CHAIN-OF-PRIMITIVES SHAPES ROBOT-AGENT ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language models (VLMs) increasingly control robots through manipulation primitives, ranging from low-level commands to geometric targets and semantic skills. How these capabilities are exposed to the agent determines which parts of spatial reasoning and control the model must solve itself. Existing systems, however, typically fix this interface by design. We introduce PRIMITIVESUITE, which exposes a common set of manipulation capabilities through three agent-facing interfaces: direct control, geometric specification, and semantic invocation. We find that no single interface is consistently best: the preferred abstraction varies across models, tasks, and even manipulation phases. Rather than prescribing this choice, we allow agents to compose heterogeneous primitives during execution and represent their decisions as a Chain-of-Primitives (COP). Successful COPs provide a natural form of in-context experience, demonstrating how available primitives should be selected, parameterized, and sequenced. COP-based in-context learning (ICL) substantially improves composition, while visual demonstrations alone provide little benefit. With COP-based ICL, GPT-6 Astra reaches 82.4% success on LIBERO-PRO, and Claude Opus 5 reaches 92.3% on ROBOMME, outperforming representative VLA, coding-agent, and VLM-harness baselines. Our results shift the design question from choosing a fixed robot interface to enabling agents to adaptively compose interfaces from experience.

1 INTRODUCTION

Recent VLMs are increasingly used as general-purpose agents for robotic manipulation. They interact with the physical world through executable primitives (Driess et al., 2023; Zitkovich et al., 2023; Fu et al., 2026; Zhang et al., 2026c; Ding et al., 2026). These primitives can expose very different interfaces to the agent. At one extreme, the model directly predicts low-level end-effector commands; at an intermediate level, it specifies geometric targets such as pixels or poses; at a higher level, it invokes semantic skills while an executor handles grounding, motion planning, and control (Dalal et al., 2021; Jiang et al., 2024; Huang et al., 2023; Ahn et al., 2022; Liang et al., 2023). The choice of interface therefore determines which parts of spatial reasoning and control must be solved by the model and which are delegated to the robot system.

This choice can dramatically change the apparent capability of the same model. For example, Claude Opus 4.6 achieves only 3.5% success on a 40-task LIBERO evaluation when directly producing seven-degree-of-freedom (7-DoF) end-effector commands (no model tested exceeds 5.5%), but 76% when supervising a pretrained vision-language-action (VLA) policy (Berman et al., 2026). Existing systems nevertheless typically prescribe the interface manually. Harness VLA, for instance, assigns non-contact motion to analytical primitives and contact-rich interaction to a frozen VLA model (Zhang et al., 2026c). Such hand-designed divisions of labor make the choice of primitive interface for a given model and task a central design decision.

To study this choice systematically, we build PRIMITIVESUITE, a primitive library and evaluation suite that exposes the same robot capabilities at three levels: C0 direct control, C1 geometric primitives, and C2 semantic primitives. We implement it on different benchmarks from sensor observations only, behind one call contract in which the argument form selects the level, so the same library can be tested one level at a time or composed within an episode while the task and robot system stay fixed. Across models, tasks, and manipulation phases, we find that primitive effectiveness

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.

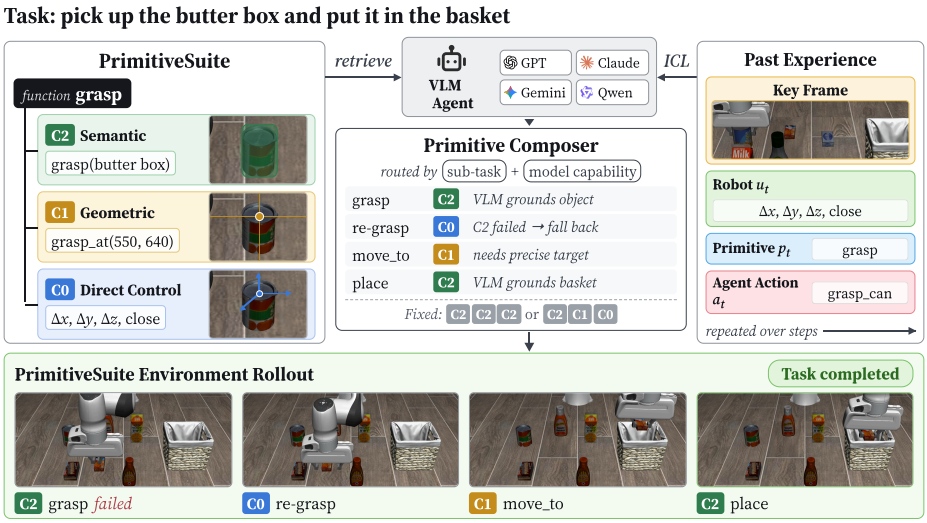


Fig. 1: Primitive-grounded ICL for robot-use agents. A VLM is given a heterogeneous primitive library together with prior successful experience represented as a Chain-of-Primitives (CoP). The context helps the agent determine how to compose primitives for a new task.

varies substantially: there is no universally optimal interface, and even the preferred abstraction can change within a single task.

Our main contributions are:

1. **PrimitiveSuite: a multi-level primitive library that we build, test, and compose.** We implement sensor-only C0/C1/C2 primitive libraries on different benchmarks behind one call contract that supports both fixed-level evaluation and composition across levels, and test each level independently per phase.
2. **A systematic study of primitive-interface dependence.** We show that the preferred primitive interface depends on the model and the manipulation phase within a task. Changing only the agent-facing interface yields success-rate gaps of up to 26.8 percentage points for GPT-6 Astra and 59.4 points for Qwen3.8-27B.
3. **CoP for in-context interface adaptation.** We show that VLM agents can select and compose heterogeneous primitives, and that CoP demonstrations further improve this capability across models and benchmarks. GPT-6 Astra reaches 82.38% success on LIBERO-Pro with CoP-based ICL, and Claude Opus 5 reaches 92.25% on RoboMME, outperforming representative agentic and policy baselines.

2 Related Work

2.1 Primitives for Robotic Agents

Classical robot systems organized control into libraries of parameterized modules, such as trajectory generators and feedback controllers, which a task planner selected and sequenced (Schaal, 2006; Kober & Peters, 2009; Kaelbling & Lozano-Pérez, 2011). Robot agents now act through such modules at different levels of abstraction: direct control (Berman et al., 2026), geometric primitives parameterized by contact points or poses (Dalal et al., 2021; Jiang et al., 2024; Huang et al., 2023), and semantic skills (Ahn et al., 2022; Liang et al., 2023).

Prior work has largely studied primitive selection within a designer-specified action interface. Parameterized policies choose and configure motion primitives from a fixed library (Dalal et al., 2021; Jiang et al., 2024), while language-based agents compose predefined skill or control application programming interfaces (APIs) (Ahn et al., 2022; Liang et al., 2023; Huang et al., 2023). Even when systems combine different executors, their roles are typically prescribed: Harness VLA, for example,

assigns non-contact motion to analytical primitives and local contact-rich interaction to a frozen VLA (Zhang et al., 2026c). ASPIRE expands the available skill library through experience (Lu et al., 2026), and CaP-X compares coding agents under separately specified levels of API abstraction (Fu et al., 2026). These studies leave open how an agent should choose among abstraction levels during execution, which we study together with whether demonstrations of successful primitive chains help agents make this choice in context.

2.2 ICL for Robotic Agents

In-context robot learning typically provides prior experience through demonstrations, from retrieved robot trajectories in RICL to human videos in Zero-WAM (Sridhar et al., 2025; Zhou et al., 2026). Prior work also suggests that how this experience is represented matters: ReAct and Voyager retain actions and execution feedback in context (Yao et al., 2023; Wang et al., 2023), while StellaVLA converts trajectories into structured plans and subgoals (Xu et al., 2026). Still, directly providing demonstrations to robot agents is not reliably helpful: on RoboDojo, one-shot image and text demonstrations reduce GPT-6 Astra's success from 22.9% to 17.9% and 12.9%, respectively (Zhang et al., 2026b). This motivates us to focus on what a demonstration exposes about using the robot interface. We therefore study CoP as the context for robot ICL and compare it against visual demonstrations under the same interface.

3 PrimitiveSuite and Chain-of-Primitives

We first introduce PrimitiveSuite, the multi-level primitive library and evaluation suite we build to expose the same robot capabilities through different agent-facing interfaces. PrimitiveSuite separates the underlying robot execution backend from the interface presented to the agent, allowing the same manipulation capability to be instantiated at different levels of abstraction. We then define the Chain-of-Primitives (CoP) as a trajectory representation within PrimitiveSuite that records how an agent selects, parameterizes, and composes primitives over time.

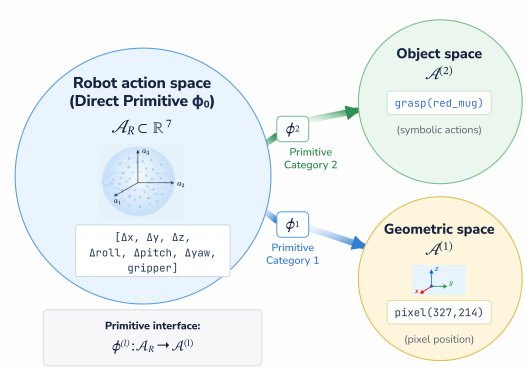


Fig. 2: Agent-facing action spaces induced by different primitive abstractions. $\mathcal{A}_R$ is the robot action space ($\mathcal{U}$ in Sec. 3.1); $\phi^{(l)}$ denotes the abstraction map that induces the category-l interface $\mathcal{A}^{(l)}$, not an executor. Execution runs in the opposite direction, from agent arguments to robot actions, via the executors Φ_p.

3.1 Primitive Abstraction

Let $\mathcal{U}$ denote the underlying robot action space and $\mathcal{P}$ the primitive library exposed to the agent. Each primitive $p \in \mathcal{P}$ has an argument space Θ_p and an executor $\Phi_p : \mathcal{O} \times \Theta_p \to \mathcal{U}^{H_p}$, which maps the current observation and primitive arguments to a sequence of H_p executable robot actions. Given observations $o_{\le t}$ and task goal g, the agent selects a primitive and its arguments, and the executor produces the corresponding actions:

$$(p_t, \theta_t) = \pi(o_{\le t}, g; \mathcal{P}), \quad \theta_t \in \Theta_{p_t}, \qquad u_{t:t+H_{p_t}-1} = \Phi_{p_t}(o_t, \theta_t).$$

With a heterogeneous primitive library, the agent's action space is $\mathcal{A}_{\mathcal{P}} = \bigcup_{p\in\mathcal{P}} (\{p\} \times \Theta_p)$.

3.2 Primitive Interface Categories

We categorize primitives by the information that the agent must provide for execution, which determines where responsibility for spatial reasoning and control lies.

Formally, let $\kappa : \mathcal{P} \to \{0, 1, 2\}$ denote the interface category of a primitive. We distinguish three representative categories (Fig. 3).

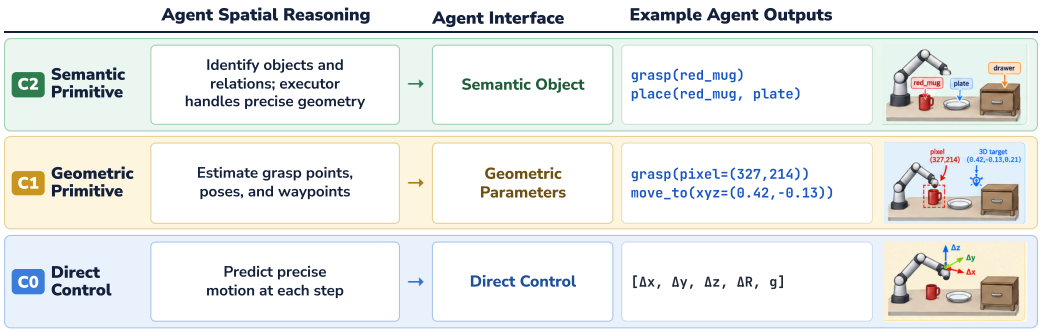


Fig. 3: Three primitive interface categories with different allocations of responsibility between the agent and executor.

C0: Direct Control. The agent directly specifies low-level robot commands, so that $\mathcal{A}^{(0)} \simeq \mathcal{U}$. For a 7-DoF end-effector controller, an action may take the form $a_t^{(0)} = [\Delta x, \Delta y, \Delta z, \Delta\text{roll}, \Delta\text{pitch}, \Delta\text{yaw}, \text{gripper}]$.

C1: Geometric Primitives. The agent specifies an explicit geometric target, $\theta_t \in \mathcal{G}$, such as a pixel, 3D point, pose, waypoint, or region (Huang et al., 2023; Liang et al., 2023; Fu et al., 2026). Example calls include `grasp(pixel=(327,214))` and `move_to(xyz=(0.42,-0.13,...))`. The executor converts the target into robot motion through components such as coordinate transformation, inverse kinematics, motion planning, and low-level control. Precise spatial grounding remains with the agent, while motion generation and control are handled by the executor.

C2: Semantic Primitives. The agent specifies a semantic operation and entity-level arguments, $\theta_t \in \mathcal{M} \times \mathcal{E}^*$, for example `grasp(red_mug)` or `place(red_mug, plate)`. The executor resolves the referenced entities into geometric targets and executes the corresponding motion, leaving the agent only semantic task decisions.

The three categories progressively shift geometric grounding and control from the agent to the executor. This changes both the reasoning burden on the agent and its dependence on the primitive implementation, making the preferred abstraction model- and task-dependent (Berman et al., 2026; Zhang et al., 2026c).

3.3 Building PrimitiveSuite Across Benchmarks

We implement PrimitiveSuite on LIBERO-Pro and RoboMME: four reusable capabilities exposed at up to three levels (Tab. 1), with each environment supplying its own controller backend, tolerances, stop predicates, and arm bindings. (i) Same executor, different responsibility. A C1 `grasp` back-projects the model's pixel with depth; a C2 `grasp` first grounds the name with SAM 3 (Carion et al., 2025) masks and a VLM selection. No executor reads simulator ground truth. (ii) One contract. The argument form selects the level (name: C2; pixel or point: C1; displacement: C0), and levels share executor state, so a C1 `grasp` can precede a C2 `place`. One library thus serves both fixed-level settings and router-free Composition. (iii) Testing. We test each level over full episodes, each primitive per phase (Fig. 4b), and the executor by replaying successful CoPs.

3.4 CoP Representation

Since robot tasks typically require multiple primitive invocations, we represent an agent trajectory as a CoP,

$$\mathcal{C} = [(p_1, \theta_1), (p_2, \theta_2), \ldots, (p_T, \theta_T)],$$

where p_t denotes the selected primitive and θ_t its arguments. The primitive sequence $p_{1:T}$ captures planning and composition, while $\theta_{1:T}$ specifies the control, geometric, or semantic information

Tab. 1: PRIMITIVESUITE interface schema. Reusable manipulation capabilities are exposed through up to three agent-facing abstraction levels. A slash (/) denotes alternative inputs, semicolons (;) denote jointly provided inputs, and brackets ([]) denote optional inputs.

Primitive	Function	C2 inputs	C1 inputs	C0 inputs
Move	Move the end effector or robot base.	Object name	xyz / pixel / displacement / path, [body: arm / base]	dxyz, drpy, gripper, steps
Grasp	Acquire an object with the gripper.	Object name	xyz / pixel, [yaw], [mode]	—
Place	Release a held object at a target.	Object name, mode	xyz / pixel, mode, [height], [depth]	—
Actuate	Apply contact motion without grasping.	Object name, mode	xyz / pixel, mode, direction	—

required to execute each primitive. Because a COP may mix interface categories, a model can retain fine-grained geometric control where its spatial reasoning is reliable and delegate grounding elsewhere, so a COP records both the task-level plan and the model-specific allocation of responsibility.

4 PRIMITIVE-GROUNDED ICL

Few-shot prompting lets models infer task structure from examples (Liang et al., 2023; Singh et al., 2023; Cheng et al., 2026), but a demonstration can help a robot agent only if the agent can relate the demonstrated behavior to the primitives it has at test time.

We therefore augment demonstrations with their corresponding COP, expressed through the same primitive interface used at test time. Given K demonstrations, the agent predicts the next primitive and its arguments as

$$\pi(p_t, \theta_t \mid o_{\leq t}, g, \mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(K)}),$$

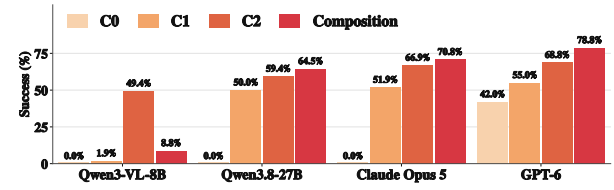
where each demonstration $\mathcal{D}^{(i)}$ is characterized by its **source** and **representation**. The source is the successful rollout the demonstration is taken from: an earlier episode of the same agent or of another model, never of the same task or seed as the evaluated episode. The representation is what of that rollout the agent sees: visual keyframes, the primitive calls alone, or the full COP with each call's returned result and rationale (App. A.2).

5 EMPIRICAL MOTIVATION: PRIMITIVE INTERFACES AND COMPOSITION SHAPE ROBOT USE

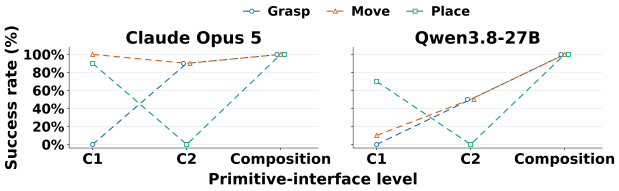
Experimental setup. We evaluate 80 LIBERO-Pro tasks (T and S splits), each from ten official initial states, giving 800 paired episodes per condition with shared environment seeds. Task, observations, executor, model, and budget are fixed; only the exposed interface varies. A fixed-interface setting offers one category (C0, C1, or C2).

5.1 PRIMITIVE INTERFACES AND MODEL-DRIVEN COMPOSITION

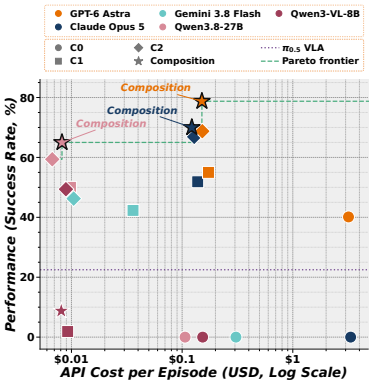
RQ1: How much does the primitive interface affect robot use? Substantially (Fig. 4a). Changing only the interface moves Qwen3-VL-8B from near zero at C0 and 1.9% at C1 to 49.4% at C2, and Qwen3.8-27B from 0.0% at C0 to 50.0% at C1 and 59.4% at C2; Claude Opus 5 and GPT-6 Astra also vary markedly. Measured robot-use capability thus depends on which decisions the interface leaves to the model. The dependence also varies within an episode. Testing GRASP, MOVE, and PLACE phase by phase (Fig. 4b), C1 fails on GRASP and C2 on PLACE for Claude Opus 5, and Qwen3.8-27B shows a similar split, whereas Composition succeeds on all three phases for both. A single interface imposes one agent–executor split on phases that need different reasoning.



(a) Fixed interfaces vs. Composition.



(b) Interface sensitivity across manipulation phases.



(c) Performance–cost trade-off.

Fig. 4: **Capable agents benefit from composing primitive interfaces.** (a) Model-driven composition can outperform any single fixed interface. (b) Interface sensitivity differs across manipulation phases. (c) Composition evaluation under the performance–cost setting.

Tab. 2: ICL context ablation (Qwen3-VL-8B, Composition).

Context	Success
None	8.8%
Video only	11.3%
Calls only	47.5%
Video + call chain	51.9%
Call + result chain	**53.8%**

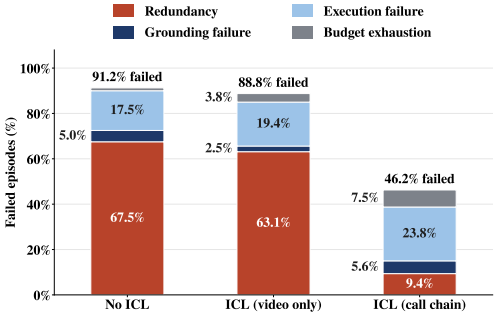


Fig. 5: Failure modes under different ICL contexts (taxonomy in App. A.3).

RQ2: Can the agent compose primitive interfaces by itself? For capable agents, yes. Composition raises the best fixed-interface result of Qwen3.8-27B from 59.4% to 64.5%, Claude Opus 5 from 66.9% to 70.8%, and GPT-6 Astra from 68.8% to 78.8%, while staying on or near the performance–cost frontier (Fig. 4c). Exposing more primitives is not enough, however: Qwen3-VL-8B drops from 49.4% at C2 to 8.8% under Composition, so orchestration is itself an agent capability. The question thus shifts from finding the best interface to helping the agent construct an effective CoP.

5.2 CoP Is the Critical Context for Robot ICL

RQ3: What information from successful experience best supports primitive composition? We vary only how retrieved demonstrations are represented. They are successful Composition rollouts of Claude Opus 5, replayed through the same executor and never from the query task or its perturbed variant (App. A.2). Visual keyframes alone barely help (11.3% vs. 8.8%; Tab. 2), the primitive calls alone reach 47.5%, and the full CoP with returned results and rationales is best at 53.8%.

The key signal is the procedural trace, not visual replay. Fig. 5 shows what it teaches: without ICL the dominant failure is redundancy, repeating a primitive whose effect already holds. Video-only ICL leaves it nearly unchanged (67.5% $\rightarrow$ 63.1%), whereas CoP ICL cuts it to 9.4% without reducing grounding, execution, or budget failures, so CoP improves composition rather than execution.

6 EXPERIMENTAL RESULTS

6.1 EXPERIMENTAL SETUP

Benchmarks and models. PRIMITIVESUITE evaluates two complementary robotic manipulation benchmarks: LIBERO-Pro (Zhou et al., 2025) and RoboMME (Dai et al., 2026). These benchmarks span diverse manipulation settings, including robustness to object, spatial, and task perturbations, language-conditioned manipulation requiring common-sense and world knowledge, and long-horizon tasks requiring reasoning over interaction history. On LIBERO-Pro, we evaluate each task from ten official initial states, each with a fixed environment seed shared across conditions. RoboMME contains 16 tasks in four suites (Counting, Permanence, Reference, and Imitation; four tasks each); we evaluate each task on all 50 test episodes (800 in total), the same episodes as the published baselines. We report the average task success rate. We consider six representative multimodal foundation models spanning both proprietary and open-weight families: GPT-6 Astra (OpenAI, 2026a), GPT-5.4 (OpenAI, 2026b), Claude Opus 5 (Anthropic, 2026a), Gemini 3.8 Flash (Doshi & Popa, 2026), Qwen3.8-27B (Qwen Team, 2026), and Qwen3-VL-8B (Qwen Team, 2025). Unless otherwise specified, all models receive the same observations, task specification, primitive interface, demonstration context, and episode budget, enabling controlled comparisons of how model capability interacts with primitive abstraction.

Baselines. We compare against end-to-end action models, coding agents, and VLM-based robot harnesses. VLA/WAM baselines are OpenVLA (Kim et al., 2024), $\pi_{0.5}$ (Black et al., 2025), StellaVLA (Xu et al., 2026), and OpenWAM (Wang et al., 2026); their +ICL variants additionally condition on video demonstrations of successful rollouts. Coding agents are CaP-Agent0 from CaP-X (Fu et al., 2026), a training-free Code-as-Policy agent that composes perception and control primitives over multiple turns; RATs (Zhang et al., 2026a), which acquires and reuses executable skills; and ASPIRE (Lu et al., 2026). VLM harnesses are Harness VLA (Zhang et al., 2026c), which pairs a frozen VLA with analytic primitives and agentic planning, configured to the benchmark episode budget; and VoLoAgent (Chen et al., 2026), which orchestrates a VLA/WAM with perception and manipulation tools. On RoboMME, we compare with memory-augmented policies: $\pi_{0.5}$ and FrameSamp+Modul, the strongest of the 14 $\pi_{0.5}$-based memory variants released with the benchmark (Dai et al., 2026); MemER (Sridhar et al., 2026); PonderPounce (Choi et al., 2026), at $1\times$ and $9\times$ training data; and SimpleMemVLA (Yin et al., 2026). Results for the first three are from Dai et al. (2026) and the others from their papers; all use 50 episodes per task.

ICL setting. For each evaluation episode, we provide up to $k = 2$ successful demonstrations from different tasks of the same benchmark as shots. Each demonstration is a successful Composition rollout, shown as the full CoP: the task instruction and the ordered calls, each with its returned result and a one-sentence rationale. To prevent leakage, no demonstration shares the same task or the same seed as the evaluation episode.

6.2 MAIN RESULT

RQ4: Does PRIMITIVESUITE support strong performance across model scales? Yes. The same primitive library, unchanged across backbones, lets models from 8B to frontier scale reach strong success on LIBERO-Pro, and lets both open-weight and proprietary models do so on RoboMME. On LIBERO-Pro (Tab. 3), GPT-6 Astra reaches 78.75% with composition alone, above Harness VLA with the same backbone (74.63%) and ASPIRE (61.38%). The open-weight Qwen3.8-27B reaches 64.50%, above every baseline except Harness VLA, and 77.38% with CoP ICL, above Harness VLA as well. Even Qwen3-VL-8B reaches 49.4% at the fixed C2 level (Sec. 5.1) and 53.75% with CoP ICL, above VoLoAgent with GPT-6 Astra (44.00%) and all VLA/WAM baselines. On RoboMME (Tab. 4), Claude Opus 5 reaches 92.13% without any RoboMME training data, above the strongest memory-augmented policy, SimpleMemVLA (88.25%), and PonderPounce trained on $9\times$ data (75.54%); Qwen3.8-27B (58.88%) already exceeds all 14 memory variants released with the benchmark. The library thus supports strong performance across model scales on LIBERO-Pro and across model families on RoboMME.

Fig. 6 shows how Claude Opus 5 uses the library on LIBERO-Pro. Composition mixes levels (59% C2, 41% C1 calls) and adapts the mix to the suite, from 98% C2 on Object to about half C1 on

Tab. 3: Performance on the eight LIBERO-Pro evaluation splits. We report task success rate (%). "+ICL" denotes the corresponding method with ICL. **Bold** and underlined entries indicate the best and second-best results in each column, respectively.

Method	Spatial		Object		Goal		Long		Overall	Δ ICL
	T	S	T	S	T	S	T	S		
VLA / WAM										
OpenVLA (Kim et al., 2024)	0	0	0	0	0	0	0	0	0.00	–
$\pi_{0.5}$ (Black et al., 2025)	55	53	10	19	17	29	17	6	25.75	–
StellaVLA (Xu et al., 2026)	27	9	2	0	3	1	12	0	6.75	–
+ICL	1	1	0	0	11	0	9	0	2.75	−4.00
OpenWAM (Wang et al., 2026)	44	35	10	10	20	38	21	18	24.50	–
+ICL	14	9	7	6	13	12	2	0	7.88	−16.63
Coding Agents										
CaP-Agent0 (Fu et al., 2026)	14	12	18	22	17	26	3	6	14.75	–
RATs (Zhang et al., 2026a)	31	29	63	61	36	43	22	13	37.25	–
ASPIRE (Lu et al., 2026)	51	60	**98**	95	81	45	38	23	61.38	–
VLM Harness										
Harness VLA + GPT-6 Astra (Zhang et al., 2026c)	73	49	95	**100**	82	78	**81**	39	74.63	–
VoLoAgent + GPT-6 Astra (Chen et al., 2026)	62	43	68	77	39	31	22	10	44.00	–
Primitive Composition (Ours)										
Qwen3-VL-8B	0	0	11	7	21	22	9	0	8.75	–
+ICL	76	44	92	83	48	42	22	23	53.75	+45.00
Qwen3.8-27B	73	72	92	**100**	57	68	27	27	64.50	–
+ICL	85	74	96	**100**	62	83	43	**76**	77.38	+12.88
Claude Opus 5	78	83	93	**100**	62	78	48	24	70.75	–
+ICL	91	78	88	**100**	86	87	52	22	75.50	+4.75
GPT-6 Astra	87	73	93	**100**	92	**88**	58	39	78.75	–
+ICL	**98**	**86**	94	**100**	**93**	87	59	42	**82.38**	+3.63

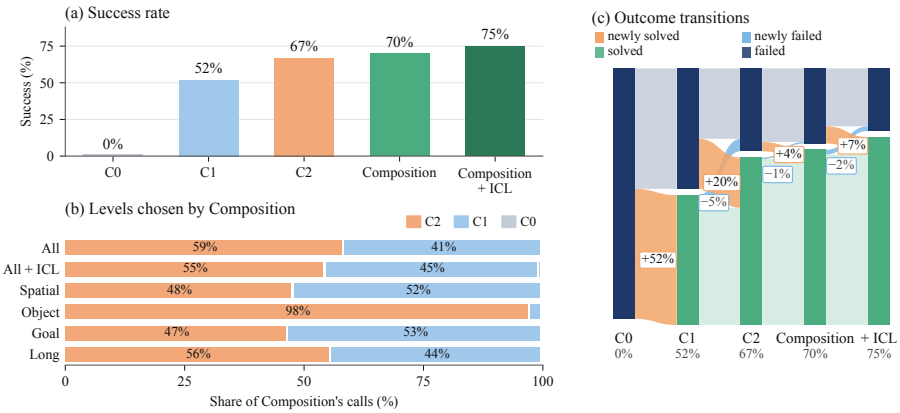


Fig. 6: **Level usage and gains of Claude Opus 5 on LIBERO-Pro.** (a) Success per setting. (b) Share of Composition's calls per level. (c) Paired episodes newly solved or failed relative to the previous setting.

Spatial and Goal; C0 is almost never chosen. Each step is mostly net gain: Composition solves 4% of episodes anew over C2 while losing 1%, and CoP ICL adds 7% while losing 2%.

RQ5: Can CoP demonstrations teach and refine primitive composition? Yes, most when the agent's own orchestration is weak. On LIBERO-Pro, ICL improves every composition agent, from +3.63 points (GPT-6 Astra) to +45.00 (Qwen3-VL-8B, 8.75%→53.75%): the model that fails to discover a composition from scratch recovers most of the gap once given a successful CoP. RoboMME shows the same pattern: Qwen3.8-27B and GPT-5.4 gain +3.25 and +4.75, whereas Claude Opus 5, already at 92.13%, changes by only +0.13.

Tab. 4: Success rate (%) on the four RoboMME suites. "+ICL" denotes our CoP-based in-context learning. Δ ICL reports the change in overall success relative to the corresponding non-ICL setting. **Bold** and underlined entries indicate the best and second-best results in each column, respectively. Baseline results are taken from prior evaluations on RoboMME: $\pi_{0.5}$, MemER, and FrameSamp+Modul are reported by Dai et al. (2026); PonderPounce and SimpleMemVLA are from Choi et al. (2026) and Yin et al. (2026), respectively. All evaluations use 50 episodes per task; each suite contains four tasks (200 episodes), for a total of 800 test episodes. $1\times$/$9\times$ denote the amount of RoboMME training data.

Method	Counting	Permanence	Reference	Imitation	Overall	Δ ICL
VLA / Memory Baselines						
$\pi_{0.5}$ (Black et al., 2025)	28.78	17.00	17.17	8.78	17.93	–
MemER (Sridhar et al., 2026)	48.83	53.17	38.00	29.50	42.38	–
FrameSamp+Modul (Dai et al., 2026)	65.22	25.11	36.33	51.39	44.51	–
PonderPounce ($1\times$) (Choi et al., 2026)	74.67	62.83	72.17	33.67	60.83	–
PonderPounce ($9\times$) (Choi et al., 2026)	81.33	80.17	**92.67**	48.00	75.54	–
SimpleMemVLA (Yin et al., 2026)	91.50	**96.00**	82.50	83.00	88.25	–
Primitive Composition (Ours)						
Qwen3.8-27B	71.50	62.50	49.50	52.00	58.88	–
+ICL	73.50	65.50	54.50	55.00	62.13	+3.25
GPT-5.4	42.00	59.00	50.00	50.00	50.25	–
+ICL	49.50	64.00	50.50	56.00	55.00	+4.75
Claude Opus 5	**95.50**	88.00	91.00	**94.00**	92.13	–
+ICL	93.50	89.50	92.50	93.50	**92.25**	+0.13

In contrast, adding demonstrations to existing policies hurts: the ICL variants of StellaVLA and OpenWAM drop from 6.75% to 2.75% and from 24.50% to 7.88% on LIBERO-Pro. Together with RQ3, this indicates that the benefit comes not from additional context but from the decision structure a CoP exposes—which primitive to invoke, at which abstraction, and how to compose successive calls.

7 Discussion and Limitations

Our study focuses on a controlled set of agent-facing interfaces and contextual representations. The three categories of PrimitiveSuite (direct control, geometric specification, and semantic invocation) are not an exhaustive taxonomy of robot interfaces, but a common abstraction for isolating how interface granularity affects robot reasoning. Likewise, our ICL analysis focuses on procedural experience represented as CoP; other structured context, such as learned skills, subtask plans, or recovery traces, is complementary and may bring further gains. Extending the framework to such interfaces and contexts is a direction for future work.

8 Conclusion

We study how the interface between a VLM agent and robot primitives shapes both robot-use capability and in-context adaptation. With the PrimitiveSuite library we build, we show that direct-control, geometric, and semantic interfaces impose different reasoning requirements, and that no single abstraction is optimal across models, tasks, or manipulation phases. Capable agents can instead compose heterogeneous interfaces during execution, recorded as a CoP, and successful CoPs used as in-context demonstrations teach them how to select, parameterize, and compose primitives. These results suggest that robot capability depends not only on the underlying model, but also on how reasoning is translated into action through the agent–primitive interface.

Reproducibility Statement

We provide the experimental code in the supplementary material. App. A specifies the evaluation protocols (80 LIBERO-Pro tasks from ten official initial states each, with paired environment seeds; 16 RoboMME tasks on all 50 test episodes each), the step budgets, the ICL demonstration sources, retrieval and presentation, the failure taxonomy, and the API cost accounting. App. B gives the key parts of the system prompt; the full prompt is included with the code, and App. C gives concrete C1 and C2 calls with their executor routines.

Ethics Statement

All reported experiments are conducted in simulation; the evaluated agents do not control physical robots. Deploying these agents on hardware would require additional safeguards, including workspace constraints, collision monitoring, and human oversight, because an incorrect primitive call could damage objects or cause injury. Our simulation results do not establish the safety of physical deployment.

AI Use Statement

We used generative AI tools to provide feedback on technical formulations and qualitative data analysis, to assist with method implementation, to edit manuscript text for clarity and readability, to suggest improvements to presentation and structure, and to assist with literature search and verification. We did not use generative AI to generate synthetic datasets, develop theoretical models, write proofs, propose hypotheses, translate content, clean or reformat datasets, or interpret experimental results. We reviewed all AI-assisted work and manually verified technical claims, citations, and suggested revisions against the original sources and our own experiments. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. URL https://arxiv.org/abs/2204.01691.

Alibaba Cloud. Model studio model pricing. https://www.alibabacloud.com/help/en/model-studio/model-pricing, 2026. International (Singapore) deployment. Accessed: 2026-09-25.

Anthropic. Introducing Claude Opus 5. Anthropic, July 2026a. URL https://www.anthropic.com/news/claude-opus-5. Accessed: 2026-09-25.

Anthropic. Pricing – Claude developer platform. https://platform.claude.com/docs/en/about-claude/pricing, 2026b. Accessed: 2026-09-25.

Shmuel Berman, Michael Ilie, Jia Deng, and Daniel Freeman. Claude plays robotics. Anthropic Research, July 2026. URL https://www.anthropic.com/research/claude-plays-robotics. Accessed: 2026-09-25.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, et al. SAM 3: Segment anything with concepts. *arXiv preprint arXiv:2511.16719*, 2025. URL https://arxiv.org/abs/2511.16719.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, Jonathan Tremblay, and Valts Blukis. Volo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026. URL https://arxiv.org/abs/2606.07723.

Dongzhou Cheng, Taoran Yi, Ye Fang, Xingwu Zhang, Fan Feng, Yixuan Li, Gengxiong Zhuang, Rongze Wang, Shuai Yang, Wei Song, Weizhi Xue, Minyan Wu, Jie Gui, Jiaqi Wang, and Tong Wu. In-context robot learning with VLM agents. *arXiv preprint arXiv:2609.19138*, 2026. URL https://arxiv.org/abs/2609.19138.

Suhwan Choi, Jaeyoon Jung, Sungkyung Kim, Yunsung Lee, and Youngjae Yu. PonderPounce: A pretrained MLLM as an episode context engine for robot control. *arXiv preprint arXiv:2608.24115*, 2026. URL https://arxiv.org/abs/2608.24115.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and understanding memory for robotic generalist policies. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.04639. Oral presentation.

Murtaza Dalal, Deepak Pathak, and Ruslan Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21847–21859, 2021.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence, 2026. URL https://arxiv.org/abs/2608.16590.

Tulsee Doshi and Raluca Ada Popa. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber. Google Blog, September 2026. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/. Accessed: 2026-09-25.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Jim Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

Google. Gemini developer API pricing. https://ai.google.dev/gemini-api/docs/pricing, 2026. Accessed: 2026-09-25.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

Bowen Jiang, Yilin Wu, Wenxuan Zhou, Chris Paxton, and David Held. HACMan++: Spatially-grounded motion primitives for manipulation. In *Robotics: Science and Systems*, 2024. URL https://arxiv.org/abs/2407.08585.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pp. 1470–1477. IEEE, 2011. doi: 10.1109/ICRA.2011.5980391.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246.

Jens Kober and Jan Peters. Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pp. 2112–2118. IEEE, 2009.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272.

OpenAI. GPT-6 Astra: A new generation of intelligence. OpenAI, September 2026a. URL https://openai.com/index/gpt-6-astra/. Accessed: 2026-09-25.

OpenAI. GPT-5.4. OpenAI API model documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.4. Accessed: 2026-09-25.

OpenAI. API pricing. https://developers.openai.com/api/docs/pricing, 2026c. Accessed: 2026-09-25.

Qwen Team. Qwen3-VL-8B-Instruct. Hugging Face model card, 2025. URL https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct. Accessed: 2026-09-25.

Qwen Team. Qwen3.8-27B. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B. Accessed: 2026-09-21.

Stefan Schaal. Dynamic movement primitives: A framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pp. 261–280. Springer, 2006. doi: 10.1007/4-431-31381-8_23.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317. URL https://arxiv.org/abs/2209.11302.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling up memory for robotic control via experience retrieval. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.20328.

Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. RICL: Adding in-context adaptability to pre-trained vision-language-action models. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 5022–5038. PMLR, 2025. URL https://proceedings.mlr.press/v305/sridhar25a.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Yuran Wang, Siqiao Huang, Mingleyang Li, Chenhao Zhang, Jiaqi Liang, Weiyang Jin, Yue Chen, Xuemin Chi, Donghao Zhou, Qize Yu, Yu-Kai Wang, Yuhan Rui, Shenzhe Yao, Zhen Yuan, Zhenhao Shen, Kefei Zhu, Zijie Zhu, Ning Gao, Xiaowei Chi, Guanqi He, Shanghang Zhang, Hao Dong, Lin Shao, and Hang Zhao. Openwam: An open, modular exploration towards systematic world-action model pretraining. *arXiv preprint arXiv:2609.07398*, 2026. URL https://arxiv.org/abs/2609.07398.

Siyu Xu, Yunke Wang, Zijian Wang, Dihao Zhu, Chenghao Xia, Chengbin Du, Daochang Liu, Tao Huang, and Chang Xu. StellaVLA: In-context structured demonstration for generalizable vision-language-action models, 2026. URL https://arxiv.org/abs/2608.11671.

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*, 2023. URL https://arxiv.org/abs/2210.03629.

Cheng Yin, Wang Xu, Junpeng Yang, Sikyuen Tam, Hanyu Liu, Yuan Yao, Xiangrui Zeng, Junbo Cui, Yequan Wang, Zhouping Yin, and Yankai Lin. SimpleMemVLA: A simple but effective native-video memory for vision-language-action models. *arXiv preprint arXiv:2609.05533*, 2026. URL https://arxiv.org/abs/2609.05533.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, Zirui Wang, Roei Herzig, Ken Goldberg, Yutong Bai, David M. Chan, Ion Stoica, Angjoo Kanazawa, Jiahui Lei, Haiwen Feng, and Trevor Darrell. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026a. URL https://arxiv.org/abs/2606.19419.

Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, Kailun Su, Weiyang Jin, Wenhao Chai, Haotian Liang, Zhiyang Dou, Yue Chen, and Tianxing Chen. An unexpected robot policy: Early evaluations of GPT-6 Astra on RoboDojo and beyond. https://robodojo-benchmark.com/report/gpt-6-astra-eval, 2026b. Technical report.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026c. URL https://arxiv.org/abs/2607.08448.

Jiaming Zhou, Qihang Zhang, Gangwei Xu, Cunxin Fan, Yujie Zhao, Ruilin Wang, Yiming Luo, Shuai Yang, Xing Zhu, Yujun Shen, Junwei Liang, and Yinghao Xu. Zero-WAM: In-context world-action modeling from human videos for open-ended task generalization, 2026. URL https://arxiv.org/abs/2608.26103.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025. URL https://arxiv.org/abs/2510.03827.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Experimental Details

A.1 LIBERO-Pro Protocol

We use all eight LIBERO-Pro evaluation splits: Spatial, Object, Goal and Long, each under its task (T) and position-swap (S) perturbation (the `_TASK` and `_SWAP` suites of the official release). Each split contains ten tasks, giving 80 tasks in total. Every task is run from ten official initial states, which yields 800 episodes per condition. Each episode is limited to the official control-step budget of its suite: 220 (Spatial), 280 (Object), 300 (Goal) and 520 (Long). Success is determined solely by the simulator's goal predicates at the end of the episode.

The BDDL files and initial states are frozen and hash-checked before each campaign. Each (task, initial state) cell uses a fixed environment seed, $\mathrm{sha256}(\mathrm{suite}/\mathrm{task_idx}/\mathrm{init})$, shared by all interface and context conditions. Every comparison between conditions is therefore paired, and we test differences with exact McNemar tests on the paired episodes. Each campaign also freezes its code snapshot, primitive contracts and system prompt; conditions within a campaign differ only in the variable under study.

A.2 ICL Context Ablation

The context ablation in Tab. 2 uses Qwen3-VL-8B-Instruct under Composition on the 800 LIBERO-Pro episodes described above.

Demonstration source. Demonstrations come from a separate zero-shot Composition run of Claude Opus 5 under the identical protocol (70.0% success). From its successful episodes we keep at most two per task, preferring the fewest control steps, giving 65 episodes over 34 tasks. Each episode is replayed through the frozen executor of the ablation campaign, issuing the recorded calls in order. An episode is kept only if the replay succeeds again: 61 of 65 replays succeed, all with exactly the original step count, leaving 61 demonstrations over 33 tasks. Every condition shows these replayed episodes, so the images and the call chain of a demonstration always describe the same rollout.

Retrieval. For each query we retrieve $k = 2$ demonstrations from different tasks by weighted overlap of instruction words. Demonstrations derived from the same base task as the query, that is, its task and its swap variant, are never shown. Two of the 80 queries retrieve no demonstration. The retrieved set is frozen per query before evaluation and verified in every episode.

Presentation. Demonstrations are placed in the first user message, between the task instruction and the initial observation, and remain in context for the whole episode. Each is introduced by its instruction and total step count. The text formats warn that pixel and point values refer to the demonstration's scene. The five conditions differ only in the content of each demonstration (Tab. 5); model, prompt, seeds, executor and code snapshot are shared, for 4,000 episodes in total.

Tab. 5: Content of each demonstration in the ICL context ablation.

Condition	Content of each demonstration
None	no demonstration
Video only	start frame and the frame after each motion call (front and wrist view, 512 px, at most six per demonstration); no calls, arguments or results
Calls only	ordered calls (`fn` and arguments), without results or rationales
Video + call chain	the call chain with the same keyframes interleaved after the calls that produced them
Call + result chain	ordered calls, each with its returned result (status, steps used, unmet goals) and a one-sentence rationale

Statistics. Call + result chain versus no demonstration is 372 to 12 on paired episodes ($p \approx 9 \times 10^{-94}$); video only versus no demonstration is 26 to 6 ($p \approx 5 \times 10^{-4}$).

A.3 Failure Taxonomy

For Fig. 5, each failed episode is assigned one primary cause.

- **Redundancy.** The agent repeats work that is already done.
- **Grounding failure.** A referenced object or target is not found.
- **Execution failure.** A manipulation does not achieve its intended effect.
- **Budget exhaustion.** The episode runs out of steps.

Tab. 6 lists the shares.

Tab. 6: Root causes of failure as a share of all 800 episodes per condition (Qwen3-VL-8B, Composition).

	No ICL	Video only	Call chain
Failed	91.2	88.8	46.2
Redundancy	67.5	63.1	9.4
Grounding failure	5.0	2.5	5.6
Execution failure	17.5	19.4	23.8
Budget exhaustion	1.2	3.8	7.5

A.4 RoboMME Protocol

RoboMME contains 16 tasks grouped into four suites of four tasks each (Tab. 7). We evaluate every task on all 50 of its test episodes, i.e., 200 episodes per suite and 800 per condition, with one run per episode. These are the same test episodes on which the baselines are reported. Under ICL, the demonstrations shown for an episode come only from other tasks, so no demonstration shares the task or the seed of the evaluated episode.

Tab. 7: RoboMME suites and tasks.

Suite	Tasks
Counting	BinFill, PickXTimes, SwingXTimes, StopCube
Permanence	VideoUnmask, VideoUnmaskSwap, ButtonUnmask, ButtonUnmaskSwap
Reference	PickHighlight, VideoRepick, VideoPlaceButton, VideoPlaceOrder
Imitation	MoveCube, InsertPeg, PatternLock, RouteStick

Baselines. The baseline rows of Tab. 4 ($\pi_{0.5}$, FrameSamp+Modul, MemER, PonderPounce at $1\times$ and $9\times$ training data, and SimpleMemVLA) are taken from prior evaluations: $\pi_{0.5}$, FrameSamp+Modul, and MemER (re-trained following the MemER recipe on a Qwen3-VL-4B backbone) are reported by Dai et al. (2026), while PonderPounce and SimpleMemVLA are from Choi et al. (2026) and Yin et al. (2026). All evaluate on the same 50 test episodes per task (800 in total). We report their published success rates.

A.5 API Cost Accounting

The horizontal axis of Fig. 4c is the API cost of one episode in US dollars. We compute it from the token usage that the provider returns with every response, not from an estimate of prompt length. Every agent turn and every auxiliary VLM grounding call is logged with its prompt and completion token counts and, where the provider reports them, its cache-read and cache-write counts. Because every agent turn re-sends the conversation context, prompt tokens accumulate over the episode. Hidden reasoning tokens are billed as output and are therefore included. The cost excludes local GPU time, the physics simulator, and the segmentation server.

Each call is priced at the provider's first-party, standard-tier list price (Tab. 8), checked on 25 September 2026 (Anthropic, 2026b; OpenAI, 2026c; Google, 2026; Alibaba Cloud, 2026). GPT-6

Astra is priced at its short-context rate, since no call in our runs exceeds that tier. Gemini 3.8 Flash is priced at the standard rate that takes effect on 1 January 2027; its introductory rate, in force through 31 December 2026, is half as high. We served both Qwen models locally with vLLM (Kwon et al., 2023) and price their tokens at Alibaba Model Studio's hosted rates (Alibaba Cloud, 2026), using the non-thinking rates because both models ran with thinking disabled. Qwen3.8-27B has no metered API, so we use the rate of Qwen3.5-27B, on whose architecture it is built (identical 27B configuration), as a proxy.

The provider gateway cached prompt prefixes automatically for Claude Opus 5 and GPT-6 Astra: about 62% of their prompt tokens were cache reads and about 36% cache writes. The figure prices these tokens at the cache rates, which lowers the per-episode cost of these runs by 33–49% relative to pricing without caching. Pricing every call without caching moves the Pareto frontier only slightly: Claude Opus 5 at C2 joins it, while the frontier's endpoints and the best interface of each model stay the same.

The locally served Qwen models carry no cache data, and we price all of their prompt tokens at the standard input rate.

For reference, priced without caching, one C1, C2, or Composition episode costs \$0.007–0.010 for Qwen3.8-27B, \$0.008–0.009 for Qwen3-VL-8B, \$0.19–0.22 for Claude Opus 5, and \$0.25–0.31 for GPT-6 Astra; one C0 episode of Qwen3-VL-8B costs \$0.15.

Tab. 8: List prices used for the API cost (USD per million tokens, standard tier). "–" marks models for which no cache data exist.

Model	Input	Output	Cache read	Cache write	Source
GPT-6 Astra (short context)	10.00	50.00	1.00	12.50	OpenAI (2026c)
Claude Opus 5	5.00	25.00	0.50	6.25[a]	Anthropic (2026b)
Gemini 3.8 Flash[b]	1.50	7.50	–	–	Google (2026)
Qwen3.8-27B[c]	0.30	2.40	–	–	Alibaba Cloud (2026)
Qwen3-VL-8B-Instruct	0.18	0.70	–	–	Alibaba Cloud (2026)

[a]5-minute cache write (1.25× input); cache reads are 0.1× input. [b]Standard rate from 1 January 2027; the introductory rate through 31 December 2026 is \$0.75/\$3.75. [c]Proxy: Qwen3.5-27B (same 27B configuration), non-thinking, context up to 256K tokens.

B Composition Prompt

This section shows the key parts of the Composition system prompt on LIBERO-Pro; the remaining text is omitted. The prompt specifies the primitives, their signatures, and how the form of a target selects the abstraction level. It does not assign subtasks to primitive categories: the model chooses and mixes C0, C1 and C2 calls on its own at every step. The same prompt is used for all four models and for the ICL runs, which differ only in the demonstrations inserted into the first user message. The fixed-interface settings (C0, C1, C2) restrict the contract to the offered category.

Task and levels

```
You control a Franka robot arm in a LIBERO tabletop scene. Make the scene satisfy the task
  instruction, then call finish. Call exactly ONE primitive per turn; you then see its
  result and new images.
PRIMITIVES: move, grasp, place, actuate, verify, plus finish.
LEVELS follow from how you write the target -- mix them freely, per step:
  C2  a name ("red mug")                          -> the system finds it and decides the motion.
  C1  a pixel {"cam", "u", "v"} or {"xyz": [...]} -> you say where, the system decides how.
  C0  move with "delta" only                      -> one short direct action chunk; you decide
      everything.
```

Primitive signatures

```
move     C1 move(to={"xyz": [x, y, z]}, yaw, gripper)
         C0 move(delta=[dx, dy, dz, droll, dpitch, dyaw], gripper, steps)
grasp    C2 grasp(object="<name>")
```

```
          C1 grasp(object={pixel}, yaw, offset, preheight, lift)
place     C2 place(object="<name>", target="<name>", mode)
          C1 place(target={pixel}, yaw, offset, preheight, retreat)
actuate   C2 actuate(mode, target="<name>")
          C1 actuate(mode, target={pixel}, by=[dx, dy, dz])
verify    verify(relation, object, target)
finish    finish()
```

Output format

```
Reply with one JSON object and nothing else:
{"fn": "<primitive>", "args": {...}, "why": "<one short sentence>"}
```

C Concrete C1 and C2 Manipulation Interfaces

C.1 Worked Examples on LIBERO-Pro

This section instantiates the CoP representation with concrete calls from the C1 geometric and C2 semantic interfaces on LIBERO-Pro, for the task "pick up the butter and place it in the basket". Calls are written in the unified JSON format of App. B. Pixels are given in the 0–1000 ruler units of the 1024×1024 agent-view image (native pixel $= 1.024\times$ ruler value). The executor routines below are simplified from the released code (`benchmarks/libero_pro/gpt6/pixel_tools.py` and `unified_prim/platforms/libero.py`); logging and error handling are omitted. In both interfaces the model receives neither simulator ground-truth object poses nor success predicates.

At C1, the model specifies an image anchor, Cartesian offsets, a wrist yaw and motion parameters. The executor back-projects the selected pixel using measured depth and runs a fixed closed-loop motion routine. At C2, the model names an object or destination, and the executor grounds the name with SAM 3 masks and a VLM selection, then runs a scripted grasp or placement. The two examples therefore express the same grasp–place structure through different agent-facing primitive contracts.

C.1.1 C1: Geometric Primitives

At C1, the model is responsible for grounding the relevant image location and selecting continuous motion parameters. The resulting CoP entries expose both the selected primitive and the geometry supplied by the model.

C1 grasp call and executor source. The following call grasps the object at ruler position $(245, 308)$:

```
{"fn": "grasp",
 "args": {"object": {"cam": "agentview", "u": 245, "v": 308},  # 0..1000
          "offset_x": 0.00, "offset_y": 0.00, "offset_z": 0.00,  # m
          "yaw": 0.00, "preheight": 0.10, "lift": 0.15,          # rad, m, m
          "close_steps": 20},                                    # env steps
 "why": "Grasp the butter at its top centre."}
```

The primitive executes the following procedure:

```
def execute_c1_grasp(world, c, anchor_obs):
    # Back-project the selected pixel to a 3D point using depth, then apply
    # the model-provided Cartesian correction.
    anchor = pixel_anchor(anchor_obs, 1.024 * c["u"], 1.024 * c["v"])
    grasp_pt = anchor + np.array([c["offset_x"], c["offset_y"], c["offset_z"]])
    pregrasp_pt = grasp_pt + [0.0, 0.0, c["preheight"]]    # approach pose
    lift_pt = grasp_pt + [0.0, 0.0, c["lift"]]             # retention check
    yaw = c["yaw"]

    # Collision-free approach with the gripper open, then descend to contact.
    execute_trajectory(world, plan_motion(world.snapshot(), pregrasp_pt),
                       gripper=OPEN, yaw=yaw, label="llm.grasp.approach")
```

```
    move_to(world, grasp_pt, gripper=OPEN, yaw=yaw, tol=0.008, max_steps=60,
            stall_tol=0.015, label="llm.grasp.descend")

    # Close for a bounded number of steps, then lift with the gripper closed.
    set_gripper(world, CLOSE, int(c["close_steps"]))
    move_to(world, lift_pt, gripper=CLOSE, yaw=yaw, max_steps=40,
            label="llm.grasp.lift")

    g = check_grasp(world.observe())  # holding? from the observed gripper state
    return {"holding": g.holding, "opening": round(g.opening, 4)}
```

The model therefore controls both perceptual grounding (`u`, `v`) and grasp geometry (`offset_*`, `yaw`, `preheight`, and `lift`). The executor fixes only the structure of the motion sequence.

C1 place call and executor source. The following call releases the held object above the support point at ruler position $(350, 250)$:

```
{"fn": "place",
 "args": {"target": {"cam": "agentview", "u": 350, "v": 250}, "mode": "into",
          "offset_x": 0.00, "offset_y": 0.00, "offset_z": 0.08,  # m
          "yaw": 0.00, "preheight": 0.10, "retreat": 0.10,       # rad, m, m
          "open_steps": 12},                                     # env steps
  "why": "Release the butter above the centre of the basket."}
```

Its execution logic mirrors grasping but reverses the contact sequence:

```
def execute_c1_place(world, c, anchor_obs):
    # Back-project the support pixel and shift it to the release point.
    anchor = pixel_anchor(anchor_obs, 1.024 * c["u"], 1.024 * c["v"])
    release_pt = anchor + np.array([c["offset_x"], c["offset_y"], c["offset_z"]])
    preplace_pt = release_pt + [0.0, 0.0, c["preheight"]]  # from above
    retreat_pt = release_pt + [0.0, 0.0, c["retreat"]]     # lift clear
    yaw = c["yaw"]

    # Carry the held object above the destination, then descend to release.
    execute_trajectory(world, plan_motion(world.snapshot(), preplace_pt),
                       gripper=CLOSE, yaw=yaw, label="llm.place.carry")
    move_to(world, release_pt, gripper=CLOSE, yaw=yaw, tol=0.012, max_steps=50,
            label="llm.place.descend")

    # Open for a bounded number of steps, then retreat vertically.
    set_gripper(world, OPEN, int(c["open_steps"]))
    move_to(world, retreat_pt, gripper=OPEN, yaw=yaw, max_steps=30,
            label="llm.place.retreat")

    # Acknowledges the executed command, not semantic success.
    return {"release_command_completed": True,
            "opening": round(world.observe().gripper_opening, 4)}
```

Omitted C1 arguments take the defaults shown above. Numeric arguments are clipped to fixed ranges before execution: pixels to $[0, 1000]$, offsets to $[-0.20, 0.20]$ m, pre-height to $[0.04, 0.20]$ m, lift to $[0.04, 0.35]$ m, retreat to $[0, 0.20]$ m, yaw to $[-\pi, \pi]$, and gripper durations to $[5, 40]$ environment steps. A pixel without valid depth returns an error without any motion. This prevents malformed outputs from becoming unconstrained robot commands.

C.1.2 C2: Semantic Primitives

C2 removes numeric geometry from the model interface. The model names the intended object or destination, and the executor resolves the name into geometry and runs a scripted motion. Thus, C2 is a semantic action interface rather than a low-level physical primitive.

C2 grasp call and executor source. For the same task, the model produces:

```
{"fn": "grasp", "args": {"object": "butter"},
 "why": "Pick up the butter first."}
```

The executor grounds the phrase and runs the scripted grasp:

```
def execute_c2_grasp(world, c, client, vlm, state):
    # Parse the model's phrase into a noun and an optional instance relation
    # ("black bowl next to the ramekin").
    ref = parse_object_ref(strip_article(c["object"]))

    # Ground it: SAM 3 proposes candidate masks and the VLM selects one;
    # the pose is estimated from the selected mask and depth.
    pose, how = locate(world, world.snapshot(), ref, client, vlm)
    if pose is None:
        return {"ok": False, "status": "ground_fail", "grounding": how}

    # Pre-grasp from the perceived pose; then approach, grasp, lift.
    plan = compute_pregrasp(pose, ref.noun, 0)
    execute_trajectory(world, plan_motion(world.snapshot(), plan.pre),
                       gripper=OPEN, yaw=plan.yaw, label="prim.grasp.approach")
    gr = grasp_scripted(world, plan, pose)

    # Remember what is held for a later place(target=...) without object.
    state["held"] = {"phrase": c["object"], "noun": ref.noun,
                     "hang": gr.hang, "yaw": plan.yaw} if gr.ok else None
    return {"ok": gr.ok, "status": "holding" if gr.ok else gr.state.label,
            "grounding": how, "object_xyz": pose.center.round(3).tolist()}
```

The executor never compares the phrase with the task instruction: a wrong phrase either grasps the wrong object or fails to ground. C2 therefore tests whether the model can name the correct object, not whether it can infer an accurate pixel coordinate, gripper yaw, or grasp height.

C2 place call and executor source. For the same task, the placement decision is:

```
{"fn": "place", "args": {"target": "basket", "mode": "into"},
 "why": "Put the held butter into the basket."}
```

The executor grounds the destination region and runs the scripted placement:

```
def execute_c2_place(world, c, client, vlm, state):
    held = state["held"]  # set by the preceding C2 grasp
    spec = parse_task(sentence(held["phrase"], c["mode"], c["target"]))

    # Ground the destination region: SAM 3, with a VLM fallback.
    region, how = locate_region(world, world.snapshot(), spec, client, vlm)
    if region is None:
        return {"ok": False, "status": "ground_fail", "target_how": how}

    # Release pose from the region and the held object's hang below the gripper.
    pp = compute_preplace(region, held["hang"],
                          target_noun=spec.target.noun,
                          object_noun=spec.obj.noun, grasp_yaw=held["yaw"])
    execute_trajectory(world, plan_motion(world.snapshot(), pp.pre),
                       gripper=CLOSE, yaw=pp.yaw, label="prim.place.carry")
    place_scripted(world, pp)  # descend, open, retreat
    state["held"] = None

    # No goal check here; the agent can call verify(...) afterwards.
    return {"ok": True, "status": "released", "target_how": how,
            "region": region.center.round(3).tolist()}
```

The whole-skill form `place(object=..., target=...)` chains the same grounding, grasp, carry and release steps in one call and ends with a sensor-only goal check; its C1 variant accepts clicked object, target and grasp pixels in place of the automatic estimates (App. B).

Interface interpretation. Tab. 9 summarizes the allocation of responsibility across the two interfaces. C1 assigns the model a substantially larger continuous decision space, whereas C2 isolates semantic reference resolution. The comparison measures how manipulation success changes when geometric grounding and control are delegated from the model to a fixed perception-and-motion pipeline.

Tab. 9: Concrete differences between the C1 and C2 example interfaces.

Property	C1	C2
Model output	Pixel anchor and numeric motion parameters	Object or target phrase, plus a placement mode
Model must solve	Visual grounding and geometric parameterization	Semantic reference resolution
Grounding	Model's pixel, back-projected with depth	SAM 3 candidate masks, VLM selection
Geometric planning	Partly determined by the model	Scripted
Example grasp	`grasp(object={u:245, v:308}, ...)`	`grasp(object="butter")`
Example place	`place(target={u:350, v:250}, ...)`	`place(target="basket", mode="into")`