ROBOMONSTER: CAPABILITY-AWARE SELECTION AND COORDINATION OF HETEROGENEOUS END-EFFECTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite rapid progress in robot hardware and algorithms, a persistent gap remains between flexible decision-making in simulation and the embodiment constraints of real robots, often leading to suboptimal execution on deceptively simple tasks. We posit that, rather than emulating human morphology, robots should *compose* heterogeneous embodied agents whose capabilities extend beyond human-like end-effectors. We introduce *RoboMonster*, a paradigm and system that reasons over and coordinates multiple, diverse agents. At the planning level, an MLLM-based Planner performs chain-of-thought selection over a Robot Manual describing each agent's skills and limits, and an LLM-based Verifier checks feasibility and efficiency; we benchmark this process with *RoboMonster-P*. At the execution level, we implement interaction logic for four end-effector types in ManiSkill, collect data, train policies under both classical imitation-learning backbones and a generalist Vision-Language-Action model, and evaluate on *RoboMonster-E*. We further run an end-to-end Planner→Execute study that feeds the Planner's actual choices into the trained policies, and validate the system on a real ALOHA platform against single-effector baselines. Experiments show that heterogeneous configurations improve end-to-end success from 37.8% to 76.4% over gripper-only and increase real-robot success from 4/60 to 40/60 trials. Execution policies are trained for each task and configuration; the results support capability-driven selection and coordinated execution on the evaluated tasks, without assuming zero-shot control of unseen effector combinations.

1 INTRODUCTION

Robot hardware and learning algorithms have advanced rapidly, with diverse embodiments (humanoid hands, quadrupeds) and capable closed-loop frameworks (Fang et al., 2023; Chi et al., 2023; Zhao et al., 2023; Brohan et al., 2023; Black et al., 2024; Kim et al., 2024). Yet a persistent gap remains between flexible decision-making in simulation and the embodiment constraints of real robots: capable algorithms are forced through a single, pre-configured end-effector that defeats them on deceptively simple tasks. Picking up a credit card lying flat on a table is a familiar example: humans solve it with a subtle skill, yet single-gripper or dexterous-hand policies, even with tactile sensors, often fail. This exposes a mismatch between algorithmic potential and the fixed hardware that algorithms must act through.

Two common responses fall short. Continuously upgrading hardware toward an ever more general end-effector is costly in design, manufacturing, and data, and every new effector requires retraining the corresponding policy. Adding more arms of the *same* kind (dual-arm or multi-arm) helps with payload and gross coordination but does not resolve the underlying single-effector bottleneck.

We therefore pose a different question: **do we need to design robots to resemble humans, or can we build heterogeneous systems whose capabilities extend beyond a single embodiment?** A suction cup lifts a card with a simple mechanism, an annular gripper cages a vase or smooth sphere, and a thin stick reaches into narrow cavities. As illustrated in Fig. 1, we introduce *RoboMonster*, a paradigm in which a robot reasons over a library of heterogeneous end-effectors, selects an optimal composition for the current task, and dispatches sub-tasks to specialised agents that collaborate, so that the union extends the task envelope reachable by any single embodiment.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

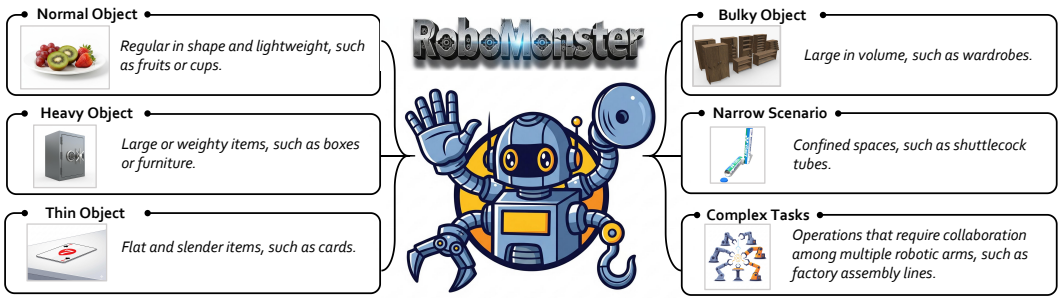

Figure 1: **Motivation for capability-aware embodiment.** Object and scene properties demand different physical affordances; a fixed gripper may struggle even with a thin card. RoboMonster selects and coordinates complementary end-effectors. Heavy and bulky objects illustrate the broader motivation, while our evaluation focuses on the five defined manipulation tasks.

We instantiate *RoboMonster* with an MLLM-based **Planner** and an LLM-based **Verifier** that consult a structured Robot Manual encoding per-effector skills, action ranges, and execution costs, and benchmark it on *RoboMonster-P*. At the execution stage, we instantiate four end-effectors (gripper, suction, stick, ring-shaped) in ManiSkill (Gu et al., 2023), collect demonstrations via an MLLM-driven pipeline, and train policies under two imitation-learning backbones (DP, DP3) and the generalist Vision-Language-Action (VLA) model $\pi_{0.5}$, benchmarking on *RoboMonster-E*. We further evaluate the **full pipeline end-to-end**, feeding the Planner's actual choices into the trained policies rather than always assuming the oracle-optimal effector, and validate on a real ALOHA platform against single-effector baselines.

Our main contributions are:

- *Concept & paradigm.* A robotics paradigm that combines heterogeneous embodied agents to connect capability-aware selection with coordinated physical execution.
- *Planning verification.* A simple Planner+Verifier system for selecting heterogeneous embodied agents, benchmarked on *RoboMonster-P*.
- *Execution verification.* Interaction logic for four end-effector types, the *RoboMonster-E* benchmark, and policies trained under DP, DP3, and $\pi_{0.5}$ that demonstrate the execution advantages of heterogeneous composition.
- *End-to-end and real-robot evaluation.* The full Planner→Execute pipeline rather than only oracle-optimal configurations, and a real ALOHA study against gripper-only configurations.

2 Related Work

Embodied multi-agent cooperation. Real-world embodied environments often demand collaboration among heterogeneous robots. Prior work studies this through task allocation (Obata et al., 2024; Wang et al., 2024b; Liu et al., 2025) and high-level multi-agent decision making (Zhang et al., 2023; Wang et al., 2025a), with LLMs improving distributed planning and communication (Bo et al., 2024a; Guo et al., 2024b; Nasiriany et al., 2024; Zhou et al., 2023), and VLMs extending these capabilities to embodied multi-agent contexts (Wang et al., 2025b; Zhang et al., 2024). Most approaches assume homogeneous capabilities or treat each agent independently, without integrating complementary skills across heterogeneous agents. VIKI (Kang et al., 2025) addresses heterogeneous robots but only at the high-level planning stage. In contrast, we enable collaborative control across diverse embodiments, jointly orchestrating heterogeneous end-effectors for tasks beyond the reach of a single agent type.

Robot learning in manipulation. Task-specific policy architectures (Chi et al., 2023; Ke et al., 2024; Liang et al., 2023; 2024; 2025; Wang et al., 2024a; Wen et al., 2025; Ze et al., 2024) achieve strong results in controlled settings but are tightly coupled to particular tasks or morphologies. Large-scale foundation models trained on diverse multi-robot datasets, including RT-1 (Brohan et al., 2022), RT-2 (Brohan et al., 2023), RDT-1B (Liu et al., 2024), π_0 (Black et al., 2024), OpenVLA (Kim et al., 2024), Octo (Octo Model Team et al., 2024), LAPA (Ye et al., 2024), and OpenVLA-OFT (Kim et al., 2025), show more promising generalization across tasks but typically presuppose homogeneous

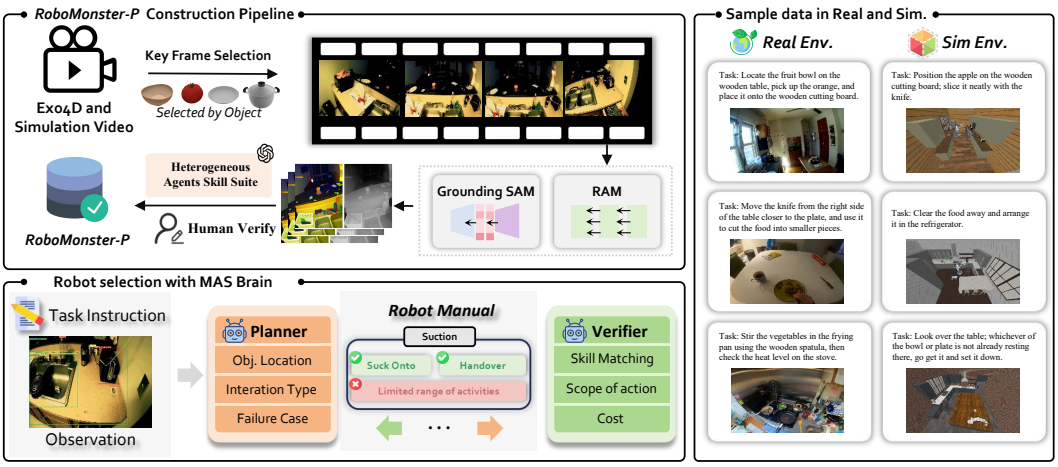

Figure 2: **Planning for Compositional Heterogeneous Embodied Agents.** (Left Top) Data collection and curation process of *RoboMonster-P*. (Left Bottom) Construction of the multi-agent system, which analyzes input images and instructions to select appropriate heterogeneous embodied agents. (Right) *RoboMonster-P* includes a diverse set of real-world and simulated environments and tasks.

agents and uniform capabilities. We explicitly organise heterogeneous embodiments into a coherent control framework that leverages their complementary skills.

Multi-agent systems for robot planning. LLM-based multi-agent systems (MAS) provide infrastructure for role-specialised collaboration and tool use (Li et al., 2023; Wu et al., 2023; Chen et al., 2023; Hong et al., 2024; Qian et al., 2024), with a growing line of work treating *planning* itself as a multi-agent process (sub-plan decomposition, expert coordination, reflection) (Guo et al., 2024a; Wei et al., 2025; Li et al., 2025; Tao et al., 2024b; Bo et al., 2024b). Robotics-oriented instantiations include RoCo (Mandi et al., 2024), MALMM (Singh et al., 2024), and SMART-LLM (SMART Lab, 2023). We instantiate an MAS specifically to plan over heterogeneous end-effectors, enabling coordinated assignment across embodiments.

Positioning relative to tool use and geometric planning. Selecting an effector shares the capability-matching problem of physical tool use. Our formulation connects that selection to separately trained executors and evaluates the resulting pipeline on the same manipulation tasks as the gripper-only baseline. The Robot Manual describes skills, ranges, and costs, but is not a geometric scene model: the Verifier checks represented constraints rather than constructing collision-free motions. Low-level actions come from imitation policies. This distinction separates semantic selection, coordinated execution, and geometric feasibility, and prevents improvements in one from being interpreted as evidence for all three. Further comparisons are discussed in Appendix E.

3 RoboMonster

The motivation in Fig. 1 is that object geometry and interaction requirements make different physical capabilities useful. We instantiate this idea with four effectors targeting ordinary grasping, thin-object lifting, caging, and narrow-space insertion. Complementary effectors also support cooperative manipulation, such as lifting and handing over a card for insertion. The broader task spectrum is discussed in Appendix F.1.

Scope of capability composition. The high-level operation selects effectors; physical composition occurs when complementary effectors take different roles within a joint executor. We evaluate selection on held-out image–instruction pairs and execution across unseen initial poses of the defined tasks. Execution policies are trained separately for each task and configuration. These evaluations establish performance within that task library, rather than zero-shot control of unseen tasks or effector combinations.

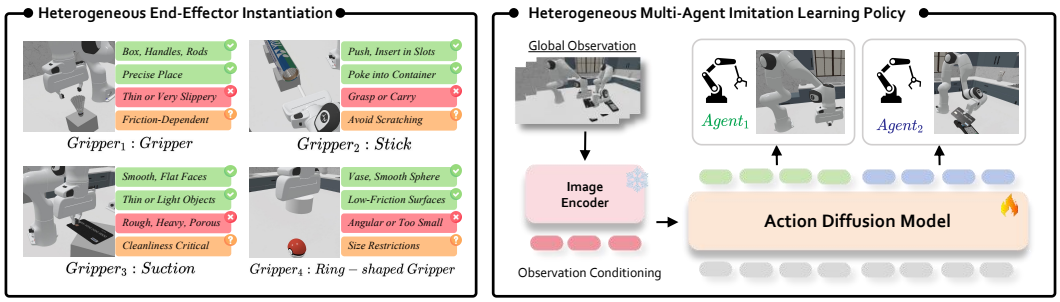

Figure 3: **Execution with Compositional Heterogeneous Agents.** (Left) Instantiation of diverse end-effectors in ManiSkill, each designed with unique structures that provide distinct capabilities. (Right) Construction of heterogeneous multi-agent imitation learning policies, enabling collaboration among different embodied agents to accomplish complex tasks.

3.1 Planning for Compositional Heterogeneous Embodied Agents

Data collection and curation. We reformulate robot selection as a visual reasoning problem (Fig. 2(a)): a task allocator chooses a set of robots from a predefined embodied-agent set $\mathcal{A}_{\text{embodied}}$, conditioned on a keyframe observation O and an instruction I, and outputs a selection $R = \{r_j\}, j \in [1, M]$. Task-specific templates determine ground-truth labels. GPT-4o proposes labels, a checker rejects template violations, and a human resolves remaining cases; GPT-4o does not define ground truth. The allocator g_{act} is powered by GPT-4o (OpenAI et al., 2024) and generates $R = g_{\text{act}}(I, O)$; a verification module C_{act} enforces task constraints with human oversight. The resulting dataset is *RoboMonster-P*.

Brain of *RoboMonster*. We build a multi-agent decision system that selects one or more embodied agents from the task instruction and current observation. To make the multi-agent scaffolding (rather than any particular backbone) the controlled variable, we keep the decision pipeline minimal and follow ReAct (Yao et al., 2023) and Reflection (Shinn et al., 2023). Before decision-making, we compile a *Robot Manual* from URDF and parameter files, specifying each agent's skills, action range, and execution cost; this serves as the knowledge base for reasoning about heterogeneous capabilities. As shown in Fig. 2(b), the MLLM-based **Planner** performs chain-of-thought reasoning over object locations, interaction logic, and potential failure cases (e.g., spatial constraints), and proposes a candidate composition. The LLM-based **Verifier** validates the proposal against the Robot Manual, checking semantic capability and cost constraints. It does not perform geometric feasibility checking or receive executor feedback. We validate the system on *RoboMonster-P* (Sec. 4).

3.2 Execution with Compositional Heterogeneous Agents

We build a set of heterogeneous agents and a broad distribution of manipulation tasks on top of the ManiSkill (Gu et al., 2023) simulation platform, gather training data with an automated MLLM-based pipeline, then train and evaluate a heterogeneous multi-agent system based on imitation learning.

Heterogeneous effector agents. We modify both control logic and visual appearance of the robot end-effectors to implement four types of heterogeneous grippers (Fig. 3(a)): (i) a *Gripper* for general rigid or graspable objects; (ii) a *Stick* for pushing or inserting in narrow cavities (e.g., a shuttlecock into a tube), unsuited to transport or grasping; (iii) a *Suction* for smooth, relatively flat surfaces (e.g., thin cards, smooth spheres) where good sealing is possible; and (iv) a *Ring-shaped Gripper* whose inner diameter is variable, which cages large round or cylindrical objects (e.g., vases, smooth spheres) that are hard to grasp securely with fingered or suction-based effectors. Details of these modifications are in Appendix D.

Data collection and curation. We design an automated MLLM-driven data-collection pipeline that gathers trajectory data for designated task–effector configurations, enabling *low-level policy* training. Inspired by RoboFactory (Qin et al., 2025), it consists of *RoboBrain*, which decomposes tasks and schedules primitive functions, and *RoboChecker*, which verifies whether the generated trajectories are reasonable and anomaly-free. When scheduling primitives we additionally include the end-effector

type as an input variable so each heterogeneous gripper produces its own action sequence. The resulting benchmark, *RoboMonster-E*, comprises **5 tasks** on a Franka Emika Panda arm (a 7-DoF robotic manipulator with interchangeable end-effectors) and is detailed in Sec. 4.2.

Compositional trajectory execution. We use a *Global-View + Shared-Policy* executor: global observations and per-arm proprioception in fixed role order condition one joint action chunk, which is sliced and dispatched to the arms. "Shared" refers to arms within one executor; weights are not shared across tasks. Each task has a designated heterogeneous executor and a matched gripper-only executor. We do not train the full task–effector Cartesian product. Unsupported selections count as planning failures, with no silent fallback. The Global/Local-View $\times$ Shared/Separate-Policy ablation is in Sec. 4.5; module roles and policy coverage are detailed in Appendix G. The execution-level study uses designated configurations; the end-to-end study uses the Planner's actual selections.

3.3 Demonstration Generation and Executor Dispatch

Primitive execution and synchronization. The demonstration library contains `approach`, `grasp/attach`, `lift`, `move`, `align`, `handover`, `insert/push`, `release`, and `retreat`. Each primitive takes object/target poses, arm/effector identity, and tolerances, and terminates on its pose/contact predicate or a timeout. RoboChecker rejects collisions, constraint violations, and incomplete terminal states. Both arms use one recording clock, yielding synchronized observation/action pairs. During handover, the donor releases only after the receiver's grasp predicate is satisfied. These routines generate demonstrations; learned policies generate actions during evaluation.

Joint policy interface. For a task τ and supported ordered effector configuration c, let x_t denote the policy's visual observation and $q_t^{(i)}$ the proprioception of arm i. The executor predicts an action chunk of horizon H:

$$\hat{\mathbf{a}}_{t:t+H-1} = \pi_{\tau,c}\Big(x_t, q_t^{(1)}, \ldots, q_t^{(N)}\Big). \tag{1}$$

The visual representation is RGB for DP and $\pi_{0.5}$, and point clouds for DP3. Fixed arm ordering specifies the roles in the joint output; each arm receives its corresponding action components. The subscript (τ, c) denotes a separately trained executor, not a universal policy conditioned on arbitrary task and embodiment tokens. Consequently, adding a new configuration requires compatible demonstrations and an executor even when its Manual entry can be added without retraining the Planner.

4 Experiments

4.1 Planning with Compositional Heterogeneous Embodied Agents

Setting. We cast planning as *agent selection* from a small library $\mathcal{E}$ of heterogeneous end-effectors. Given a single RGB scene image and a natural-language instruction, the model outputs one or multiple end-effectors from $\mathcal{E}$ (label set in Appendix C.2). We evaluate on a 200-example test set sampled from *RoboMonster-P* (sampling protocol and distribution in Appendix C). All single-pass VLMs share the same prompt schema and are restricted to one-shot selection; *MAS* may internally perform at most one Planner–Verifier iteration but still outputs a single final label. Predictions are canonicalised by string matching and scored by top-1 accuracy.

Baselines. We compare against strong open- and closed-source VLMs (*Qwen2.5-VL-32B-Instruct*, *GLM-4.5V*, *GPT-5*, *Gemini-2.5-Pro*, *Claude Sonnet 4 (2025-05-14)*) and our **MAS**, which couples a Planner (MLLM reasoning) with a Verifier (LLM rule/constraint checker) over a Robot Manual.

Agent selection results. Tab. 1 summarises top-1 accuracy on *RoboMonster-P*. Policies based on a single gripper are limited by their mechanical structure and reach only $\sim$12%, confirming that single-effector decision-making is structurally lower-bounded on these tasks. Closed-source models outperform open-source ones, and our MAS attains the best overall accuracy. The margin over GPT-5 is small ($+1$ pt), so the MAS does not claim a VLM-level capability: it provides a constraint-aware selection layer that (i) feeds the executor with feasible configurations and (ii) participates in the end-to-end pipeline of Sec. 4.3.

Table 1: **Agent selection accuracy** on *RoboMonster-P* (200 examples). No finetuning; temperature $= 0.0$.

Model	Top-1 Acc.
Gripper Only	0.120
Qwen2.5-VL-32B-Instruct	0.235
GLM-4.5V	0.260
GPT-5	0.440
Gemini-2.5-Pro	0.425
Claude Sonnet 4	0.415
MAS (Planner+Verifier, ours)	**0.450**

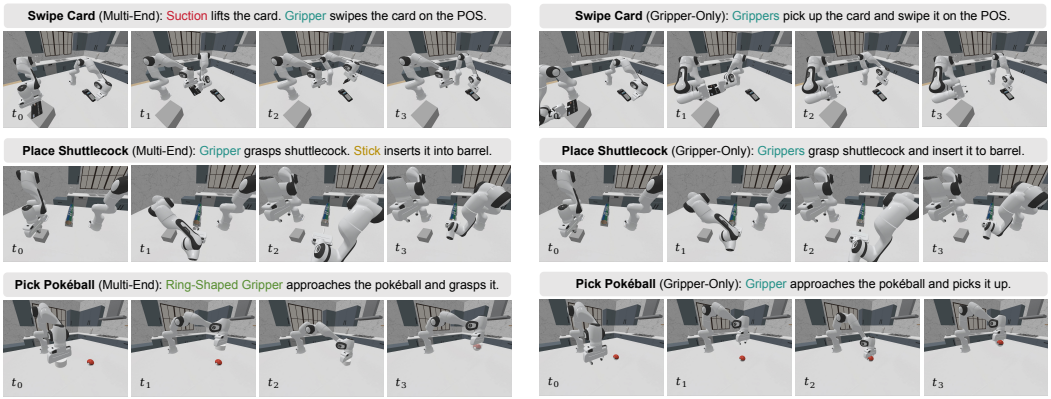

Figure 4: Three representative tasks (**Swipe Card**, **Place Shuttlecock**, **Pick Pokéball**) covering all four end-effector types. Left: the heterogeneous end-effector setup. Right: the gripper-only counterpart.

4.2 Execution with Compositional Heterogeneous Embodied Agents

Setting. We collect expert demonstrations using the *RoboFactory* framework (Qin et al., 2025) on top of ManiSkill (Tao et al., 2024a), with demonstration budgets of 25/50/75 trajectories for DP (Chi et al., 2023) and 50/100/150 for DP3 (Ze et al., 2024) (DP3 trajectories carry less information due to sparse point-cloud sampling). To further test whether a stronger backbone closes the heterogeneous-vs-gripper gap, we additionally fine-tune the generalist Vision-Language-Action model $\pi_{0.5}$ (Intelligence et al., 2025) on the same 25/50/75 budgets as DP under the identical *Global-View + Shared-Policy* interface, so that only the backbone changes. Hardware tiers and training hyperparameters are in Appendix A; the $\pi_{0.5}$ recipe is in Appendix F.3. Each task is evaluated across 100 seeds with varied initial poses; a maximum action-step limit (twice the warm-up average over 20 samples) terminates unsuccessful rollouts. Success criteria are in Appendix A.

We design five tasks under single- and dual-agent settings. The descriptions below assume that the *RoboMonster* brain has already filtered out unsuitable effectors; the discussion is therefore between the **optimal end-effector combinations** and the gripper-only setting. The end-to-end study (Sec. 4.3) relaxes this assumption. **(1) Suction-lift Card**: $\mathcal{A}_1$ uses suction or gripper to lift a credit card placed on a cube. **(2) Pick Pokéball**: $\mathcal{A}_1$ uses a ring-shaped or normal gripper to pick up a smooth Pokéball. **(3) Pick Vase**: $\mathcal{A}_1$ uses a ring-shaped or normal gripper to pick up a vase. **(4) Place Shuttlecock**: $\mathcal{A}_1$ grasps the shuttlecock with a gripper and positions it at the barrel opening; $\mathcal{A}_2$ pushes it in with a stick (or, in the baseline, a closed gripper). **(5) Swipe Card**: $\mathcal{A}_1$ lifts the credit card (suction or gripper) and hands off to $\mathcal{A}_2$, which aligns the card to the POS-terminal slot and inserts it.

Single-agent results. On *Suction-lift Card*, *Pick Pokéball*, and *Pick Vase*, the heterogeneous setup reaches average success rates of **98.6%**, **95.6%**, and **100%**, substantially above the gripper-only baseline (24.9%, 66.0%, 39.0%). Stronger backbones raise the gripper-only floor (e.g., Suction-lift Card 12% on DP, 21% on DP3, 41% on $\pi_{0.5}$) but never close the gap to the heterogeneous setup. This

Table 2: Success rates of heterogeneous end-effectors and gripper-only across five tasks and three policy backbones (DP, DP3, $\pi_{0.5}$) with matched demonstration budgets. **Avg.** is the nine-cell DP+DP3+$\pi_{0.5}$ mean of each row. (Abbr.: E-e. = End-effector, R-s. = Ring-shaped, Gri. = Gripper, Sti. = Stick, Suc. = Suction.)

Task Name	**E-e. Setup**	*Diffusion Policy*			*3D Diffusion Policy*			$\pi_{0.5}$ (VLA)			
		25 Demo	50 Demo	75 Demo	50 Demo	100 Demo	150 Demo	25 Demo	50 Demo	75 Demo	**Avg.**
Suction-lift Card	Gripper Only	7%	14%	**15%**	21%	20%	**23%**	35%	42%	**47%**	24.9%
	Ours (Suction)	**100%**	**100%**	**100%**	**100%**	**100%**	93%	94%	**100%**	**100%**	98.6%
Pick Pokéball	Gripper Only	37%	**69%**	64%	54%	**75%**	73%	52%	**86%**	84%	66.0%
	Ours (R-s. Gri.)	78%	**100%**	**100%**	88%	**100%**	**100%**	94%	**100%**	**100%**	95.6%
Pick Vase	Gripper Only	28%	**41%**	37%	14%	52%	**54%**	26%	47%	**52%**	39.0%
	Ours (R-s. Gri.)	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	100%
Place Shuttlecock	Gripper Only	46%	43%	**48%**	42%	45%	**46%**	39%	56%	**59%**	47.1%
	Ours (Gri. & Sti.)	37%	51%	**54%**	67%	**76%**	75%	31%	62%	**78%**	59.0%
Swipe Card	Gripper Only	17%	23%	**25%**	2%	19%	**31%**	22%	27%	**31%**	21.9%
	Ours (Suc. & Gri.)	60%	67%	**77%**	5%	62%	**71%**	57%	71%	**82%**	61.3%

indicates that the gap is structural, a property of the embodiment for which a fingered gripper has no reliable affordance on thin cards or tall vases, rather than an artefact of the policy class. Specialized end-effectors reduce task ambiguity and let the policies exploit clearer action affordances.

Dual-agent results and data efficiency. On long-horizon dual-agent tasks the heterogeneous setup still clearly outperforms gripper-only (*Place Shuttlecock*: 59.0% vs. 47.1%; *Swipe Card*: 61.3% vs. 21.9%, both averaged over all nine demo×policy cells), though absolute performance declines due to compounding coordination errors. Data efficiency depends on the regime: simple single-agent tasks saturate at moderate budgets (DP 50, DP3 100, $\pi_{0.5}$ already at 25 on Pick Vase), while long-horizon dual-agent tasks generally need the largest budget within each backbone (DP 75, DP3 150, $\pi_{0.5}$ 75). Two recurring failure modes on **Swipe Card**, end-effector misalignment during the initial card sweep and unstable regrasping during the handover, are visualised and discussed in Appendix F (Fig. 8). The effect of global observations and shared control is evaluated in Sec. 4.5.

4.3 End-to-End Evaluation: Planner → Execute

The per-stage execution numbers assume the *oracle-optimal* end-effector combination and therefore do not stress the Planner under realistic uncertainty. We feed the Planner's actual outputs into the trained policies of Sec. 3.2 and report *end-to-end* task success on the five *RoboMonster-E* tasks (Tab. 3); the Oracle uses DP with 75 demonstrations, as listed in Tab. 2.

Results. Averaged over the five tasks, the full pipeline reaches **76.4%**, more than doubling gripper-only (37.8%) and recovering about 80% of the heterogeneous-versus-gripper gap ($(76.4-37.8)/(86.2-37.8)$). The Planner matches the Oracle on *Suction-lift Card*, *Pick Pokéball*, and *Place Shuttlecock*, indicating that for tasks where the right composition is unambiguous the rule-aware Verifier reliably routes the executor to it. The remaining gap concentrates on *Pick Vase* (64% vs. Oracle 100%) and *Swipe Card* (64% vs. Oracle 77%), where selection errors on visually ambiguous instances or transient failures in the dual-arm handover propagate to end-to-end failures.

4.4 Real-World Experiments on ALOHA

We performed real-world experiments on ALOHA without bespoke hardware redesign, emulating heterogeneous end-effectors (suction, rod, ring) on four tasks: **Suction-lift Card** and **Place Shuttlecock** (single- and dual-arm primitives), **Pick Vase** (caging affordance), and the long-horizon dual-arm **Swipe Card** task that stresses the full planning-to-execution interface on the real robot. The Planner runs once at task start to dispatch an executor, with no replanning. For each task we collected 50 trajectories for training and ran 10 trials (Suction-lift Card, Place Shuttlecock) or 20 trials (Pick Vase, Swipe Card) at test time. Across the four tasks, gripper-only succeeds on only $4/60$ trials, while the heterogeneous setup succeeds on $40/60$ ($6.7\% \rightarrow 66.7\%$ aggregate). The gap is most striking where a single gripper has no reliable affordance: gripper-only fails outright on *Suction-lift Card* ($0/10$)

Table 3: **End-to-end Planner → Execute success rate** on *RoboMonster-E* (100 seeds per task; DP with 75 demonstrations per task). GRIPPER-ONLY is the lower bound (no selection); ORACLE is the upper bound (selection always optimal); ROBOMONSTER (E2E) is the full pipeline.

Task	GRIPPER-ONLY	ROBOMONSTER (E2E)	ORACLE
Suction-lift Card	15%	100%	100%
Pick Pokéball	64%	100%	100%
Pick Vase	37%	64%	100%
Place Shuttlecock	48%	54%	54%
Swipe Card	25%	64%	77%
Average	37.8%	**76.4%**	86.2%

Table 4: Real-world success rate on ALOHA (successful / total trials, matched seeds).

Task	**Gripper-Only**	**Ours**
Suction-lift Card	0/10	7/10
Place Shuttlecock	1/10	9/10
Pick Vase	3/20	16/20
Swipe Card (long-horizon, dual-arm)	0/20	8/20

and on the long-horizon *Swipe Card* ($0/20$), whereas the heterogeneous setup reaches $7/10$ and $8/20$ respectively. On *Pick Vase* caging lifts success from $3/20$ to $16/20$, and on *Place Shuttlecock* the stick effector lifts it from $1/10$ to $9/10$. Fig. 5 shows example executions; switching between suction, rod, and ring effectors gives a clear adaptability and robustness advantage on real hardware.

4.5 ABLATION: GLOBAL OBSERVATION AND JOINT CONTROL

We compare four observation/control interfaces on Swipe Card with DP and the same 25/50/75 demonstration budgets (Tab. 5). Under heterogeneous effectors, Global-View + Shared-Policy reaches 60/67/77%, while Global-View + Separate-Policy reaches 35/42/51%. At 75 demonstrations, joint control improves success by 26 percentage points with the observation scope held fixed. Local-View + Separate-Policy reaches 8%, and Local-View + Shared-Policy reaches 2%, showing that parameter sharing alone does not compensate for restricted observations in this setting.

The gripper-only comparison follows the same ordering at 75 demonstrations: 25% for global/shared, 19% for global/separate, and 0% for both local variants. Global information and joint output organization therefore help coordination, while the remaining 77% versus 25% gap within the global/shared interface supports the contribution of heterogeneous affordances. This ablation concerns one long-horizon task; it motivates the chosen executor interface without establishing that sharing is universally preferable across tasks or embodiments.

4.6 DISCUSSION

The execution, end-to-end, and hardware studies consistently support task-matched heterogeneous affordances. Scaling DP to DP3 or $\pi_{0.5}$ improves some gripper-only results but does not remove the observed gap. Our contribution is the formulation and system-level evidence, rather than a new policy backbone or planning algorithm.

Selection and execution failure. The oracle comparison provides complementary diagnostics. Pick Vase reaches 100% with the designated executor but 64% end-to-end, identifying selection and dispatch as targets for improvement. Place Shuttlecock reaches 54% in both settings, so improving selection alone cannot remove the failures observed with its designated executor. These are aggregate comparisons, rather than a per-rollout causal decomposition. The Swipe Card examples in Fig. 8 expose two concrete execution problems: misalignment during the initial card sweep and loss of the card during handover. Global observation and synchronized training help organize these interactions but do not guarantee contact stability.



Figure 5: Real-world tasks. Top row: Place Shuttlecock. Bottom row: Suction-lift Card.

Table 5: Paradigm ablation. Performance comparison across global/local view and shared/separate policy on **Swipe Card** under *Diffusion Policy*.

		Swipe Card (*Diffusion Policy*)		
Paradigm	**E-e. Setup**	25 Demo	50 Demo	75 Demo
Global-View	Gripper Only	17%	23%	25%
+ *Shared-Policy*	Heterogeneous E-e.	**60%**	**67%**	**77%**
Local-View	Gripper Only	0%	0%	0%
+ *Separate-Policy*	Heterogeneous E-e.	0%	2%	8%
Local-View	Gripper Only	0%	0%	0%
+ *Shared-Policy*	Heterogeneous E-e.	0%	0%	2%
Global-View	Gripper Only	10%	17%	19%
+ *Separate-Policy*	Heterogeneous E-e.	35%	42%	51%

Reading the evidence. The main execution table varies demonstration budget within each backbone, whereas the end-to-end comparison fixes DP at 75 demonstrations. Their averages answer different questions and should not be interchanged. Hardware rates report raw successful-trial counts, with only 10 or 20 trials per task. They provide evidence of feasibility on ALOHA, while leaving uncertainty about performance across broader object distributions and repeated training runs. Likewise, a one-point selection advantage on 200 examples is insufficient by itself to establish a general reasoning improvement.

4.7 Limitations

Execution policies remain task/configuration-specific: adding a new multi-effector combination requires synchronized demonstrations and a compatible trained executor. The Planner reasons semantically over a hand-authored Manual and a four-effector library, without geometric feasibility checking or execution feedback. Long-horizon coordination remains difficult. The simulator's one-shot attachment abstraction idealizes contact, and the limited ALOHA study does not establish broad real-world generalization. The experiments do not evaluate zero-shot compositional control or compare against a gripper with exchangeable tools.

5 Conclusion

RoboMonster connects capability-aware end-effector selection with coordinated execution. It reaches 76.4% end-to-end success versus 37.8% for gripper-only, and 40/60 ALOHA successes versus 4/60. Comparisons across three policy backbones support the value of task-matched heterogeneous affordances within the evaluated task library. Zero-shot compositional control, geometric validation, and closed-loop replanning remain open challenges.

AI USE STATEMENT

Large language models are used in the research pipeline for agent selection, verification, and demonstration generation, as described in Section 3.1, Section 3.2, and Appendix C. We also used generative AI tools for language polishing, manuscript editing and restructuring, and assistance with converting the manuscript to the ICLR format. The authors take responsibility for the final content of the paper, including AI-assisted text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

Section 4 describes the evaluation tasks, demonstration budgets, and success metrics. Appendix A reports training hyperparameters and task success criteria; Appendix C describes the planning setup and prompts; Appendix D provides simulator implementation details. The $\pi_{0.5}$ fine-tuning recipe and additional experiments are provided in Appendix F. Policy coverage and hardware protocols are described in Appendix G.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024a.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024b.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, pp. 02783649241273668, 2023.

Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu. Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 39(5):3929–3945, 2023.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. In *The Eleventh International Conference on Learning Representations*, 2023.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges, 2024a. URL https://arxiv.org/abs/2402.01680.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams. *arXiv preprint arXiv:2403.12482*, 2024b.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Li Kang, Xiufeng Song, Heng Zhou, Yiran Qin, Jie Yang, Xiaohong Liu, Philip Torr, Lei Bai, and Zhenfei Yin. Viki-r: Coordinating embodied multi-agent cooperation via reinforcement learning. *arXiv preprint arXiv:2506.09049*, 2025.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems, 2025. URL https://arxiv.org/abs/2410.02189.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023.

Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. In *International Conference on Machine Learning*, pp. 20725–20745. PMLR, 2023.

Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16467–16476, 2024.

Zhixuan Liang, Yao Mu, Yixiao Wang, Tianxing Chen, Wenqi Shao, Wei Zhan, Masayoshi Tomizuka, Ping Luo, and Mingyu Ding. Dexhanddiff: Interaction-aware diffusion planning for adaptive dexterous manipulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1745–1755, 2025.

Jiaqi Liu, Chengkai Xu, Peng Hang, Jian Sun, Mingyu Ding, Wei Zhan, and Masayoshi Tomizuka. Language-driven policy distillation for cooperative driving in multi-agent reinforcement learning. *IEEE Robotics and Automation Letters*, 2025.

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

Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 286–299. IEEE, 2024.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

Kazuma Obata, Tatsuya Aoki, Takato Horii, Tadahiro Taniguchi, and Takayuki Nagai. Lip-llm: Integrating linear programming and dependency graph with large language models for multi-robot task planning. *IEEE Robotics and Automation Letters*, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo

Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Yiran Qin, Li Kang, Xiufeng Song, Zhenfei Yin, Xiaohong Liu, Xihui Liu, Ruimao Zhang, and Lei Bai. Robofactory: Exploring embodied agent collaboration with compositional constraints. *arXiv preprint arXiv:2503.16408*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

H Singh, RJ Das, M Han, P Nakov, and I Laptev. Malmm: Multi-agent large language models for zero-shot robotics manipulation. arxiv 2024. *arXiv preprint arXiv:2411.17636*, 2024.

Purdue University SMART Lab. SMART-LLM: Smart multi-agent robot task planning using large language models. https://github.com/SMARTlab-Purdue/SMART-LLM, 2023.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024a.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems*, 37:51963–51993, 2024b.

Chenxi Wang, Hongjie Fang, Hao-Shu Fang, and Cewu Lu. Rise: 3d perception makes real-world robot imitation simple and effective. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2870–2877. IEEE, 2024a.

Weizheng Wang, Ike Obi, and Byung-Cheol Min. Multi-agent llm actor-critic framework for social robot navigation. *arXiv preprint arXiv:2503.09758*, 2025a.

Yongdong Wang, Runze Xiao, Jun Younes Louhi Kasahara, Ryosuke Yajima, Keiji Nagatani, Atsushi Yamashita, and Hajime Asama. Dart-llm: Dependency-aware multi-robot task decomposition and execution using large language models. *arXiv preprint arXiv:2411.09022*, 2024b.

Yujin Wang, Quanfeng Liu, Zhengxin Jiang, Tianyi Wang, Junfeng Jiao, Hongqing Chu, Bingzhao Gao, and Hong Chen. Rad: Retrieval-augmented decision-making of meta-actions with vision-language models in autonomous driving. *arXiv preprint arXiv:2503.13861*, 2025b.

Han Wei et al. A modern survey of llm planning capabilities. In *Proceedings of ACL*, 2025.

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Se June Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, et al. Latent action pretraining from videos. In *CoRL 2024 Workshop on Whole-body Control and Bimanual Manipulation: Applications in Humanoids and Beyond*, 2024.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023.

Hongxin Zhang, Zeyuan Wang, Qiushi Lyu, Zheyuan Zhang, Sunli Chen, Tianmin Shu, Behzad Dariush, Kwonjoon Lee, Yilun Du, and Chuang Gan. Combo: compositional world models for embodied multi-agent cooperation. *arXiv preprint arXiv:2404.10775*, 2024.

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

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, et al. Sotopia: Interactive evaluation for social intelligence in language agents. *arXiv preprint arXiv:2310.11667*, 2023.

RoboMonster Supplementary Material

A TRAINING DETAILS

In this section, we present the training details, covering the hyperparameter configurations of the two baselines as well as representative training times.

Table 6: Hyperparameters for *Diffusion Policy* training. (Abbr.: H-Paras = Hyperparameters, Pre. = Prediction, Obs. = observation, Act. = Action, BS = Batch Size)

	Diffusion Policy				
H-Paras	**Suction-lift Card**	**Pick Pokéball**	**Pick Vase**	**Place Shuttlecock**	**Swipe Card**
Pre. Horizon	32	32	32	32	32
Obs. Horizon	20	20	20	20	20
Act. Horizon	8	8	8	8	8
Image Shape	3 × 256 × 256	3 × 256 × 256	3 × 256 × 256	3 × 256 × 256	3 × 256 × 256
Action Shape	8	8	8	15	16
4060 BS	N/A	N/A	N/A	N/A	N/A
2 × 4090 BS	32	32	32	32	32
2 × H800 BS	64	64	64	64	64
Learning Rate	1.0e-4	1.0e-4	1.0e-4	1.0e-4	1.0e-4
Warm-up Steps	500	500	500	500	500
Betas	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]
Weight Decay	1.0e-6	1.0e-6	1.0e-6	1.0e-6	1.0e-6
Epsilon	1.0e-8	1.0e-8	1.0e-8	1.0e-8	1.0e-8
Epochs	300	300	300	300	300

Table 7: Hyperparameters for *3D Diffusion Policy* training.

	3D Diffusion Policy				
Hyperparameters	**Suction-lift Card**	**Pick Pokéball**	**Pick Vase**	**Place Shuttlecock**	**Swipe Card**
Prediction Horizon	32	32	32	32	32
Observation Horizon	20	20	20	20	20
Action Horizon	8	8	8	8	8
Point Cloud Shape	3 × 1024	3 × 1024	3 × 1024	3 × 1024	3 × 1024
Action Shape	8	8	8	15	16
4060 Batch Size	32	32	32	32	32
2 × 4090 Batch Size	128	128	128	128	128
2 × H800 Batch Size	256	256	256	256	256
Learning Rate	1.0e-4	1.0e-4	1.0e-4	1.0e-4	1.0e-4
Warm-up Steps	500	500	500	500	500
Betas	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]	[0.95, 0.999]
Weight Decay	1.0e-6	1.0e-6	1.0e-6	1.0e-6	1.0e-6
Epsilon	1.0e-8	1.0e-8	1.0e-8	1.0e-8	1.0e-8
Epochs	300	300	300	300	300

Diffusion Policy. We employ a **CNN-based** *Diffusion Policy* and evaluate its training performance across representative resource-constrained (NVIDIA RTX 4060 GPU), moderate-resource (2 × NVIDIA RTX 4090 GPU), and high-resource (2 × NVIDIA H800 GPU) computing platforms. The corresponding hyperparameter settings are summarized in Tab. 6. On the resource-constrained platform, the training could not be completed due to the large data volume. For researchers limited to low-resource platforms, one may attempt to simultaneously reduce the Horizon, Image Shape, and Batch Size during training (though we do not recommend this as a preferred strategy).

It is worth noting that we adopted the `torch.optim.AdamW` optimizer. The hyperparameter values for the Learning Rate, Warm-up Steps, Betas, Weight Decay, and Epsilon are all specified in the corresponding Tab. 6. We report several representative training times as follows:

1. On 2 × 4090 GPU: with 75 demos and 300 epochs for the **Swipe Card** task, approximately **20 hours**.
2. On 2 × H800 GPU: with 75 demos and 300 epochs for the **Swipe Card** task, approximately **13 hours**.
3. On 2 × H800 GPU: with 75 demos and 300 epochs for the **Suction-lift Card** task, approximately **6 hours**.

Table 8: Task Descriptions for the *RoboMonster-E* Benchmark

Task	Description	Target Condition
Suction-lift Card	A credit card is placed at the edge of a cube, and a robotic arm must choose an appropriate end-effector to grasp the card and lift it to a specified height.	The height of the credit card reaches a predefined threshold.
Pick Pokéball	A Pokéball is placed on the table, and a robotic arm must choose an appropriate end-effector to grasp it and lift it to a specified height.	The height of the Pokéball reaches a predefined threshold.
Pick Vase	A vase is placed on the table, and a robotic arm must choose an appropriate end-effector to grasp it and lift it to a specified height.	The height of the vase reaches a predefined threshold.
Place Shuttlecock	A shuttlecock and a barrel are placed on the table. Two robotic arms should each choose an appropriate end-effector: one to place the shuttlecock at the rim of the barrel, and the other to insert it inside.	We assume that the number of shuttlecocks already inside the barrel follows a 50% probability of being six and a 50% probability of being seven. The task requires pushing the shuttlecock into a position adjacent to the outermost shuttlecock.
Swipe Card	A credit card is placed on top of a cube, while a POS terminal is located on the table. Two robotic arms should each choose an appropriate end-effector: one to lift the credit card and hand it over to the other arm, and the other to insert the card into the slot of the POS terminal.	The distance between the credit card and the slot of the POS terminal is smaller than a predefined threshold.

3D Diffusion Policy. For *3D Diffusion Policy*, we adopt an almost identical training strategy. The specific training configurations are listed in Tab. 7. A notable distinction compared to *Diffusion Policy* is that *3D Diffusion Policy* achieves much shorter training times, and is considerably more friendly for researchers with low-resource platforms. We report several representative training times as follows:

1. On 1 $\times$ 4060 GPU: with 150 demos and 300 epochs for the **Swipe Card** task, approximately **23 hours**.
2. On 2 $\times$ 4090 GPU: with 150 demos and 300 epochs for the **Swipe Card** task, approximately **7 hours**.
3. On 2 $\times$ H800 GPU: with 150 demos and 300 epochs for the **Swipe Card** task, approximately **2.5 hours**.

B Evaluation Details

Tasks Evaluation. We interpolated the action sequences generated by the model to make the motion trajectories smoother. For each task, we evaluated it across 100 seeds, varying the initial object positions and environmental conditions. We introduced a maximum action step limit for each task to assess the success rate. If the task was not completed within this limit, it was considered a failure. To set a reasonable threshold, we conducted warm-up tests on 20 samples to estimate the average number of steps required to complete the task. The maximum action step limit was set to 2 times this average value. The success criteria for each task, including target conditions, are detailed in Tab. 8.

We provide a detailed illustration of the overall evaluation pipeline of ***RoboMonster*** (see Fig. 6). The system first extracts information from the image, and then, through planning in ***RoboMonster* Brain**, selects suitable end-effectors for the agents $\mathcal{A}_1$ and $\mathcal{A}_2$. The selected end-effectors are employed to execute the tasks using either DP or DP3.

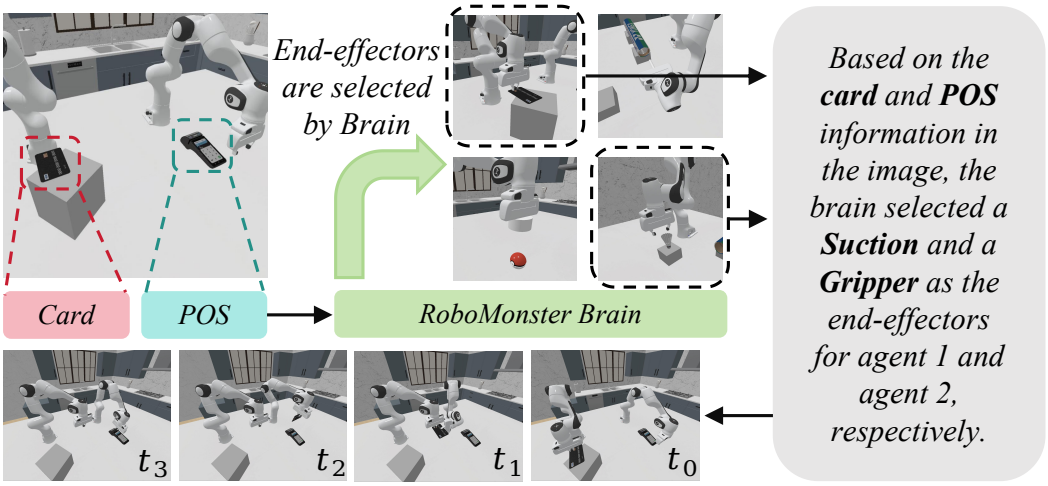

Figure 6: Flowchart of the complete evaluation process, including both planning and execution.

C Details of RoboMonster Brain

C.1 Implementation Details

Single-Agent Pipeline. We implement a single-agent pipeline. Images are provided as base64 data URLs when available. End-effector options are strictly derived from the provided robots of the current sample.

Multi-Agent System. The deployed system has two agents: Planner and Verifier. Analyzer, Planner, and Selector below denote internal stages of the deployed Planner; Validator corresponds to the Verifier. These stage names do not imply four independently deployed agents. API transport retries in the helper below are distinct from the bounded Planner–Verifier reasoning iteration.

For Single-Agent Pipeline and Multi-Agent System, our relevant call site is:

```
@backoff.on_exception(backoff.expo, Exception, max_tries=5, max_time=60)
def api_call_with_retry(messages, model_name):
    return client.chat.completions.create(
        model=model_name,
        messages=messages,
        temperature=0,
        max_tokens=2000,
    )
```

C.2 Prompt Design

C.2.1 End-Effector Availability

The following dictionary provides descriptions for the planning runner. Options are filtered by each sample's available-robot list (Appendix C.5); listing Unitree or ANYmal here does not indicate execution evaluation on those platforms. Execution experiments use the four effector types described in the main text.

```
    robot_to_end_effector_desc = {
        'stompy': 'Claw hand on Stompy: Stompy is a bipedal robot
    designed for dynamic walking and stomping tasks, featuring
    articulated arms. Color: Light blue body with yellow and orange
    accents. Equipped with a claw hand for grasping objects.',

        'fetch': 'Gripper on Fetch: Fetch is a wheeled robot with a
    flexible arm for object manipulation, designed for mobility and
```

```
    dexterity. Color: White with blue and black accents. Uses a gripper
    end-effector: two symmetric \'fingers/plates\' that open and close in
     parallel along the same line. From the front, it looks like two
    parallel small flat plates with a gap in the middle; from the side,
    you can see the \'top/bottom or left/right clamping\' shape.
    Versatile and precise, suitable for most regular or rigid objects
    that can be gripped; adapts to some size variation. May be unstable
    for very thin, slippery, or objects with insufficient gripping
    surfaces.',

        'unitree_h1': 'Dexterous hands on Unitree_H1: Unitree_H1 is a
    humanoid robot with arms and legs designed for human-like movements
    and tasks. Color: Black. Equipped with dexterous hands for complex
    manipulation tasks requiring fine motor control. Best for delicate
    operations and complex assembly tasks. Excellent for precise
    manipulation of various objects.',

        'panda': 'Gripper on Panda: Panda is a fixed robotic arm designed
     for precise and delicate manipulation tasks. Color: White with black
     accents. Uses a gripper end-effector: two symmetric \'fingers/plates
    \' that open and close in parallel along the same line. From the
    front, it looks like two parallel small flat plates with a gap in the
     middle; from the side, you can see the \'top/bottom or left/right
    clamping\' shape. Versatile and precise, suitable for most regular or
     rigid objects that can be gripped; adapts to some size variation.
    May be unstable for very thin, slippery, or objects with insufficient
     gripping surfaces.',

        'unitree_go2': 'Claw hand on Unitree_Go2: Unitree_Go2 is a
    compact quadrupedal robot optimized for agile movement and stability
    with four legs for efficient locomotion. Color: White. Equipped with
    a claw hand for grasping objects.',

        'anymal_c': 'Claw hand on Anymal_C: Anymal_C is a quadrupedal
    robot built for navigating rough terrains and performing complex
    tasks with four articulated legs. Color: Red and black with some
    accents. Equipped with a claw hand for grasping objects.',

        'Suction': 'The end looks like a small round contact pad (not two
     clearly separated jaws). In simulation, it is always closed, without
     a real vacuum mechanism; it \'simulates suction\' by pressing the
    small round pad (actually a closed gripper) normally against the
    object\'s surface. Suitable for smooth, relatively flat targets (such
     as thin cards or flat blocks on a table), most stable when a good
    sealing surface is available. Not suitable for porous/rough/high
    curvature or overly heavy objects, or when there are insufficient
    suction points.',

        'Circle': 'A short cylindrical tube with an opening. No obvious
    jaws or rod-like actuators. The opening cannot be seen from top or
    side views, only from below. SPECIFICALLY DESIGNED for round/
    spherical objects of all sizes including balls. Provides stable
    constraint by \'caging\' round targets like balls, spheres, vases or
    buckets that are hard to grip with gripper jaws and not suitable for
    suction. IDEAL CHOICE for any ball-related tasks.',

        'Stick': 'A thin, smooth rod with no jaws or suction pad at the
    end. Used for pushing, inserting, or clearing in narrow cavities/
    channels (e.g., pushing a shuttlecock into a bucket, clearing a thin
    pipe); not suitable for carrying or holding objects.'
     }
```

C.2.2 System Instruction

The system message enumerates the available end-effectors and hard-constrains the answer space:

```
instruction_following = (
    r'Available end-effectors in this scenario: {robot_set} '
    r'Available end-effector options: {available_end_effectors} '
    r'CRITICAL: You can ONLY choose from the end-effector options
listed above! These are the ONLY available options for this specific
scenario. '
    r'Task: Analyze the given task and select the appropriate end-
effector(s) in the correct execution order. '
    r'IMPORTANT RULES: '
    r'1. You MUST select ONLY from the available end-effector options
 listed above - no exceptions! '
    r'2. If an end-effector is not in the available options list, you
 CANNOT use it, even if it might seem suitable for the task. '
    r'3. For single-step tasks, choose one end-effector. For multi-
step tasks, list multiple end-effectors in execution order. '
    r'4. Consider object properties (size, shape, weight, material)
when selecting from the available options. '
    r'5. Consider manipulation requirements (precision, force,
dexterity) when choosing from available options. '
    r'6. Use the exact end-effector names as shown in the available
options list. '
    r'Reasoning process: Think through the task step-by-step,
considering object properties, manipulation requirements, and
execution sequence, while staying within the constraints of available
 options. '
    r'Response format: Provide your reasoning in <think> </think>
tags, followed by your final answer in <answer> </answer> tags. '
    r'The answer must be a Python list of end-effector names in
execution order, using the EXACT names from the available options. '
    r'Example formats: ["claw hand on stompy"] or ["gripper on fetch",
 "dexterous hands on unitree_h1"] for multi-step tasks.'
)
```

C.2.3 User Message

The user content contains an optional image (base64 data URL) and a text line "`Task Description: ...`".

MAS Prompt. Our Multi-Agent System employs a sequential four-stage pipeline where each agent has specialized prompts:

Analyzer Agent. Extracts task components using pattern-based fallback methods:

```
system_prompt = """You are a task analysis expert specializing in
robotic manipulation tasks.

Your role is to analyze the given task description and identify:
1. What objects are involved in the task
2. What actions need to be performed
3. Important properties of the objects (shape, surface, size)

Keep your response simple and clear. Focus on the key information
needed for tool selection."""
```

Planner Agent. Creates execution plans with tool requirement analysis:

```
system_prompt = """You are an execution planning expert for robotic
manipulation tasks.

Based on the task analysis, create a detailed execution plan.
```

```
    PLANNING GUIDELINES:
    1. TASK COMPLEXITY ANALYSIS:
    - Single-step tasks: One main action (e.g., "move spoon")
    - Multi-step tasks: Multiple distinct actions (e.g., "insert bread,
    activate toaster, place bowl")

    2. STEP IDENTIFICATION:
    - Break down the task into atomic actions
    - Identify if different actions require different tools
    - Note sequence dependencies (what must happen first)

    3. TOOL REQUIREMENTS:
    - Consider if each step needs a different end-effector
    - Flag when precision vs. power tools are needed
    - Identify if specialized tools are required (e.g., 'circle' for
    balls, 'suction' for cards)

    4. OUTPUT FORMAT:
    - Clearly state if this is a SINGLE-STEP or MULTI-STEP task
    - List each step with its tool requirements
    - Highlight any special considerations

    Focus on step-by-step breakdown and tool requirement analysis."""
```

Selector Agent. Makes final tool selections with enhanced decision logic:

```
    system_prompt = """Available end-effectors:

    {ROBOT_SET}

    CRITICAL TASK ANALYSIS FRAMEWORK:

    1. PRECISE TASK TYPE CLASSIFICATION:
    a) Simple single-action tasks: "place apple on table", "pick up
    banana"
     Use EXACTLY ONE tool best suited for the primary object

    b) Multi-action same-capability tasks: "place banana and apple in
    bowl"
     Use ONE versatile tool that can handle all objects

    c) Multi-action different-capability tasks: "insert bread, activate
    toaster, place bowl"
     Use MULTIPLE tools: each action needs different capabilities
     Bread insertion: delicate manipulation (dexterous hands)
     Toaster activation: button/lever pressing (gripper/hands)
     Bowl placement: general manipulation (gripper/hands)

    2. ENHANCED OBJECT-TOOL MATCHING RULES:
    a) OBJECT-SPECIFIC tools (use ONLY when object demands it):
    - Spherical objects (balls, oranges): 'circle' IF no other tool can
    handle safely
    - Flat rigid items (cards, plates): 'suction' IF precision placement
    needed
    - Delicate/soft items (bread, fruit): 'dexterous hands' IF crushing
    is a risk

    b) VERSATILE tools (prefer when possible):
    - 'gripper on fetch': Mobile platform, good for most pick/place tasks
    - 'dexterous hands on unitree_h1': Complex manipulation, fine motor
    control
    - 'claw hand on stompy/anymal_c': Power tasks, robust grasping

    3. TOOL SELECTION DECISION TREE:
    Step 1: Count distinct action types needed
```

```
    Step 2: For each action, determine minimum capability required
    Step 3: Find tools that can handle each capability
    Step 4: Minimize tool count while meeting all requirements

    Example: "Insert bread, activate toaster, place bowl"
    - Action 1: Insert bread  needs delicate manipulation  dexterous
    hands
    - Action 2: Activate toaster  needs precise button press  gripper or
    hands
    - Action 3: Place bowl  needs general manipulation  gripper or hands
    - Decision: Since actions 2&3 need similar capability but action 1 is
     unique
    - Result: ['dexterous hands on unitree_h1', 'gripper on fetch']

    4. COMMON ERRORS TO AVOID:
    - DON'T use multiple tools when one versatile tool can handle
    everything
    - DON'T use single tool when actions need genuinely different
    capabilities
    - DON'T default to 'gripper on fetch' without considering object
    properties
    - DON'T use specialized tools (circle, suction) unless truly
    necessary

    5. PLATFORM CONSIDERATIONS:
    - Mobile platforms (fetch, stompy, anymal_c, unitree_h1): Kitchen
    navigation
    - Fixed platform (panda): Limited workspace, high precision
    - Choose platform based on workspace requirements, not just tool type

    VALIDATION CHECKLIST BEFORE SELECTION:
    - Have I identified all distinct actions required?
    - Does each action need different tool capabilities?
    - Can a single versatile tool handle all requirements safely?
    - Are specialized tools only used when objects demand them?
    - Is the total tool count justified by genuine capability differences
    ?

    IMPORTANT: You must provide your final selection in the following
    format:
    <answer>['tool1', 'tool2']</answer>
    Where 'tool1', 'tool2' are the exact names of the selected end-
    effectors.
    For single tool selection, use: <answer>['tool1']</answer>

    CRITICAL: For this task, you can ONLY choose from the following end-
    effectors: {allowed}. Do NOT select any other tools, even if they
    seem suitable."""
```

Validator Agent. Provides quality control with retry mechanism:

```
    system_prompt = """Available end-effectors:

    {ROBOT_SET}

    ENHANCED VALIDATION CRITERIA:

    1. TOOL COUNT APPROPRIATENESS:
    a) Single-action tasks: Should use EXACTLY ONE tool
    - "place apple"  ONE tool only
    - RED FLAG: Multiple tools for simple placement

    b) Multi-action same-capability tasks: Should use ONE versatile tool
    - "place banana and apple"  ONE versatile tool
    - RED FLAG: Multiple tools when one can handle all objects
```

```
c) Multi-action different-capability tasks: Should use MULTIPLE tools
- "insert bread, activate toaster, place bowl"  TWO tools minimum
- Bread needs delicate manipulation, toaster needs button press
- RED FLAG: Single tool for genuinely different capabilities

2. OBJECT-TOOL MATCHING VALIDATION:
a) Specialized tool usage CHECK:
- 'circle' used ONLY for spherical objects that require it
- 'suction' used ONLY for flat items requiring precision
- 'dexterous hands' used for delicate/complex manipulation
- RED FLAG: Specialized tools used unnecessarily

b) Versatile tool preference CHECK:
- When objects can be handled by general tools, prefer them
- 'gripper on fetch' for mobile pick/place tasks
- RED FLAG: Over-specialization when not needed

3. CAPABILITY-REQUIREMENT MATCHING:
a) Action analysis:
- Delicate manipulation  dexterous hands required
- Button/lever activation  precise gripper or hands
- General pick/place  any gripper suitable
- Heavy lifting  robust claw hands

b) Platform requirements:
- Kitchen navigation  mobile platforms (fetch, stompy, unitree_h1)
- Fixed workspace  panda acceptable
- RED FLAG: Platform mismatch with workspace needs

4. CRITICAL ERROR PATTERNS TO FLAG:
a) Tool quantity errors:
- Multiple tools for single-capability tasks (OVERUSE)
- Single tool for multi-capability tasks (UNDERUSE)

b) Object mismatch errors:
- Hard grippers for soft objects when gentle options available
- Specialized tools when general tools suffice
- Missing specialized tools when objects demand them

c) Platform selection errors:
- Fixed platforms for tasks requiring mobility
- Wrong capability level for task complexity

5. SCORING GUIDELINES (STRICTER CRITERIA):
- 0.9-1.0: Perfect tool count + perfect object matching + optimal
platform
- 0.7-0.8: Correct tool count + good matching + appropriate platform
- 0.5-0.6: Minor count/matching issues + acceptable platform choice
- 0.3-0.4: Major count errors OR poor matching + suboptimal platform
- 0.0-0.2: Wrong tool count AND poor matching AND inappropriate
platform

6. VALIDATION DECISION TREE:
Step 1: Count distinct capabilities needed  Expected tool count
Step 2: Check if each selected tool matches required capability
Step 3: Verify no over-specialization or under-specialization
Step 4: Confirm platform choice matches workspace requirements
Step 5: Score based on how well selection meets all criteria

COMMON FAILURE PATTERNS TO DETECT:
- Using 2+ tools when 1 versatile tool can handle everything
- Using 1 tool when actions genuinely need different capabilities
- Choosing specialized tools without clear object-specific need
- Missing mobile platform for kitchen navigation tasks
```

```
    Format: <answer>['tool1', 'tool2']</answer>
    Where 'tool1', 'tool2' are the exact names of the selected end-
    effectors.

    CRITICAL: For this task, you can ONLY choose from the following end-
    effectors: {allowed}. Do NOT select any other tools, even if they
    seem suitable.

    After your reasoning, please rate the appropriateness of your
    selected end-effectors for this specific task on a scale from 0.0 (
    completely unreasonable) to 1.0 (perfectly reasonable). Return your
    score in the format <score>0.85</score>.
    """
```

C.3 Scoring and Validation

We use a task-specific metric `viki_1` that combines accuracy and format compliance:

- **Base Metric**: `score = 0.9 * acc_reward + 0.1 * format_reward`
- **Smart Matching**: We use `smart_compute_score`, which first evaluates `viki_1`. If not perfect, it parses the `<answer>` list and applies a normalization (e.g., mapping `"gripper on panda"` and `"gripper on fetch"` to `"gripper"`) before re-checking sequence equality.

Relevant implementation excerpts:

```
    original_score = viki_1.compute_score(predict_str, ground_truth_str)
    # If not perfect, normalize and compare element-wise in order
    if all(normalize(pred) == normalize(gt) for pred, gt in zip(pred_list,
     gt_list)):
    format_score = viki_1.format_reward(predict_str)
    return 0.9 * 1.0 + 0.1 * format_score
```

C.4 Model List

Tested models include:

- `gpt-5` accuracy: 0.4400
- `gemini-2.5-pro` accuracy: 0.4250
- `Qwen/Qwen2.5-VL-32B-Instruct` accuracy: 0.2350
- `Pro/Qwen/Qwen2.5-VL-7B-Instruct` accuracy: 0.2900
- `claude-sonnet-4-20250514` accuracy: 0.4150
- `glm-4.5v` accuracy: 0.2600

C.5 Dataset Schema

Each sample provides the following fields used by our runner:

- `prompt`: an array of chat messages; index 1 contains the user message text (prefixed by "Task Description:")
- `images`: optional list with binary bytes; the first item is written to a temporary PNG file for base64 encoding
- `robot`: list of available robots for this scenario; only these determine the end-effector options
- `reward_model.ground_truth`: a stringified Python list (e.g., `"['gripper on panda']"`) representing the reference sequence

The runner maps robots to end-effector option names and builds the system message accordingly. Independent tools (`suction`, `circle`, `stick`) are only included if explicitly present in `robot`.

D HETEROGENEOUS EFFECTOR AGENTS

In this section, we provide a detailed report on the code implementation and other technical details of heterogeneous multi-end effector embodied agents in the ManiSkill simulator (Tao et al., 2024a).

The specific implementation of heterogeneous multi–end effector agents in the simulator is as follows:

1. ***Gripper:*** We directly load the `panda.urdf` file integrated into ManiSkill.
2. ***Suction:*** Since accurately modelling the pressure of a suction cup in simulation remains challenging, we maintain the `panda` model with its right and left fingers permanently closed. Upon invocation of the `open_suction()` interface, if a finger is detected in contact with another object, the contacted object is switched from a dynamic to a kinematic state and is constrained to follow the finger. When the `close_suction()` interface is called, the object's dynamic properties are restored.
3. ***Stick:*** We directly load the `panda_stick.urdf` file integrated into ManiSkill.
4. ***Ring-shaped Gripper:*** First, we designed `panda_circle.urdf`, adapted for ManiSkill, as the URDF model of a ring-shaped gripper. At the same time, the ring-shaped gripper's wrapper inherits from the suction wrapper: the interface methods are replaced by `open_circle()` and `close_circle()`, while the rest of the logic remains unchanged.

D.1 SIMULATOR AGENT WRAPPER

Each sim step, if any finger link touches a valid dynamic rigid body, this function performs a one-shot "attach." It flips the target to kinematic, stores the parent-to-target relative pose via `self._relative_pose = fcomp.pose.inv() * tcomp.pose`, and records internal state; returns `True` on success, `False` otherwise.

```
def grab_once(self) -> bool:
    """Attach a valid dynamic target if any finger link is in contact."""
    if self.is_attached():
        return True
    if not self._finger_links:
        return False

    finger_names = {self._pretty_name(l) for l in self._finger_links}
    for c in self.scene.get_contacts(): # contacts available after each
        ↪ simulation step
        side0, side1 = self._collect_side(c, 0), self._collect_side(c, 1)
        for s_touch, s_other in ((side0, side1), (side1, side0)):
            finger_entity = self._pick_by_name(s_touch, finger_names)
            if not finger_entity:
                continue
            target_entity = self._pick_valid_target(s_other) # dynamic, not
                ↪ an articulation link
            if not target_entity:
                continue

            fcomp = finger_entity.find_component_by_type(physx.
                ↪ PhysxArticulationLinkComponent)
            tcomp = target_entity.find_component_by_type(physx.
                ↪ PhysxRigidDynamicComponent)
            if not (fcomp and tcomp):
                continue

            # Switch to kinematic and record relative pose (parent^-1 *
                ↪ target)
            tcomp.set_kinematic(True)
```

```
            self._attached_comp = tcomp
            self._parent_link_comp = fcomp
            self._relative_pose = fcomp.pose.inv() * tcomp.pose
            return True
    return False
```

Call every step while attached. It drives the target using the stored relation `target_world = parent_world * relative_pose`; writes `new_pose` to the kinematic body.

```
def update_attachment(self):
    """Force target pose = parent_pose * relative_pose (per-step follow).
        ↪ """
    if not self.is_attached():
        return
    new_pose = self._parent_link_comp.pose * self._relative_pose
    self._attached_comp.entity.set_pose(new_pose) # kinematic body is
        ↪ driven externally
```

Detach. Switch the target back to dynamic, zero its linear/angular velocities to prevent bursts, and clear internal state

```
def release(self):
    """Restore dynamic state and zero velocities to avoid bursts on
        ↪ release."""
    if not self.is_attached():
        return
    try:
        self._attached_comp.set_kinematic(False) # back to dynamic
        self._attached_comp.linear_velocity = np.zeros(3)
        self._attached_comp.angular_velocity = np.zeros(3)
    finally:
        self._attached_comp = None
        self._parent_link_comp = None
        self._relative_pose = None
```

D.2 Simulator Record Wrapper

On each env step, coerce per-agent action vectors to fixed target dims (stick = 7, circle/suction/gripper = 8) by truncating/padding, then forward the normalized action to the base recorder; this keeps dataset shapes consistent without changing environment execution.

```
def step(self, action):
    """Normalize per-agent action dims for recording without altering env
        ↪ behavior: - stick: 7D - circle/suction/gripper: 8D"""
    dim_map = self._target_action_dim_map()

    def _fix_vec(vec: np.ndarray, want: int) -> np.ndarray:
        v = np.asarray(vec).reshape(-1).astype(np.float32)
        if v.size == want:
            return v
        if v.size > want:
            return v[:want]
        out = np.zeros((want,), dtype=np.float32)
        out[: v.size] = v
        return out

    if isinstance(self.env.action_space, gym_utils.spaces.Dict):
        # Multi-agent dict: normalize each agent vector to its target dim
        assert isinstance(action, dict), "Expect dict action for multi-
            ↪ agent env"
        norm = {}
        keys = (list(self.env.action_space.spaces.keys())
```

```
            if hasattr(self.env.action_space, "spaces")
            else list(action.keys()))
        for k in keys:
            want = dim_map.get(k, int(np.prod(np.asarray(action[k]).shape)))
            norm[k] = _fix_vec(action[k], want)
    else:
        # Single agent
        want = dim_map.get("__single__", int(np.prod(np.asarray(action).
            ↪ shape)))
        norm = _fix_vec(action, want)

    # Delegate to base recorder (records + steps env)
    return super().step(norm)
```

When flushing, write a complete episode slice into an HDF5 group `"traj_N"`: skip the initial dummy frame, dump observations (gzip for `rgb`/`depth`/`seg`), write per-agent actions (with optional key renaming), flags, states, and rewards; also append JSON episode metadata. Inputs: internal trajectory buffer; output: files on disk plus updated episode index.

```
def flush_trajectory(self,
                     verbose=False,
                     ignore_empty_transition=True,
                     env_idxs_to_flush=None,
                     save: bool = True):
    """Flush a trajectory slice to disk as HDF5 + JSON metadata.Skips
        ↪ first dummy frame; compresses images; preserves fixed action
        ↪ dims."""
    flush_count = 0
    if env_idxs_to_flush is None:
        env_idxs_to_flush = np.arange(0, self.num_envs)

    for env_idx in env_idxs_to_flush:
        start_ptr = self._trajectory_buffer.env_episode_ptr[env_idx]
        end_ptr = len(self._trajectory_buffer.done)
        if ignore_empty_transition and end_ptr - start_ptr <= 1:
            continue
        flush_count += 1

        if save:
            # Create /traj_N
            self._episode_id += 1
            traj_id = f"traj_{self._episode_id}"
            group = self._h5_file.create_group(traj_id, track_order=True)

            # Minimal recursive writer (dicts + arrays), with special
                ↪ handling for vision & actions
            def recursive_add_to_h5py(group: h5py.Group, data: Union[dict,
                ↪ Array], key):
                if isinstance(data, dict):
                    subgrp = group.create_group(key, track_order=True)
                    for k in data.keys():
                        recursive_add_to_h5py(subgrp, data[k], k)
                else:
                    if key in ("rgb", "depth", "seg"):
                        group.create_dataset(
                            key,
                            data=data[start_ptr:end_ptr, env_idx],
                            dtype=data.dtype,
                            compression="gzip",
                            compression_opts=5,
                        )
                    elif group.name.endswith("/actions"):
                        # actions already sliced to (T, D) per-agent below
                        group.create_dataset(key, data=data[start_ptr+1:end_ptr
                            ↪ ], dtype=data.dtype)
```

```
                else:
                    group.create_dataset(
                        key,
                        data=data[start_ptr:end_ptr, env_idx],
                        dtype=data.dtype,
                    )

            # Observations
            if self.record_observation:
                obs_buf = self._trajectory_buffer.observation
                if isinstance(obs_buf, dict):
                    recursive_add_to_h5py(group, obs_buf, "obs")
                elif isinstance(obs_buf, np.ndarray):
                    if self.cpu_wrapped_env:
                        group.create_dataset("obs",
                                             data=obs_buf[start_ptr:end_ptr],
                                             dtype=obs_buf.dtype)
                    else:
                        group.create_dataset("obs",
                                             data=obs_buf[start_ptr:end_ptr, env_idx
                                                 ↪ ],
                                             dtype=obs_buf.dtype)
                else:
                    raise NotImplementedError(f"Unsupported␣obs␣type:␣{type(
                        ↪ obs_buf)}")

            # Episode metadata (JSON sidecar is updated later)
            episode_info = dict(
                episode_id=self._episode_id,
                episode_seed=self.base_env._episode_seed[env_idx],
                control_mode=self.base_env.control_mode,
                elapsed_steps=end_ptr - start_ptr - 1,
            )
            if self.num_envs == 1:
                episode_info.update(reset_kwargs=self.last_reset_kwargs)
            else:
                episode_info.update(reset_kwargs=dict())

            # Slice runtime tensors (skip first dummy frame)
            actions = common.index_dict_array(
                self._trajectory_buffer.action,
                (slice(start_ptr + 1, end_ptr), env_idx),
            )
            terminated = self._trajectory_buffer.terminated[start_ptr+1:
                ↪ end_ptr, env_idx]
            truncated = self._trajectory_buffer.truncated [start_ptr+1:
                ↪ end_ptr, env_idx]

            # Optional agent key normalization (example: rename stick agent
                ↪ key)
            def _rename_agent_key_for_write(k: str) -> str:
                if isinstance(k, str) and k.startswith("panda_stick"):
                    return "panda" if "-" not in k else "panda-" + k.split("-",
                        ↪  1)[1]
                return k

            # Actions
            if isinstance(self._trajectory_buffer.action, dict):
                actions_to_write = {}
                for k, v in actions.items():
                    new_k = _rename_agent_key_for_write(k)
                    if (new_k in actions_to_write) and (new_k != k):
                        new_k = k
                    actions_to_write[new_k] = v
                recursive_add_to_h5py(group, actions_to_write, "actions")
```

```
            else:
                group.create_dataset("actions", data=actions, dtype=np.
                    ↪ float32)

            # Flags
            group.create_dataset("terminated", data=terminated, dtype=bool)
            group.create_dataset("truncated", data=truncated, dtype=bool)

            # Success / fail (optional)
            if self._trajectory_buffer.success is not None:
                end_ptr2 = len(self._trajectory_buffer.success)
                group.create_dataset(
                    "success",
                    data=self._trajectory_buffer.success[start_ptr+1:end_ptr2,
                        ↪  env_idx],
                    dtype=bool,
                )
                episode_info.update(success=self._trajectory_buffer.success[
                    ↪ end_ptr2 - 1, env_idx])
            if self._trajectory_buffer.fail is not None:
                group.create_dataset(
                    "fail",
                    data=self._trajectory_buffer.fail[start_ptr+1:end_ptr,
                        ↪ env_idx],
                    dtype=bool,
                )
                episode_info.update(fail=self._trajectory_buffer.fail[end_ptr
                    ↪  - 1, env_idx])

            # Environment states (if enabled)
            if self.record_env_state:
                recursive_add_to_h5py(group, self._trajectory_buffer.state, "
                    ↪ env_states")

            # Rewards (if enabled)
            if self.record_reward:
                group.create_dataset(
                    "rewards",
                    data=self._trajectory_buffer.reward[start_ptr+1:end_ptr,
                        ↪ env_idx],
                    dtype=np.float32,
                )

            # Update manifest JSON
            self._json_data["episodes"].append(episode_info)
            dump_json(self._json_path, self._json_data, indent=2)

            if verbose:
                print(f"Recorded episode {self._episode_id}")

    # Truncate in-memory buffer to save RAM and advance per-env pointers
    if flush_count > 0:
        self._trajectory_buffer.env_episode_ptr[env_idxs_to_flush] = len(
            ↪ self._trajectory_buffer.done) - 1
        min_env_ptr = self._trajectory_buffer.env_episode_ptr.min()
        N = len(self._trajectory_buffer.done)

        if self.record_env_state:
            self._trajectory_buffer.state = common.index_dict_array(self.
                ↪ _trajectory_buffer.state, slice(min_env_ptr, N))
        self._trajectory_buffer.observation = common.index_dict_array(self.
            ↪ _trajectory_buffer.observation, slice(min_env_ptr, N))
        self._trajectory_buffer.action = common.index_dict_array(self.
            ↪ _trajectory_buffer.action, slice(min_env_ptr, N))
        if self.record_reward:
```

```
            self._trajectory_buffer.reward = common.index_dict_array(self.
                ↪ _trajectory_buffer.reward, slice(min_env_ptr, N))
        self._trajectory_buffer.terminated = common.index_dict_array(self.
            ↪ _trajectory_buffer.terminated, slice(min_env_ptr, N))
        self._trajectory_buffer.truncated = common.index_dict_array(self.
            ↪ _trajectory_buffer.truncated, slice(min_env_ptr, N))
        self._trajectory_buffer.done = common.index_dict_array(self.
            ↪ _trajectory_buffer.done, slice(min_env_ptr, N))
        if self._trajectory_buffer.success is not None:
            self._trajectory_buffer.success = common.index_dict_array(self.
                ↪ _trajectory_buffer.success, slice(min_env_ptr, N))
        if self._trajectory_buffer.fail is not None:
            self._trajectory_buffer.fail = common.index_dict_array(self.
                ↪ _trajectory_buffer.fail, slice(min_env_ptr, N))
        self._trajectory_buffer.env_episode_ptr -= min_env_ptr
```

E Extended Positioning: Comparison with Existing Paradigms

A reasonable concern is that *end-effector selection* can look like (a) classical task-and-motion / grasp-and-regrasp planning with multiple manipulators, (b) VLM-driven *tool use*, or (c) prior heterogeneous-MAS work such as VIKI (Kang et al., 2025). We position our contribution against each.

Vs. classical task-and-motion planning. Classical pipelines plan sequences of grasps/regrasps on a *fixed* embodiment with hand-engineered symbolic or geometric models. Here the embodiment itself becomes the planning variable: the Planner chooses *which* end-effector(s) to use from a library $\mathcal{E}$ for a given task, against a structured Robot Manual that exposes capability, range, and cost. Low-level execution is delegated to imitation-learning policies, not motion-primitive scripts.

Vs. VLM tool use. Tool selection can operate over software APIs or physical tools; robotic tool use also faces payload, geometry, and contact constraints. The Verifier checks constraints represented in the Manual, not geometric feasibility. Our experiments compare heterogeneous effectors with gripper-only configurations. They do not establish an advantage over a gripper equipped with exchangeable tools.

Vs. VIKI and other heterogeneous-MAS work. VIKI (Kang et al., 2025) addresses heterogeneous robots at the high-level planning stage only. We extend this to a complete pipeline: high-level selection *plus* low-level execution *plus* end-to-end evaluation that feeds the planner's choices into the actual policies (Sec. 4.3), rather than only reporting selection accuracy.

F Extended Experiments and Discussion

This appendix provides the full task-spectrum discussion that motivates the effector library, two method overflows ($\pi_{0.5}$ fine-tuning recipe and a scalability argument for the library), and a failure-case study. The policy architecture ablation appears in Sec. 4.5.

F.1 Spectrum of Real-World Robotic Tasks

Real-world manipulation spans a broad distribution of object and scene properties; a single, fixed effector is well-matched to only a slice of it. We organise our library $\mathcal{E}$ around five recurring object/scene properties for which the same gripper is repeatedly outmatched: (i) *thin/flat* (e.g. a card on a table), (ii) *round/cylindrical that escape fingered grasps* (e.g. a vase, a smooth sphere), (iii) *requires insertion or pushing inside a narrow cavity* (e.g. slotting a shuttlecock into a tube), (iv) *requires cooperative manipulation between two arms* (e.g. handover-and-insertion), and (v) all *regular rigid* objects on which a gripper is itself the right choice. We do *not* claim that this list is exhaustive or that the resulting four-effector library $\mathcal{E} = \{\text{Gripper}, \text{Suction}, \text{Stick}, \text{Ring-shaped Gripper}\}$ covers the full distribution of manipulation tasks; it is an *instantiation* that suffices to expose where heterogeneous

composition helps and where it does not (Sec. 4). The next subsection shows how a new effector is added without retraining the Planner.

F.2 Scalability of the Effector Library (Method)

A frequent concern is that the four-effector library is hand-designed and small. By construction, adding a new effector e_{K+1} requires only two steps: (1) add an entry to the Robot Manual describing its skills, action range, and cost; (2) connect a compatible executor (trained as in Sec. 3.2 or supplied externally). New multi-effector combinations require synchronized demonstrations and a task/configuration-specific joint executor. **Crucially, the Planner is not retrained**; the new entry enters the manual at planning time. We do not claim this procedure discovers new effectors automatically; designing physically novel effectors remains an open problem orthogonal to the present work and is discussed in Sec. 5.

F.3 $\pi_{0.5}$ Fine-tuning Recipe

The $\pi_{0.5}$ baseline in the main execution table follows a single-pass fine-tuning recipe: we feed the same global RGB observation and per-arm proprioception as for DP, decode actions via the model's native action head, and use the official fine-tuning protocol on 25/50/75 trajectory budgets. At inference time the policy is rolled out with the *Global-View + Shared-Policy* interface so that, relative to DP, only the policy backbone changes. This isolates the contribution of the backbone class (task-specific imitation vs. generalist VLA) from the observation and dispatch protocol.

F.4 Real-World Task Visualisation

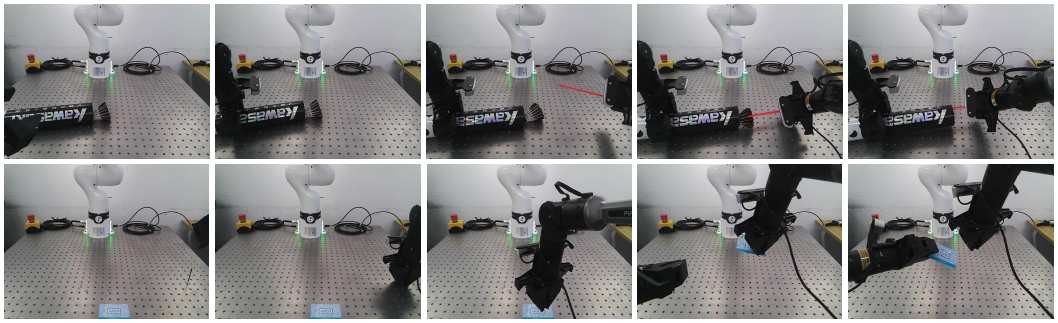

Figure 7: Visualisation of real-world tasks. Top row: Place Shuttlecock. Bottom row: Suction-lift Card.

F.5 Visualisation of Failure Cases

We present the visualisation of failure cases encountered during the Swipe Card task. Two primary scenarios are identified and illustrated in Fig. 8. The first occurs when the end-effector misaligns with the card, preventing proper sweeping and resulting in an unsuccessful lift (top-left). The second arises during the handover, where the end-effector fails to securely grasp the card, often due to insufficient grip force or improper positioning, causing it to fall (bottom-right). These failures provide concrete targets for improvement: alignment precision during pickup and robust regrasping during handover.

G Policy Coverage and Implementation Clarifications

G.1 Executor coverage and role encoding

For each execution task, the original comparison trains two executors: one for the designated heterogeneous configuration and one for its matched gripper-only configuration. The system does not train an executor for every element of the task–effector Cartesian product, and there is no weight sharing across tasks. Table 9 summarizes the executor coverage and demonstration budgets.

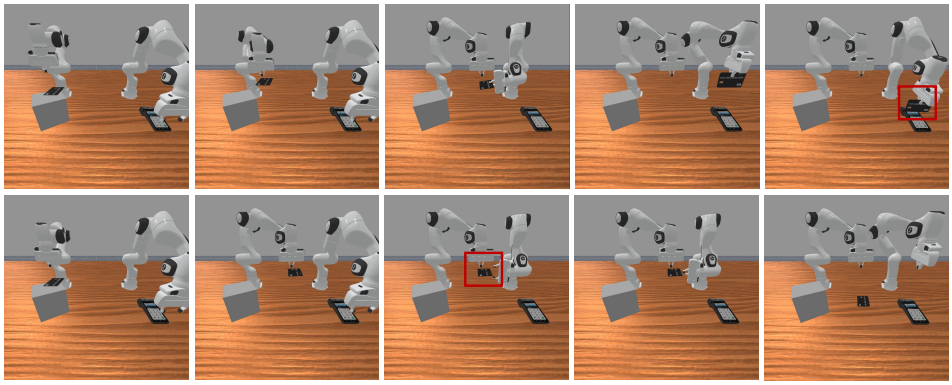

Figure 8: Visualisation of failure cases in simulation. Top row: misalignment during card sweeping, leading to an unsuccessful lift. Bottom row: failure to grasp the card during handover, causing it to fall.

Table 9: Executor coverage. Each row has separately trained heterogeneous and gripper-only executors. Demonstration budgets per configuration: DP and $\pi_{0.5}$, 25/50/75; DP3, 50/100/150.

Task	Heterogeneous configuration	Comparator
Suction-lift Card	Suction	Gripper
Pick Pokéball	Ring-shaped gripper	Gripper
Pick Vase	Ring-shaped gripper	Gripper
Place Shuttlecock	Gripper + stick	Gripper + gripper
Swipe Card	Suction + gripper	Gripper + gripper

Within an executor, "Shared-Policy" means one joint action chunk for the arms, conditioned on global observations and per-arm proprioception in fixed role order. Arm/effector identities are encoded by that ordering and the configuration-specific executor, rather than by a claim of a universal policy over arbitrary embodiments. An unsupported Planner selection is counted as a planning failure, with no silent fallback to an oracle executor.

G.2 Modules and phases

Separation of construction and inference. GPT-4o label proposals and template checks construct RoboMonster-P; RoboBrain and RoboChecker construct execution demonstrations. At test time, the deployed Planner performs analysis and selection, and the Verifier checks the proposal. The Analyzer/Planner/Selector names used in the prompt descriptions denote stages inside this Planner; Validator denotes the Verifier. The resulting selection dispatches one available executor. This separates annotation and demonstration generation from the two-agent inference pipeline.

Table 10 distinguishes data construction, demonstration generation, training, and inference. Analyzer, Planner, and Selector are internal stages of the deployed Planner; Validator is the Verifier. This naming does not define a separate four-agent inference architecture.

G.3 Ground-truth construction

Task templates define the RoboMonster-P ground truth. GPT-4o only proposes a label; the checker rejects template violations, and a human resolves the remaining cases. Model proposals do not independently establish the reference labels.

Table 10: Module roles, interfaces, and phases.

Module	Model / mechanism	Input → output	Phase
Label proposer	GPT-4o	Image, instruction → proposed selection	RoboMonster-P construction
Label checker	Template rules + human resolution	Proposal, constraints → verified label	RoboMonster-P construction
RoboBrain	MLLM-based scheduler	Task, effector identity → primitive sequence	Demonstration generation
RoboChecker	Trajectory validity checks	Trajectory, predicates → accept/reject	Demonstration generation
Planner	MLLM	Observation, instruction, Manual → candidate selection	Test-time planning
Verifier	LLM + Manual constraints	Candidate, Manual → checked selection	Test-time planning
Executor	DP / DP3 / $\pi_{0.5}$	Policy observations → joint action chunk	Imitation learning and rollout

G.4 Motion primitives and synchronization

The primitive library, termination rules, shared recording clock, and handover condition are described in Sec. 3.3. These routines generate synchronized demonstrations; the trained executor produces actions during evaluation.

G.5 Hardware protocol

The ALOHA study uses 50 demonstrations per task and matched initializations for the compared configurations. It evaluates 10 trials each for Suction-lift Card and Place Shuttlecock and 20 trials each for Pick Vase and Swipe Card, totaling 60 trials per system. The Planner runs once at the beginning of each hardware task and dispatches the selected executor. Thus planning is included in the hardware loop, while execution-time replanning is not.

G.6 Contact abstraction and geometric feasibility

The one-shot attachment abstraction is applied identically across the compared simulation configurations. It simplifies parallel-jaw contact and handovers, which limits the conclusions that can be drawn about contact-rich real-world manipulation.

The Planner reasons about semantic capabilities and Manual constraints, without explicit geometric feasibility checking or feedback from the low-level controller. In the main evaluation, Pick Vase reaches 64% end-to-end versus 100% with oracle selection, whereas Place Shuttlecock reaches 54% in both cases. These aggregate gaps locate tasks that warrant further diagnosis; identical aggregate rates alone do not prove that every selection was correct, and the tables do not establish a unique causal decomposition of failures.