* then plans the manipulation trajectory from the given schedule skeleton and refines the initial solutions for 2s and 6s per skeleton for 10- and 40- target problems respectively. As only the weld targets are different in each evaluation episode while the workspace remains the same, *cuArms* precomputes a base roadmap without any episode-specific joint objectives and reuses it. During planning, *cuArms* first connects each target to the base roadmap using RRT-guided steering. Once the system solves the initial skeleton, it backtracks and proposes alternative schedule skeletons with sampled targets for *cuArms* to solve iteratively. A total of 16 schedule skeletons are evaluated.

Results. *cuArms* finds solutions in real time, with a 0.2s median time-to-initial-solution for 10 targets and 0.41s for 40 targets, each with 10 episodes (Table II). The base roadmap can be precomputed offline in less than 0.22s per robot. Shortcutting and evaluating alternative schedules reduce the initial makespan by about 47% in both settings (Fig. 5). The retrained RoboBallet RL policy completes only 19.3% of 10-target episodes and 33.3% of 40-target episodes, as its predicted actions sometimes fail to avoid collisions and end up in deadlocks, or fail to reach the goal within the 2.5 cm and 15° tolerance. On episodes successfully solved by both methods, *cuArms* achieves median makespans that are

TABLE III: *cuArms* median time-to-initial solution (ms) breakdown in RoboBallet. Roadmap construction refers to connecting new weld targets to the base roadmap. Collision checking includes queries in roadmap construction and the conflict table.

Targets	4 arms		8 arms
	10	40	40
Greedy Scheduler	0.59	5.12	3.06
Roadmap Construction	98.70	155.39	271.17
Heuristics	47.68	120.81	146.44
Candidate Expansion & Sort	8.24	24.24	32.06
Conflict Table	22.46	65.94	277.46
LaCAM search & PIBT	6.81	13.42	19.51
Collision Checking (% of total)	18.1%	24.7%	50.2%

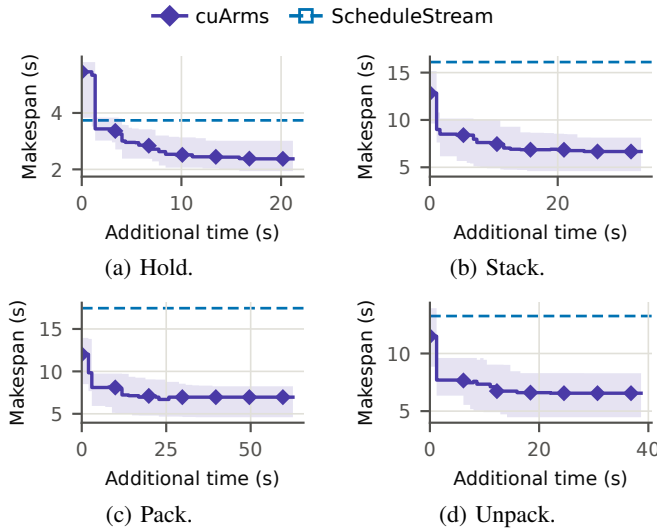


Fig. 6: **Tabletop manipulation refinement.** *cuArms* improves plans across skeletons and manipulation modes. Makespan can increase while the weighted optimization objective improves.

20.1% and 27.9% lower than those of RL in the 10-target and 40-target settings, respectively. When doubling the number of arms to 8, *cuArms* can reduce the median makespan by 10.2% while still finding feasible solutions within 1 second. As shown in Table III, GPU can significantly reduce the collision-checking bottleneck to 18.1-24.7% of time-to-initial solution for 4 arms and 50.2% for 8 arms. Another major computational bottleneck is constructing the full roadmap that connects the weld targets, which may require more iterative sampling and steering steps towards the base roadmap since the environment obstacles can impose tight corridor-like constraints.

B. Multi-Arm TAMP

We evaluate our full multi-robot TAMP system for tabletop manipulation tasks such as holding, stacking, packing, and unpacking 2-4 cubes with two Panda arms, as shown in Fig. 1 (a). Packing and unpacking additionally evaluate 6 and 8 objects. Both *cuArms* and the ScheduleStream baseline use ScheduleStream’s symbolic planner to search over sched-

TABLE IV: **Multi-arm TAMP for tabletop manipulation** with two to four objects. J_0 and J_f are the initial and final makespan, respectively.

Task	<i>cuArms</i>				ScheduleStream		
	Solved	t_0	J_0	J_f	Solved	t_0	J_0
Hold	45/45	0.48	5.46	2.37	44/45	1.75	3.74
Stack	45/45	2.26	12.84	6.67	38/45	4.02	16.11
Pack	45/45	2.96	12.02	6.96	38/45	4.54	17.45
Unpack	45/45	1.60	11.50	6.55	39/45	3.30	13.25

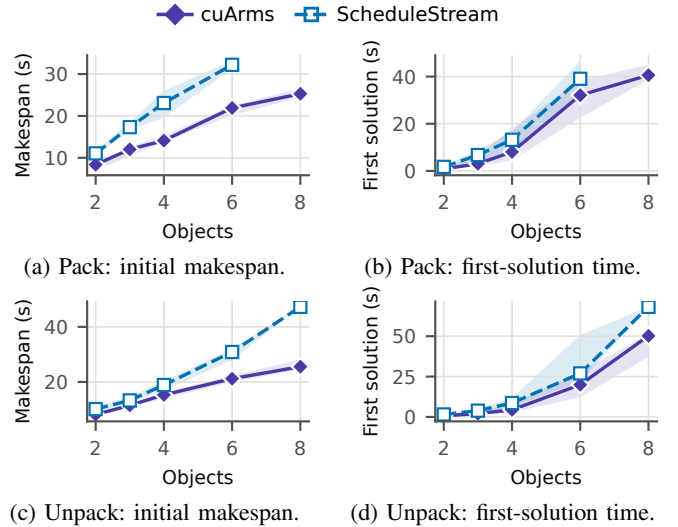


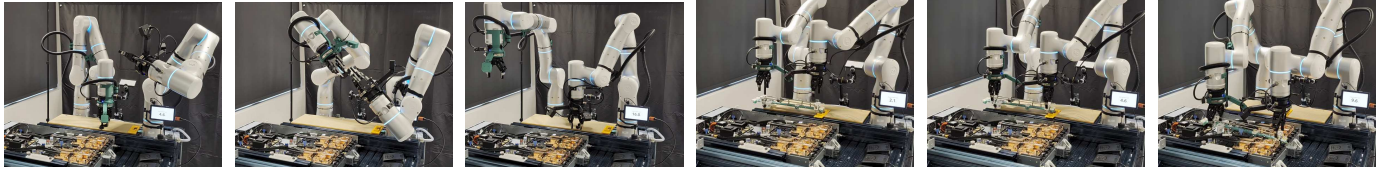
Fig. 7: **Scaling with number of objects.** *cuArms* finds more efficient manipulation plans in less time than ScheduleStream, which also fails to find any solutions for 8-object packing.

ule skeletons and sample grasps, placements, and inverse-kinematics solutions with a total high-level search budget of 60s. However, the ScheduleStream baseline plans each arm’s path independently and then retimes the trajectories to avoid conflicts, whereas *cuArms* jointly coordinates the manipulation skills and *move* trajectories. *cuArms* explores up to 16 schedule skeletons with different sampled parameters and refines initial trajectories for 2s per skeleton in Pack and 1s per skeleton in the other tasks. *cuArms* optimizes the weighted objective with both makespan and path-distance costs.

Results. As shown in Table IV, *cuArms* finds initial solutions in 0.48-2.96 s, 46.4% faster than ScheduleStream, with a better initial makespan except for the Hold task. *cuArms* also solves all instances, while ScheduleStream can fail on some instances requiring tight coordination. Multi-arm shortcutting and exploration of alternative candidates can efficiently reduce the median *cuArms* makespan by 42–57% across the four tasks. (Fig. 6). Fig. 7 further demonstrates that *cuArms* can scale up to 8 objects, whereas ScheduleStream finds fewer feasible solutions and higher-makespan plans.

C. Real-Robot Electronics Part Assembly

Finally, we evaluate the full multi-robot TAMP on two real-world electronics assembly tasks with two Flexiv Rizon4s arms, as shown in Fig. 8. The system uses a GPU-



(a) **Handover.** Re-orient a component in hand using an automatically sampled and optimized handover pose.

(b) **Bimanual carry.** Collision-free approach, coordinated grasp, and transport of the busbar fixture into the server tray.

Fig. 8: Bimanual Electronic Assembly Tasks with *cuArms*.

accelerated skill sampler to sample constrained poses instead of hardcoded handover poses or carry trajectories and generates schedule skeletons with a ScheduleStream-based symbolic planner. *cuArms* produces a coordinated manipulation plan with collision-free approach motions around the camera mounts. Videos of successful execution are available in our supplementary material and <https://cuarms.github.io/>.

VII. CONCLUSION

We present *cuArms*, a new conflict-based multi-arm manipulation planning algorithm. Using a GPU to batch-evaluate geometry, *cuArms* efficiently searches the joint state space with decomposed and reusable per-arm roadmaps and inter-robot conflict resolution. *cuArms* can be integrated into a general multi-arm TAMP algorithm with a *schedule-then-coordinate* approach to improve both low-level coordinated manipulation and high-level scheduling. With solutions found near-instantaneously without requiring task-specific training, users can use *cuArms* to rapidly evaluate the feasibility of multi-arm task and workspace configurations, and integrate it with domain-specific costs and collaborative skills.¹

REFERENCES

- [1] P. Huang, R. Liu, S. Aggarwal, C. Liu, and J. Li, "APEX-MR: Multi-robot asynchronous planning and execution for cooperative assembly," in *Proceedings of Robotics: Science and Systems*, 2025.
- [2] Y. Tian, J. Jacob, Y. Huang, J. Zhao, E. L. Gu, P. Ma, A. Zhang, F. Javid, B. Romero, S. Chitta, S. Sueda, H. Li, and W. Matusik, "Fabrica: Dual-arm assembly of general multi-part objects via integrated planning and learning," in *Conference on Robot Learning*, 2025, pp. 406–419.
- [3] Y. Shaoul, I. Mishani, M. Likhachev, and J. Li, "Accelerating search-based planning for multi-robot manipulation by leveraging online-generated experiences," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2024, pp. 523–531.
- [4] V. N. Hartmann, T. Heinle, Y. Huang, and S. Coros, "Sampling-based multi-modal multi-robot multi-goal path planning," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2026.
- [5] P. Huang, C. Gao, and J. Li, "VAMP-MR: Vector-accelerated motion planning and execution for multi-robot-arms," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2026.
- [6] K. Okumura, "LaCAM: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 11 655–11 662.
- [7] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [8] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.
- [9] P. Huang, Y. Shaoul, and J. Li, "Benchmarking shortcutting techniques for multi-robot-arm motion planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2025, pp. 13 258–13 265.
- [10] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, 2019, pp. 151–158.
- [11] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2019, pp. 535–542.
- [12] A. Moldagalieva, K. Okumura, A. Prorok, and W. Hönig, "db-LaCAM: Fast and scalable multi-robot kinodynamic motion planning with discontinuity-bounded search and lightweight MAPF," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2026, pp. 738–746.
- [13] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3–4, pp. 443–467, 2020.
- [14] R. Shome and K. E. Bekris, "Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs," in *Proceedings of the IEEE International Symposium on Multi-Robot and Multi-Agent Systems*, 2019, pp. 37–43.
- [15] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 265–293, 2021.
- [16] C. Garrett and F. Ramos, "ScheduleStream: Temporal planning with samplers for GPU-accelerated multi-arm task and motion planning & scheduling," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2026.
- [17] V. N. Hartmann and M. Toussaint, "Towards computing low-makespan solutions for multi-arm multi-task planning problems," *arXiv preprint arXiv:2305.17527*, 2023.
- [18] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "A general task and motion planning framework for multiple manipulators," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3168–3174.
- [19] W. Thomason, Z. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2024, pp. 8749–8756.
- [20] B. Sundaralingam, A. Murali, and S. Birchfield, "cuRoboV2: Dynamics-aware motion generation with depth-fused distance fields for High-DoF robots," *arXiv preprint arXiv:2603.05493*, 2026.
- [21] L. Chen, S. R. Iyer, and Z. Kingston, "Differentiable particle optimization for fast sequential manipulation," *arXiv preprint arXiv:2510.07674*, 2025.
- [22] W. Shen, C. R. Garrett, N. Kumar, A. Goyal, T. Hermans, L. P. Kaelbling, T. Lozano-Pérez, and F. Ramos, "Differentiable GPU-parallelized task and motion planning," in *Proceedings of Robotics: Science and Systems*, 2025.
- [23] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.
- [24] M. Lai, K. Go, Z. Li, T. Kröger, S. Schaal, K. R. Allen, and J. Scholz, "RoboBallet: Planning for multirobot reaching with graph neural networks and reinforcement learning," *Science Robotics*, vol. 10, no. 106, p. eads1204, 2025.

¹We used OpenAI Codex and Anthropic Claude to assist with code optimization, monitoring experiments, creating plots, and proofreading.