SCAFFOLDING GENERAL VISION LANGUAGE MODELS FOR ZERO-SHOT ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models have advanced robotic manipulation, but their zero-shot generalization remains limited, and their reliance on specialized robotics data keeps them from benefiting directly from rapidly advancing general-purpose vision-language models (VLMs). Agentic robotic systems instead use VLMs for high-level reasoning, yet typically depend on substantial external modules. This motivates our question: *Can a general-purpose VLM itself serve as the decision-making core for robotic manipulation?* Evaluating VLMs on action selection, progress assessment, and task completion reveals a *capability-control gap*: VLMs reason well about manipulation, but their decisions do not match the abstraction of continuous, embodiment-specific control. We introduce ***MotorMind***, a VLM-centric manipulation harness with an asynchronous inference loop. Without task-specific policy training, ***MotorMind*** achieves 66.7% base success and 53.8% success under perturbations on LIBERO-PRO, compared with at most 13.3% and 19.2% for the prior zero-shot methods we evaluate, and the same interface reaches 95% average success on a real xArm6 robot across direct manipulation and human-perturbation settings. A stronger VLM backbone further raises base success without changing the harness, and the remaining errors, mainly in visual grounding, embodied reasoning, and action knowledge, decrease with stronger VLMs. These results reframe zero-shot manipulation as aligning general-purpose model capabilities with the control representation, rather than solely learning specialized action policies.

1 INTRODUCTION

Robotics foundation models such as vision-language-action (VLA) models have emerged as a promising paradigm for robotic manipulation by mapping visual observations and language instructions directly to robot actions (Kim et al., 2025b; Black et al., 2025a). Despite strong in-distribution performance, their zero-shot transfer remains limited under changes in object appearance, spatial configuration, task semantics, and environment structure. In practice, strong performance in an unknown setting often relies on additional robot-action data or policy adaptation (Kim et al., 2025a). In addition, these models were created via training on specialized robotics datasets and cannot directly benefit from the emerging intelligence from recent generic vision-language foundation models.

A parallel line of work explores *agentic robotics*, where general-purpose VLMs are used for planning and high-level reasoning while external components handle perception, grounding, action generation, motion planning, or execution(Lu et al., 2026; Zhang et al., 2026). This avoids learning every stage of manipulation end-to-end, but often rely on substantial specialized infrastructure around the VLM, including learned action experts(Black et al., 2025a), segmentation model(Carion et al., 2026), skill libraries(Lu et al., 2026; Fu et al., 2026), motion planners(Huang et al., 2023), and iterative execution modules(Chen et al., 2026c). As a result, advances in the spatial understanding and embodied reasoning capabilities of general-purpose VLMs have not necessarily been translated into simpler and more general robotic systems. This motivates a more direct question: *Can a general-purpose VLM itself serve as the decision-making core for robotic manipulation?*

To study this question, we first systematically evaluate VLMs on the decision-making capabilities required during manipulation: selecting the next action, judging task progress, and determining task completion. Our analysis reveals a *capability-control gap*. Modern VLMs already exhibit substantial semantic reasoning capability, but their decisions are expressed at an abstraction that

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.

Method	Action Generation	Extra Components	Execution
$\pi_{0.5}$ (Black et al., 2025a)	Learned action policy	Trained action model	Action chunks
MolmoAct2 (Fang et al., 2026)	Learned action policy	Trained action model	Closed-loop prediction
OpenVLA-OFT (Kim et al., 2025b)	Trained action model	Fine-tuned action decoder	Action chunks
GR00T N1.5 (NVIDIA, 2025)	Trained action model	Diffusion action model	Policy rollout
CaP-X (Fu et al., 2026)	Generated robot code	Perception and control tools	Program execution
VoLoAgent (Chen et al., 2026a)	VLA + robot primitives	Perception models and tools	Tool orchestration
Harness VLA (Zhang et al., 2026)	VLA + primitives	VLA + primitive library	Feedback-guided retries
MotorMind	**VLM Only**	**None**	**Asynchronous Execution**

Table 1: **Comparison of robotic manipulation methods.** Most existing works rely on a learned action policy, generated robot code, or extra perception and control modules. ***MotorMind*** uses the VLM directly for semantic action generation without a coding agent, learned action model, SAM-based perception, or external inverse kinematics (IK) solver. See Appendix for more related works.

does not directly match continuous, embodiment-specific robot control. The central challenge is therefore not only whether a VLM can reason about manipulation, but how to expose physical control in a form that preserves this reasoning ability while remaining executable on a robot.

Motivated by this observation, we introduce ***MotorMind***, a VLM-centric manipulation harness designed around two principles. First, we use a compact mid-level action representation consisting of translation, rotation, gripper operation, and stopping decisions for the VLM, while embodiment-specific controllers translate the decisions into physical motion. Second, we introduce an asynchronous inference loop in which planning, execution, and progress assessment operate concurrently. Rather than generating a long action sequence in advance, the system continually incorporates execution outcomes and updates future decisions as interaction unfolds.

This design enables the same VLM-facing reasoning and action interface to operate across different robot embodiments. In simulation, ***MotorMind*** achieves a 66.7% success rate on the LIBERO-PRO base suite, compared with 13.3% for the strongest prior method we evaluated without target-domain policy training(Fu et al., 2026). Under semantic, object, position, and task perturbations, it achieves 53.8% success, while the strongest prior zero-shot baseline reaches 19.2%. Importantly, the same interface transfers to a *real xArm6 robot without additional policy adaptation*, achieving 95% average success across direct manipulation and human disturbance settings. These results show that a general-purpose VLM can provide a strong zero-shot decision-making core when coupled with an appropriate control abstraction and execution structure.

MotorMind also seamlessly improves with the VLM backbone: replacing the Qwen3.8-Flash-Next backbone (targeted for fast local robotics inference) with GPT-6 Sol (stronger but slower via API) further boosts the success rate on LIBERO-PRO base suite from 66.7% to 83.3%. This suggests that *improvements in general-purpose multimodal foundation models may translate directly into stronger robotic behavior* without requiring a redesign of the overall control architecture, or retraining with specialized data or model architecture. Our extensive failure analysis sheds light on the error modes that stronger VLM models reduce, with most failures arising from three sources: imperfect visual grounding, such as selecting the wrong object or target location; embodied reasoning errors, such as incorrectly determining that a task has completed; and insufficient action knowledge, where the model repeatedly proposes ineffective motions for particular situations.

Overall, our results support a complementary direction to conventional VLA scaling. Rather than learning robotic behavior fully from demonstrations, increasingly capable VLMs can provide much of the semantic decision making required for manipulation when physical interaction is exposed at the right level of abstraction. This reframes zero-shot robotic control as a problem of *aligning general-purpose model capabilities with control representation*, rather than solely learning specialized action policies.

2 Diagnosing VLMs for Robotic Manipulation

Using a general-purpose VLM as a robotic policy requires bridging semantic reasoning and embodiment-specific motion. We first examine how semantic decisions can be grounded into robot control and which VLM capabilities are sufficiently reliable for manipulation.

Model	Action	Progress	Completion	Overall	Latency ↓	Cost-Eff. ↑
Qwen3.8-Flash-Next-FP8(Qwen Team, 2026)	36.25	55.00	65.00	52.08	**276**	**188.70**
HY-Embodied-0.5 MoT-2B(Tencent Robotics X et al., 2026)	20.00	50.00	47.50	39.17	402	97.44
Hy-Embodied-VLM-1.0 A3B(Wang et al., 2026)	18.75	43.75	50.00	37.50	592	63.34
Cosmos3-Nano(NVIDIA, 2026)	18.75	50.00	62.50	43.75	3035	14.42
GLM-5.3-Flash(Z.ai, 2026)	37.50	52.50	58.75	49.58	403	123.03
GPT-6 Astra(OpenAI, 2026)	**60.00**	**76.25**	**82.50**	**72.92**	8724	8.36

Table 2: **Diagnostic Results.** Accuracy (%) on action selection, progress verification, goal completion, and all 240 questions overall. Latency denotes mean inference time per query in milliseconds.

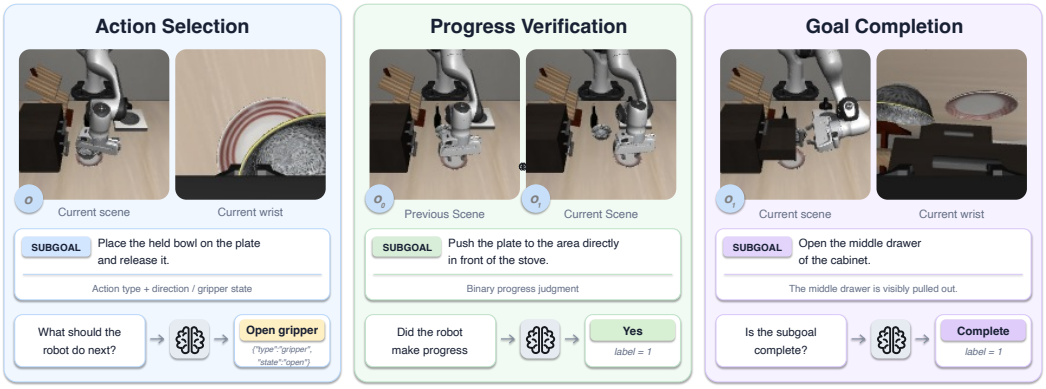

Figure 1: **Diagnosing VLM capabilities for robotic manipulation.** We evaluate action selection, progress assessment, and goal completion.

A VLM policy must repeatedly answer three questions: *What should happen next? Did the previous action make progress? Is the task complete?* We therefore study three capabilities:

- **Action Selection:** Choose the next action from the instruction and current observations.
- **Progress Assessment:** Determine whether an observed transition advances the task.
- **Goal Completion:** Determine whether the current state satisfies the instruction.

Action selection proposes future behavior, progress assessment interprets execution outcomes, and goal completion determines whether further actions are needed.

Diagnostic Benchmark. We construct an embodied question-answering benchmark with 240 questions, 80 for each capability. Action selection and progress assessment requires the model to use past and current observations to predict the type of the next action and determine whether progress toward the current subgoal has been achieved. Goal completion requires the model to determine whether the task goal has been completed from the current observation. The diagnose is designed to assess the decision-making ability in visual robot manipulation. See Appendix B for more details.

Diagnostic Results and Analysis. The benchmark results from Table 2 reveal four observation: **First**, general-purpose VLMs already exhibit non-trivial manipulation decision capability, achieving comparable performance across action selection and goal completion. This suggests that pretrained multimodal models already possess substantial manipulation reasoning capabilities. **Second**, translating this reasoning into precise robot actions remains challenging. VLM predictions are often semantically appropriate at the object and task level, yet errors remain when decisions must be grounded into specific spatial actions such as motion type and direction. We refer to this mismatch as a capability–control gap. Rather than committing to a single decision, this observation motivates repeatedly refining the VLM's actions based on updated observations. **Third**, VLMs are also imperfect at assessing execution outcomes. Progress assessment remains difficult, and even goal-completion judgments are not consistently correct, indicating that a single VLM prediction should not be treated as a reliable execution verifier. **Last**, stronger VLMs present an important opportunity. GPT-6 Astra performs substantially better across all three diagnostic capabilities, demonstrating stronger spatial understanding and manipulation reasoning. However, these gains come with much higher inference latency, creating a trade-off between reasoning quality and interaction frequency.

3 *MotorMind*: VLM HARNESS FOR ZERO-SHOT ROBOTIC MANIPULATION

The analysis in Section 2 motivates three design choices for ***MotorMind***. First, we use Qwen3.8-Flash-Next as a practical balance between manipulation reasoning capability and inference latency. Second, the system continuously refines short-horizon action decisions, allowing spatial errors to be corrected as interaction unfolds. Third, as VLMs are not fully reliable at assessing execution progress or task completion, MotorMind repeatedly verifies both progress and completion during execution. These reasoning and verification steps are performed asynchronously with robot motion to maintain frequent closed-loop interaction without requiring each VLM call to block execution.

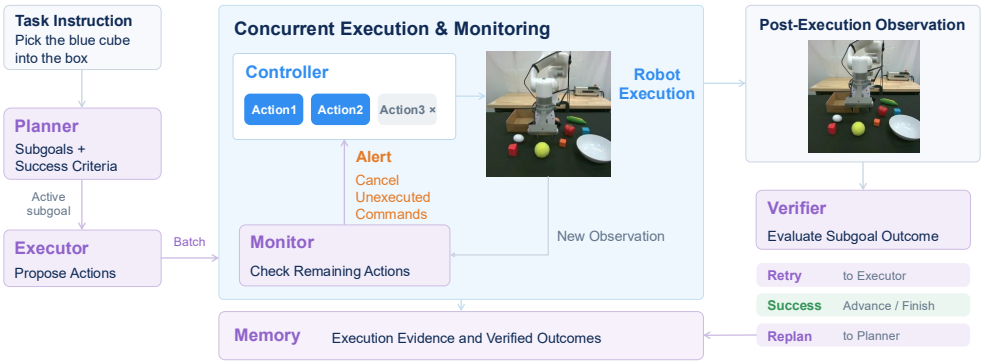

Figure 2: **Overview of *MotorMind*.** A Planner decomposes the task into subgoals, and an Executor converts each active subgoal into short batches of mid-level actions. A deterministic Controller grounds these actions into robot-specific motion. During execution, an asynchronous Monitor continuously examines new observations and can interrupt an outdated action batch. A Verifier evaluates subgoal outcomes, while Memory carries execution evidence into subsequent decisions.

3.1 ARCHITECTURE OVERVIEW

MotorMind decomposes decision making into five reasoning roles, *all instantiated by the same general-purpose VLM*, together with a deterministic Controller for physical execution. The roles differ in their responsibilities and input context instead of requiring separately trained models.

- **Planner.** The Planner translates the task instruction into a sequence of subgoals, each describing an intended state change and a corresponding success criterion. It maintains the remaining task plan and revises it when execution evidence invalidates an earlier assumption.
- **Executor.** The Executor determines how to achieve the active subgoal from the current visual observations, robot state, and recent execution context. Instead of predicting low-level control commands, it produces a short batch of mid-level actions that can be grounded by the Controller.
- **Monitor.** The Monitor runs concurrently with physical execution and inspects newly arriving observations for events that may invalidate the current action batch, such as grasp failure, unexpected object motion, interaction with the wrong target, or early completion of the current subgoal. When necessary, it requests interruption at a safe action boundary.
- **Verifier.** After a subgoal attempt, the Verifier evaluates whether its success criterion has been satisfied using visual observations and robot-state evidence. It also determines whether the overall task has completed.
- **Memory.** Memory condenses recent observations, executed actions, physical measurements, Monitor alerts, and verification results into compact execution context for subsequent Planner and Executor calls.
- **Controller.** The Controller validates mid-level actions, converts them into embodiment-specific Cartesian and gripper commands, executes them, and records the resulting robot state. It contains no learned component.

3.2 MID-LEVEL ACTION REPRESENTATION

For each active subgoal, the Executor produces a short action batch

$$\mathcal{A} = (a_1, \ldots, a_K), \qquad a_k = (\tau_k, \boldsymbol{\eta}_k),$$

where $\tau_k \in \{\texttt{move}, \texttt{rotate}, \texttt{gripper}\}$. Here, $\boldsymbol{\eta}_k$ contains the continuous parameters associated with the selected action. A `move` specifies a translational direction and displacement, a `rotate` specifies an end-effector rotation, and a `gripper` command opens or closes the gripper.

3.3 Feedback-Constrained Execution

The mid-level action still needs to be grounded in the current physical scene. For each active subgoal, the Executor receives the latest images, robot state, Memory, and the associated success criterion. It reasons directly over these inputs to identify the relevant objects, spatial relationships, and required motion, without relying on a separate learned detector or segmentation model.

A plausible VLM prediction is not necessarily a valid robot command. The requested motion may exceed workspace limits, approach an unsafe configuration, or become inappropriate because the physical scene has changed since the action was proposed. The Controller therefore checks each action before execution, including workspace constraints, displacement magnitude, target distance, clearance, and gripper state. Invalid commands can be rejected or clipped, and the resulting feedback is returned to the reasoning modules. Execution itself provides an additional source of information. A grasp may slip, an object may move differently than expected, or the achieved end-effector pose may differ from the requested one. ***MotorMind*** records both commanded and observed outcomes. These updated observations and robot states condition subsequent action generation and provide the evidence required for retrying a subgoal or revising the remaining plan.

3.4 Asynchronous Scheduling

A conventional VLM agent often follows a fully sequential loop: generate an action, execute it, wait for completion, observe the result, and invoke the model again. In robotics, this coupling makes model latency part of every control step and delays reasoning about execution changes until the current action has already finished.

Once the Executor produces a short batch of mid-level actions, the Controller can execute them without waiting for another VLM call. At the same time, the Monitor receives new observations and evaluates whether the assumptions underlying the current batch remain valid. Execution continues whenever behavior is proceeding as expected, while reasoning remains active in parallel. This concurrency is important because an action sequence that was valid when generated may become obsolete during execution. A grasp can fail before a subsequent placement action, an object can move unexpectedly, or the current subgoal may be completed earlier than expected. When the Monitor detects such an event, it requests interruption at the next safe action boundary, preventing the Controller from blindly completing the remaining batch.

Asynchronous scheduling also applies at the level of task progress. While the Controller executes the current batch, higher-level reasoning can assess whether the active subgoal and remaining plan remain consistent with new observations. If execution evidence invalidates the current plan, the Planner can revise future subgoals based on the updated state. After a subgoal attempt terminates, either normally or through an interruption, the Verifier evaluates its success criterion and Memory summarizes the resulting evidence. The system can then advance to the next subgoal, request another short-horizon action batch, retry the current subgoal, or adopt a revised plan.

4 Experiments

4.1 Main Experiment

Evaluation Settings. We evaluate ***MotorMind*** on the Goal, Spatial, and Object suites of LIBERO-PRO(Zhou et al., 2025). The *base evaluation* uses the original, unperturbed episodes. The *perturbation evaluation* measures performance under Semantic, Object, Position, and Task perturbations. Together, these settings test task completion in the original environments and generalization when instructions, objects, layouts, or tasks change. See Appendix D for more details.

Baselines and Implementation. We compare with four VLA policies ($\pi_{0.5}$, MolmoAct2, OpenVLA/OFT, and GR00T N1.5) in zero-shot and fine-tuned configurations where available. We also evaluate CaP-X and Harness VLA with their single-pass and iterative configurations, and VoLoAgent with either a fine-tuned or a zero-shot VLA backbone. We group methods according to whether

Method	Configuration	Base success rate (%)				Perturbation success rate (%)					Base efficiency		Perturbation efficiency	
		Goal	Spatial	Object	Avg.	Semantic	Object	Position	Task	Avg.	Time (s) ↓	Cost-Eff. ↑	Time (s) ↓	Cost-Eff. ↑
Fine-Tuned Policies / Agentic Methods with Fine-Tuned Policies														
$\pi_{0.5}$	Fine-tuned	95.0	**100.0**	**100.0**	98.3	**98.3**	**95.0**	36.7	23.3	63.3	5.9	997.1	6.8	**562.1**
MolmoAct2	Fine-tuned	**100.0**	**100.0**	**100.0**	**100.0**	95.0	90.0	36.7	30.0	62.9	6.6	914.6	7.7	492.2
OpenVLA / OFT	Fine-tuned	95.0	**100.0**	**100.0**	98.3	96.7	86.7	11.7	10.0	51.2	**5.7**	**1033.5**	**6.7**	459.9
GR00T N1.5	Fine-tuned	0.0	95.0	0.0	31.7	30.0	30.0	1.7	16.7	19.6	15.6	121.8	16.2	72.4
VoLoAgent (FT VLA as Tool)	Zero-shot agent	55.0	90.0	**100.0**	81.7	88.3	83.3	45.0	25.0	60.4	106.2	46.1	216.9	16.7
	Zero-shot	35.0	30.0	30.0	31.7	30.0	31.7	28.3	25.0	28.8	354.6	5.4	342.5	5.0
	S1: 5 loops	55.0	50.0	55.0	53.3	65.0	51.7	50.0	50.0	54.2	1040.4	3.1	1286.2	2.5
Harness VLA (FT VLA as Tool)	S1: 10 loops	70.0	90.0	80.0	80.0	75.0	66.7	**53.3**	**66.7**	**65.4**	1764.0	2.7	2184.1	1.8
	S2: 5 loops	-	-	-	-	55.0	48.3	43.3	41.7	47.1	-	-	1125.2	2.5
	S2: 10 loops	-	-	-	-	70.0	58.3	41.7	45.0	53.8	-	-	1866.2	1.7
Zero-Shot Methods														
$\pi_{0.5}$	Zero-shot	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.3	0.0	8.2	0.0
MolmoAct2	Zero-shot	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.4	0.0	9.3	0.0
OpenVLA / OFT	Zero-shot	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.3	0.8	34.7	0.0	34.4	1.5
GR00T N1.5	Zero-shot	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**6.3**	0.0	**6.2**	0.0
	Zero-shot	5.0	0.0	5.0	3.3	15.0	3.3	1.7	3.3	5.8	68.5	2.9	72.4	4.8
CaP-X (GPT5.6-Terra + Claude Opus 5)	5 loops	25.0	0.0	15.0	13.3	25.0	16.7	16.7	15.0	18.3	290.1	2.8	270.2	4.1
	10 loops	25.0	0.0	15.0	13.3	26.7	16.7	16.7	16.7	19.2	346.4	2.3	320.4	3.6
VoLoAgent (zero-shot VLA as Tool)	Zero-shot	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	500.0	0.0	500.0	0.0
MotorMind (Qwen3.8-Flash-Next)	Zero-shot	**45.0**	**75.0**	**80.0**	**66.7**	**58.3**	**46.7**	**51.7**	**58.6**	**53.8**	223.4	**17.91**	248.5	**12.99**

Table 3: **Main Evaluation Results.** Success rates and averages are reported separately for unperturbed base tasks and perturbed tasks; walltime and cost-efficiency use the corresponding evaluation subset. Methods are grouped by whether their underlying policy uses task-specific fine-tuning.

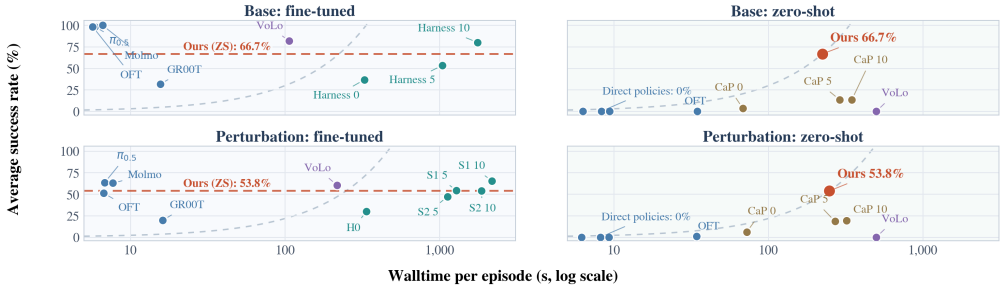

Figure 3: **Success rate versus episode wall time.** The four panels separate base and perturbation evaluations and distinguish methods with fine-tuned policies from zero-shot methods. Orange points denote ***MotorMind***; orange horizontal lines show its success rate for reference in the fine-tuned panels. Gray dashed curves indicate the same cost-efficiency as ***MotorMind***.

their underlying action policy uses task-specific fine-tuning; agent-level test-time interaction alone does not place a method in the fine-tuned group.***MotorMind*** uses Qwen3.8-Flash-Next as its VLM backbone. It uses no LIBERO demonstrations, task-specific policy fine-tuning, or learned task-specific action model. Its VLM-facing semantic actions and asynchronous execution follow Sec. 3.

Metrics. We report task success rate (SR, %) for each suite or perturbation and the corresponding average. We also report average episode wall time in seconds. To relate success to execution time, we define $\mathrm{CostEff.} = \frac{60\,(\mathrm{SR}/100)(\%)}{\mathrm{Walltime\ (s)}}$. Higher cost-efficiency indicates more success per unit of time under this definition. Each success rate and wall time is computed on the corresponding base or perturbation evaluation subset.

Experiment Results. ***MotorMind*** achieves the highest success rate among the evaluated zero-shot methods, with average success rates of 66.7% on the base suites and 53.8% under perturbations (Table 3). Without task-specific action-policy training, it achieves 75.0% success on base Spatial tasks and 58.6% under Task perturbations. As shown in Figure 3, this zero-shot performance is also competitive with task-fine-tuned policies: across 240 perturbed episodes, ***MotorMind*** achieves 53.8% average success, compared with 51.2% for OpenVLA-OFT. The modest 2.6-percentage-point gap indicates comparable performance, highlighting ***MotorMind***'s ability to match a task-fine-tuned policy without task-specific policy training. These results demonstrate that ***MotorMind*** can translate the capabilities of a pretrained multimodal model into effective manipulation through observation-driven execution. Its strong zero-shot performance across the evaluated suites and perturbations supports this approach as a practical alternative to task-specific policy fine-tuning.

Method	Dynamic Reasoning *(10 tasks)*	Scene Shift *(10 tasks)*	Dynamic Manipulation *(5 tasks)*	Prompt Shift *(5 tasks)*
$\pi_{0.5}$	0%	<u>70%</u>	0%	<u>20%</u>
GR00T N1.5	0%	10%	0%	0%
MolmoAct2	20%	50%	<u>40%</u>	0%
CaP-X	<u>30%</u>	20%	20%	**60%**
MotorMind	**70%**	**90%**	**80%**	**60%**

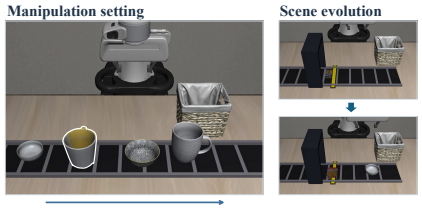

Table 4: **Adaptive reasoning evaluation. Left:** Success rates across four task groups. **Right:** Example in which the robot must reason and act while the scene evolves. Parentheses indicate the number of tasks; bold and underlined entries denote the best and second-best results, respectively.

4.2 Adaptive Tasks Experiment

We further test whether ***MotorMind*** can adapt its decisions to changes that occur during execution. In the conveyor setting shown on the right of Table 4, objects continue moving while the robot identifies and approaches its target, requiring the agent to reason from the latest observations rather than a static initial scene. We evaluate four groups of tasks: **Dynamic Reasoning, Dynamic Manipulation, Scene Shift, and Prompt Shift**.

- **Dynamic Manipulation:** Track and manipulate objects moving on a conveyor during execution.
- **Dynamic Reasoning:** Track and manipulate moving objects whose target is specified through semantic, relational, or temporal descriptions rather than named directly.
- **Scene Shift:** Keep the instruction fixed while moving target object or destination during execution.
- **Prompt Shift:** Change the instruction after a physical trigger without resetting the robot or scene, requiring the agent to revise its decisions from the current state.

As shown in Table 4, ***MotorMind*** achieves 70%, 90%, 80%, and 60% success across the four groups, respectively. The strongest baseline achieves 30%, 70%, 40%, and 60%, respectively. These results suggest that the same semantic action interface and asynchronous inference process can support decisions even when the environment changes during execution.

5 Real Robot Deployment

We deploy ***MotorMind*** on a physical xArm6 robot to evaluate its real-world performance with tabletop tasks. ***MotorMind*** receives observations from the robot and RealSense D455 cameras and executes actions through the standard xArm6 API, without task-specific demonstrations or fine-tuning. We design tasks in three settings to evaluate ***MotorMind***'s ability to perceive objects, adapt to scene changes, and interpret instructions, respectively.

Direct Perception. The instruction explicitly identifies both the object and its destination, i.e., *"Put the blue cube in the white bowl."* This setting tests whether the system can locate the specified object and destination and execute the actions required to complete the placement on the physical robot.

Human Perturbation. We use the same object-destination tasks as in Direct Perception, but introduce human interventions after execution begins. A person deliberately moves or replaces the relevant object during execution to disrupt task progress. This setting tests whether the system can use subsequent observations to detect the scene change and adapt its remaining actions.

Semantic Understanding. The instruction identifies the intended object or destination through semantic categories (e.g., *"the food item"*), visual attributes (e.g., *"the same color"*), or spatial relations (e.g., *"in the middle"*), rather than explicitly naming it. This setting tests whether the system can interpret the instruction in the observed scene and execute the corresponding action.

Results and Case Study In our evaluation, each object-destination pair in Direct Perception and Human Perturbations is evaluated over 10 trials per setting, whereas each of the four Semantic Understanding tasks is evaluated over 5 trials. Object arrangements are randomized.

Results. As shown in Table 5, ***MotorMind*** achieves a pooled success rate of 95% (114/120) across Direct Perception (98%: 59/60) and Human Perturbations (92%: 55/60), showing high placement success even under human interventions. On the four Semantic Understanding tasks, success rates

Object	Direct Perception		Human Intervention		Avg. SR
	Bowl	Box	Bowl	Box	
Blue Cube	100%	100%	100%	100%	100%
Corn	100%	100%	100%	80%	95%
Battery	90%	100%	80%	90%	90%
Avg. SR	**97%**	**100%**	**93%**	**90%**	**95%**

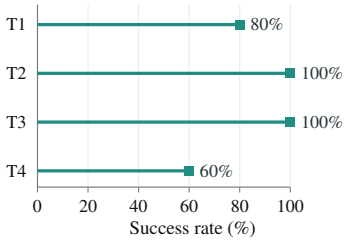

Table 5: **Real-robot zero-shot deployment. Left:** Success rates in Direct Perception and Human Perturbation tasks. **Right:** Success rates on Semantic Understanding tasks. Instructions include semantic categories, visual attributes, or spatial relations.

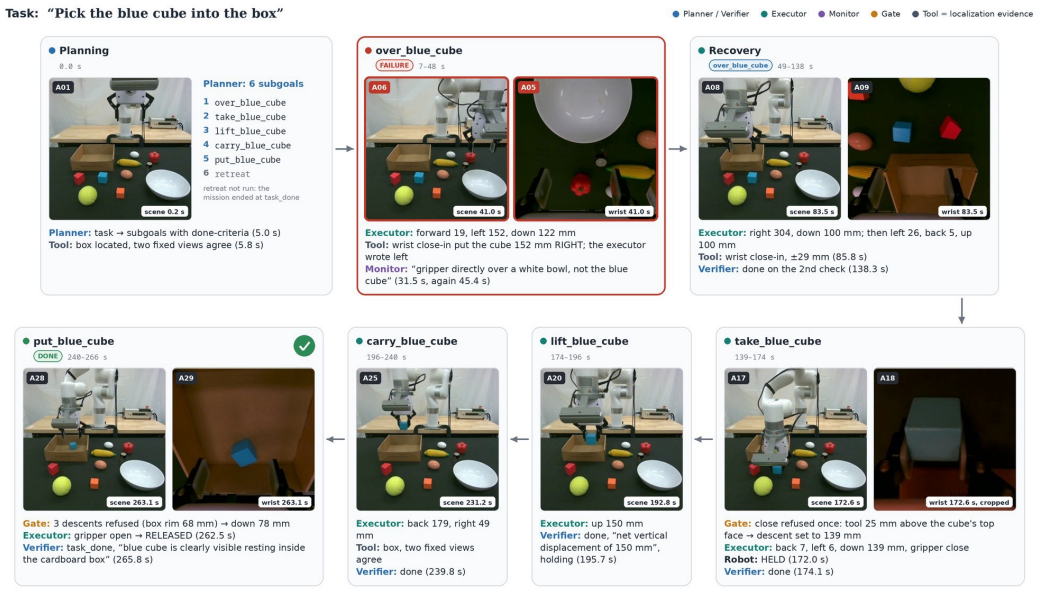

Figure 4: **Real-robot execution and recovery.** In this case, the instruction is to place the blue cube in the box, ***MotorMind*** initially moves toward an incorrect region near the white bowl. The Monitor identifies a mismatch with the intended subgoal. From a new observation, the Executor re-localizes the cube, grasps it, and completes the placement.

are 80% (4/5), 100% (5/5), 100% (5/5), and 60% (3/5). These results highlight ***MotorMind***'s ability to translate semantic descriptions into successful physical actions across varied instruction types.

Qualitative Case Study. Figure 4 illustrates how ***MotorMind*** detects and corrects an execution error. After the Planner decomposes the instruction into subgoals, the initial approach positions the gripper above the white bowl rather than the blue cube. The Monitor flags this mismatch, and the Executor uses updated visual observations to correct the gripper's position before proceeding, and continue the task through placement in the box. This rollout illustrates the role of observation-driven correction during physical execution.

6 Discussion

Backbone Sensitivity We replace Qwen3.8-Flash-Next with GPT-6 Sol (Medium Reasoning). As shown in Figure 5, GPT-6 Sol improves success from 70% to 80% on Spatial, 80% to 100% on Object, and 50% to 70% on Goal. Its average success rate across these suites is 83.3%, compared with 66.7% for Qwen3.8-Flash-Next. This gain comes with longer wall time in every suite, most notably on Spatial (370.4 s versus 199.0 s). These results suggest that the harness can benefit from a stronger reasoning backbone.

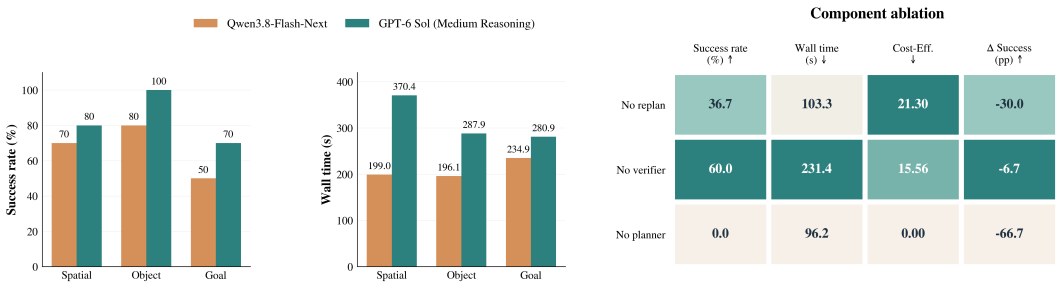

Figure 5: **Backbone sensitivity and component ablations.** Left: success rate and wall time across three LIBERO-PRO base task suites on single seed with Qwen3.8-Flash-Next and GPT-6 Sol (Medium Reasoning) Right: performance after removing replanning, verification, or planning. Δ Success is measured in percentage points relative to the full system's 66.7% success rate.

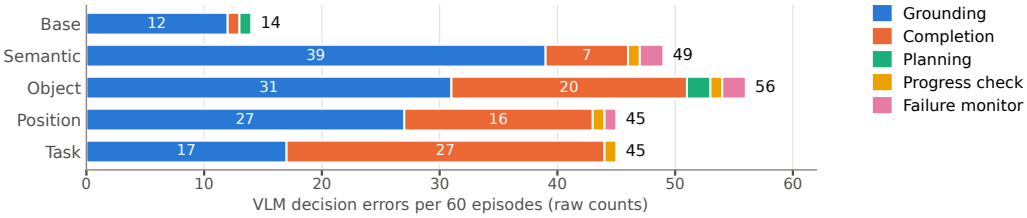

Figure 6: **Failure analysis of VLM decision errors across LIBERO-PRO settings.**

Ablation Study The ablations identify which reasoning functions sustain task success. Removing replanning reduces success from 66.7% to 36.7%, a 30.0 percentage-point drop: the system has fewer opportunities to recover when execution diverges from its plan. Removing the verifier yields 60.0% success, a smaller but measurable 6.7-point drop, indicating that explicit progress checks improve reliability. Removing the planner is most damaging, reducing success to 0.0%. The shorter wall times of some ablated variants therefore do not indicate more effective execution; in particular, the 0.00 Cost-Eff. value without the planner must be read alongside its zero success rate.

Failure Analysis We categorize errors except malformed-output into grounding, completion, planning, progress checking, and failure monitoring. Grounding errors account for the largest share overall, particularly under Semantic, Object, and Position perturbations, while premature completion claims are also a major source of failure, especially for Task and Object perturbations. These results suggest that as foundation models continue to improve, a lightweight harness around a general-purpose VLM may become an increasingly practical path toward robust, general-purpose robotic manipulation. See Appendix G for more details.

7 Conclusion

We presented ***MotorMind***, a zero-shot robotic manipulation harness that uses a single frozen VLM across five complementary roles, connects its reasoning to robot control through a mid-level action representation, and coordinates execution through an asynchronous loop with explicit Monitor and Verifier components. On LIBERO-PRO, ***MotorMind*** achieves 66.7% success on the base suites and 53.8% under perturbations, the highest among the zero-shot methods evaluated; it further reaches 95% success on real-world xArm6 placement tasks, handles adaptive manipulation scenarios with moving objects, and improves to 83.3% average success when using GPT-6 Sol as a stronger backbone. Our failure analysis shows that the remaining errors are dominated by inaccurate visual grounding and premature claims of task completion, suggesting that these bottlenecks can increasingly be addressed by stronger future VLMs with improved visual understanding and self-verification.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, organizing the literature discussed in the related work section, and synthetic benchmark building. We have not used generative AI tools to generate research ideas, develop experimental code, or analyze experimental results. The authors reviewed all AI-assisted text and checked the literature summaries and citations against the original sources to ensure accuracy and appropriate attribution. We take responsibility for the final content of this work, including all text and claims produced with the aid of generative AI.

REFERENCES

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 (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025b. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. SAM 3: Segment anything with concepts. In *International Conference on Learning Representations (ICLR)*, 2026. 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*, 2026a. URL https://arxiv.org/abs/2606.07723.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026b. URL https://arxiv.org/abs/2609.10522.

Yitong Chen, Zezheng Huai, Sixian Li, Yubang Wang, Haozhe Zhang, Yifei Zhang, Hechang Chen, Jingjing Gong, Yu-Gang Jiang, and Xipeng Qiu. ETA: A new agentic paradigm for embodied tasks. *arXiv preprint arXiv:2608.03924*, 2026c. URL https://arxiv.org/abs/2608.03924.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026. URL https://arxiv.org/abs/2606.16458.

Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, Weikai Huang, Xiang Fan, Wei-Chuan Tsai, Shirui Chen, Yi Ru Wang, Shanli Xing, Jaemin Cho, Jae Sung Park, Ainaz Eftekhar, Peter Sushko, Karen Farley, Angad Wadhwa, Cole Harrison, Winson Han, Ying-Chun Lee, Eli VanderBilt, Rose Hendrix, Suveen Ellawela, Lucas Ngoo, Joyce Chai, Zhongzheng Ren, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026. URL https://arxiv.org/abs/2605.02881.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2603.22435.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems (RSS)*, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://arxiv.org/abs/2405.12213.

Meng-Hao Guo, Zhe-Han Mo, Jia-Jun Wang, Yi Zhang, Kejin Wang, Yi-Xuan Deng, Jia-Peng Zhang, Yongming Rao, and Shi-Min Hu. Transferring the intelligence of VLMs to robotic control. *arXiv preprint arXiv:2609.22966*, 2026. URL https://arxiv.org/abs/2609.22966.

Asher J. Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning VLMs into VLAs without catastrophic forgetting. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=sFO9d6XSlf.

Hengyuan Hu, Priya Sundaresan, Jensen Gao, and Dorsa Sadigh. VIA: Visual interface agent for robot control. *arXiv preprint arXiv:2607.11119*, 2026. URL https://arxiv.org/abs/2607.11119.

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, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4573–4602. PMLR, 2025. URL https://proceedings.mlr.press/v270/huang25g.html.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Robotics: Science and Systems (RSS)*, 2025a. doi: 10.15607/RSS.2025.XXI.017. URL https://www.roboticsproceedings.org/rss21/p017.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. 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. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713, 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

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.

Haowen Liu, Xirui Li, Shaoxiong Yao, Peng Shi, Tianyi Zhou, Jia-Bin Huang, Furong Huang, and Jiayuan Mao. Guava: An effective and universal harness for embodied manipulation. *arXiv preprint arXiv:2606.18363*, 2026. URL https://arxiv.org/abs/2606.18363.

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.

Chancharik Mitra, Yusen Luo, Raj Saravanan, Dantong Niu, Anirudh Pai, Jesse Thomason, Trevor Darrell, Abrar Anwar, Deva Ramanan, and Roei Herzig. Mechanistic finetuning of vision-language-action models via few-shot demonstrations. *arXiv preprint arXiv:2511.22697*, 2025. URL https://arxiv.org/abs/2511.22697.

NVIDIA. GR00T N1.5: An improved open foundation model for generalist humanoid robots. NVIDIA Research, jun 2025. URL https://research.nvidia.com/labs/gear/gr00t-n1_5/.

NVIDIA. Cosmos 3: Omnimodal world models for physical AI. *arXiv preprint arXiv:2606.02800*, 2026. URL https://arxiv.org/abs/2606.02800.

OpenAI. GPT-6 Astra: A new generation of intelligence. Official model release, 2026. URL https://openai.com/index/gpt-6-astra/.

Qwen Team. Qwen3.8-Flash-Next: A new architecture, towards ultimate cost-efficiency, August 2026. URL https://qwen.ai/blog?id=qwen3.8-flash-next.

Ameesh Shah, William Chen, Adwait Godbole, Federico Mora, Sanjit A. Seshia, and Sergey Levine. Learning affordances at inference-time for vision-language-action models. *arXiv preprint arXiv:2510.19752*, 2025. URL https://arxiv.org/abs/2510.19752.

Tencent Robotics X, HY Vision Team, Xumin Yu, Zuyan Liu, Ziyi Wang, He Zhang, Yongming Rao, Fangfu Liu, Yani Zhang, Ruowen Zhao, Oran Wang, Yves Liang, Haitao Lin, Minghui Wang, Yubo Dong, Kevin Cheng, Bolin Ni, Rui Huang, Han Hu, Zhengyou Zhang, Linus, and Shunyu Yao. HY-Embodied-0.5: Embodied foundation models for real-world agents. *arXiv preprint arXiv:2604.07430*, 2026. URL https://arxiv.org/abs/2604.07430.

Ziyi Wang, Xumin Yu, Yongming Rao, Yonggen Ling, Yunheng Li, Oran Wang, Mingqi Gao, Yuchen Zhou, Yves Liang, Zuyan Liu, Yani Zhang, Rui Huang, Xiaoran Xu, Bowen Yuan, Yifu Yuan, Xu Tan, He Zhang, Yufei Huang, Shenghao Zhang, Hongsheng Wu, Han Hu, and Zhengyou Zhang. HY-Embodied-VLM-1.0: Efficient physical-world agents. *arXiv preprint arXiv:2607.12894*, 2026. URL https://arxiv.org/abs/2607.12894.

Z.ai. GLM-5.3-Flash: More intelligence with less compute. Official model release, 2026. URL https://autoclaw.z.ai/blog/model/glm-5.3-flash/.

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*, 2026. URL https://arxiv.org/abs/2607.08448.

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.

A Related Work

A.1 Robot Foundation Models

Vision-language-action (VLA) models adapt pretrained vision-language models into robot policies and map observations and language instructions directly to low-level actions (Brohan et al., 2023; Kim et al., 2025b; Ghosh et al., 2024; Black et al., 2025b). More recent systems improve continuous action generation and open-world transfer by leveraging the rapid growth of robot datasets and advances in generative modeling (Black et al., 2025a; NVIDIA, 2025; Fang et al., 2026).

Despite their broader task coverage, the performance of robot foundation policies remains bounded by the coverage of their robot-action training distribution. Deployment to new task settings and embodiments commonly requires collecting in-domain demonstrations and fine-tuning the policy, limiting truly zero-shot transfer(Kim et al., 2025a; Mitra et al., 2025). Moreover, adapting a VLM to low-level actions can weaken the general multimodal understanding, instruction following, and semantic reasoning capabilities that originally motivates the VLA paradigm Hancock et al. (2026).

In contrast, we explore zero-shot robot control with a general-purpose VLM, without task-specific robot data or policy fine-tuning, and directly leverage its spatial priors and semantic knowledge through an explicit primitive action interface.

A.2 Coding Agents for Robot Manipulation

Language models can also control robots by generating programs over robot APIs. Prior works use language models to generate executable robot programs or spatial objectives and constraints that are passed to motion planners (Liang et al., 2023; Huang et al., 2023; 2025). More recent work extends this idea in different ways. CaP-X studies agents that generate, execute, and revise robot programs, ASPIRE acquires reusable skills through exploration, and RHO uses execution feedback to optimize multi-file control software before deployment (Fu et al., 2026; Lu et al., 2026; Elmaaroufi et al., 2026).

However, coding-agent systems face a trade-off between responsiveness and efficiency. Online code revision introduces the latency of multi-turn program generation, while offline variants rely on programs or skills fixed before deployment and cannot adapt to unexpected environmental changes during execution.

A.3 VLM-Based Robot Control

Recent systems combine a high-level VLM with a learned robot policy. LITEN reflects on past execution trajectories to learn which instructions a low-level VLA can perform and uses this experience to improve later plans (Shah et al., 2025). VoLo and Harness VLA similarly use VLMs for planning and orchestration while relying on frozen VLA policies and analytic primitives for physical execution (Chen et al., 2026a; Zhang et al., 2026). These systems add reasoning around learned robot policies, but their physical capabilities still depend on the underlying action models and therefore can't be readily deployed in new settings without adapting the action backend.

Another line of work enables general-purpose VLMs to control robots through compact interfaces composed of translation, rotation, gripper, and other semantic actions (Hu et al., 2026; Guo et al., 2026; Liu et al., 2026). (Hu et al., 2026; Guo et al., 2026; Liu et al., 2026).

Concurrent with our work, Show-Harness introduces a modular, plugin-based framework that augments a sequential observe–reason–act loop with situated planning, action chunking, and predefined failure-recovery routines (Chen et al., 2026b). Both works are motivated by a similar intuition: enabling general-purpose VLMs to control robots through an interface resembling human teleoperation. However, the resulting harness designs differ substantially. First, execution scheduling differs: Show-Harness follows a primarily sequential interaction loop, whereas ***MotorMind*** adopts asynchronous execution in which reasoning and monitoring proceed concurrently with physical actions, leading to a fundamentally different runtime topology. Second, decision autonomy differs: Show-Harness formulates control largely as VLM-based tool selection over predefined Python plugins, while ***MotorMind*** explicitly decomposes decision making across planning, execution, monitoring, and verification, allowing the VLM to reason directly about task decomposition and intermediate de-

Task	Decision being diagnosed	Visual input	Count
Action selection	Primary primitive to execute after the current observation t_1	Scene and wrist views at t_0, t_1	80
Progress verification	Whether the transition from t_0 to t_1 advances the stated subgoal	Scene and wrist views at t_0, t_1	80
Goal completion	Whether the state at t_1 satisfies the stated subgoal and criterion	Scene and wrist views at t_1	80

Table 6: **Composition of the manipulation decision diagnostic.**

cisions. Third, motion parameterization differs: Show-Harness grounds actions through predefined or rule-selected motion increments, whereas our Executor predicts parameterized mid-level actions with VLM-reasoned motion magnitudes. Finally, correction mechanisms differ: Show-Harness primarily performs reactive recovery after execution failures, while ***MotorMind*** supports proactive interruption by monitoring ongoing execution and canceling pending actions when they become inconsistent with the evolving scene. Because Show-Harness appeared concurrently with our work, we focus on clarifying these methodological distinctions rather than providing a direct experimental comparison.

B Diagnostic Evaluation Details

B.1 Data Construction

The diagnostic seperates three local decisions that arise during manipulation: selecting what to do next, deciding whether an observed transition advances the current subgoal, and deciding whether the current state satisfies that subgoal. Separating these questions prevents a single episode-level success indicator from obscuring their different failure modes. The unit of evaluation is therefore one image-conditioned question about a local subgoal, rather than a closed-loop execution of a complete task. Table 6 summarizes the resulting 240-question set.

We constructed the questions from LIBERO expert demonstrations. Codex inspected the expert trajectories and their private state evidence, selected temporal windows, and decomposed each task into local decisions. Rather than sampling frames uniformly, it selected segments with a visually discernible change and a meaningful decision at t_1, then adjusted t_0 and t_1 when the change was too small or the interval extended beyond the relevant decision.

Codex also rewrote complete task instructions into local, observable subgoals and success criteria, such as approaching a handle, aligning the gripper, grasping or lifting an object, carrying it above a destination, lowering and releasing it, opening or closing a drawer, and changing an object's orientation. A single visual segment may support both an action question and a progress question, whereas completion questions use states selected for judging the result. Exact text deduplication gives 138 distinct subgoal strings and 142 distinct `(subgoal, success criterion)` pairs.

B.2 Question Generation and Labels

Question construction jointly specifies the observation window, a local subgoal, its observable success criterion, and the reference answer. During annotation, Codex had access not only to the two camera views shown in each question, but also to private construction-time evidence: end-effector and object poses, joint and gripper states, expert controls, short future windows, simulator predicates, and the outcomes of perturbed and control rollouts.

Codex proposed the initial questions, labels, and refines by jointly inspecting the images and private evidence. These proposals were then reviewed by a human expert, whose feedback was used to correct labels, replace ambiguous frames, revise object relations, and narrow or expand subgoal scope.

The action task asks for one primary next action and uses three JSON schemas: `move` with `forward`, `backward`, `left`, `right`, `up`, or `down`; `rotate` with `yaw_left`, `yaw_right`, `pitch_up`, `pitch_down`, `roll_ccw`, or `roll_cw`; and `gripper` with state `open` or `close`. Both translations and rotations are expressed in the robot base frame using the right-hand rule. The

label specifies only the prioritized primitive, not a distance, angle, speed, or complete executable command.

We determined each reference action from the local subgoal together with the next expert control, the end-effector displacement over a short window, and changes in object and grasp state. For rotational evidence, we computed relative orientation as

$$\Delta R = R_1 R_0^{-1}, \tag{1}$$

converted it to a rotation vector, and expressed that vector in the robot base frame.

Progress is labeled as a function of $(O_{t_0}, O_{t_1}, \text{subgoal}, \text{criterion})$. A transition receives label 1 when it advances the specified subgoal and label 0 when it does not, acts on the wrong object, disrupts a required relation, or moves toward an incompatible state.We construct negative progress examples either by pairing an expert transition with a visually grounded counterfactual subgoal, or by restoring an expert state and executing a physically simulated action perturbation. We retain only perturbations whose realized outcomes do not advance the specified subgoal.

Goal completion is judged only from the two views at t_1 and the explicitly stated success criterion. Label 1 means that the criterion is satisfied; label 0 means that it remains incomplete or that the scene is in an incompatible state. Completion negatives capture two distinct cases: a goal or state mismatch, or correct progress that has not yet met the completion criterion.

B.3 Prompts and Scoring

We use the following prompts for the three diagnostic tasks.

Action Selection Prompt

```
You are evaluating robot observations for an embodied decision task.
    Use only
the supplied images, subgoal and success criterion. Return exactly one
     JSON
object with no explanation.
Images are upright RGB. The images labeled "scene" are from the fixed
    EXTERNAL
camera; the images labeled "wrist" are from the camera mounted on the
    gripper,
which moves with it. Camera motion alone is not object motion. Compare
     the two
views and, when supplied, the two times.
SUBGOAL REFERENCE FRAME: Unless explicitly stated otherwise, left/
    right and
leftmost/rightmost in the natural-language subgoal or success
    criterion refer
to LEFT and RIGHT IN THE FIXED EXTERNAL CAMERA IMAGE (the view labeled
"scene"), as seen by the viewer looking at the displayed image. Image-
    left
means the left side of that image; image-right means the right side of
     that
image. For example, "the bowl to the right of the plate" identifies
    the bowl
on the plate's image-right side in the scene view. These words do not
    specify a
robot-base action direction. Do not use the moving wrist camera to
    redefine
these relations. Object-relative descriptions such as "in front of the
     stove"
retain their contextual spatial meaning; do not automatically
    reinterpret them
as base +X or -X.
ACTION OUTPUT REFERENCE FRAME: Only the direction names in the output
    action
```

```
use the robot BASE coordinate system. Translation: forward=+X,
    backward=-X,
left=+Y, right=-Y, up=+Z, down=-Z. Rotations use the right-hand rule
    about base
axes: yaw_left=+Z, yaw_right=-Z, pitch_up=-Y, pitch_down=+Y, roll_ccw
    =+X,
roll_cw=-X. Base and world axes align in these scenes. A target on
    image-right
does not necessarily require the action "right"; identify the intended
     target
in the scene view, then choose the base-frame motion toward it. The
    reference
frames are distinct; no universal image-left/base-left or
image-right/base-right equivalence is specified.
For action selection, choose the primary NEXT primitive after current
    t1 that
advances the subgoal; do not describe the past motion. Valid outputs:
{"type":"move","direction":"forward|backward|left|right|up|down"},
{"type":"rotate","direction":"yaw_left|yaw_right|pitch_up|pitch_down|
    roll_ccw|roll_cw"},
or {"type":"gripper","state":"open|close"}. Choose a
single value, not the literal pipe-separated list. No distance or
    angle is
requested.
```

Progress Verification Prompt

```
You are evaluating robot observations for an embodied decision task.
    Use only
the supplied images, subgoal and success criterion. Return exactly one
     JSON
object with no explanation.
Images are upright RGB. The images labeled "scene" are from the fixed
    EXTERNAL
camera; the images labeled "wrist" are from the camera mounted on the
    gripper,
which moves with it. Camera motion alone is not object motion. Compare
     the two
views and, when supplied, the two times.
SUBGOAL REFERENCE FRAME: Unless explicitly stated otherwise, left/
    right and
leftmost/rightmost in the natural-language subgoal or success
    criterion refer
to LEFT and RIGHT IN THE FIXED EXTERNAL CAMERA IMAGE (the view labeled
"scene"), as seen by the viewer looking at the displayed image. Image-
    left
means the left side of that image; image-right means the right side of
     that
image. For example, "the bowl to the right of the plate" identifies
    the bowl
on the plate's image-right side in the scene view. These words do not
    specify a
robot-base action direction. Do not use the moving wrist camera to
    redefine
these relations. Object-relative descriptions such as "in front of the
     stove"
retain their contextual spatial meaning; do not automatically
    reinterpret them
as base +X or -X.
```

```
For progress verification, output {"label":1} if the change from t0 to
    t1
advances the stated subgoal, otherwise {"label":0}. A subgoal need not
    be
complete to show progress; preparation counts when it advances the
   stated
subgoal. Mere unrelated motion does not.
```

Goal Completion Prompt

```
You are evaluating robot observations for an embodied decision task.
   Use only
the supplied images, subgoal and success criterion. Return exactly one
    JSON
object with no explanation.
Images are upright RGB. The images labeled "scene" are from the fixed
   EXTERNAL
camera; the images labeled "wrist" are from the camera mounted on the
   gripper,
which moves with it. Camera motion alone is not object motion. Compare
    the two
views and, when supplied, the two times.
SUBGOAL REFERENCE FRAME: Unless explicitly stated otherwise, left/
   right and
leftmost/rightmost in the natural-language subgoal or success
   criterion refer
to LEFT and RIGHT IN THE FIXED EXTERNAL CAMERA IMAGE (the view labeled
"scene"), as seen by the viewer looking at the displayed image. Image-
   left
means the left side of that image; image-right means the right side of
    that
image. For example, "the bowl to the right of the plate" identifies
   the bowl
on the plate's image-right side in the scene view. These words do not
   specify a
robot-base action direction. Do not use the moving wrist camera to
   redefine
these relations. Object-relative descriptions such as "in front of the
    stove"
retain their contextual spatial meaning; do not automatically
   reinterpret them
as base +X or -X.
For goal completion, output {"label":1} only if the stated subgoal and
    its
success criterion are satisfied at t1, otherwise {"label":0}. If
   release is
required, holding an object at the target is insufficient.
```

Table 7 reports action-selection accuracy separately for translation, rotation, and gripper primitives.

C MotorMind Implementation Details

C.1 VLM Roles and Prompts

MotorMind separates task-level reasoning, action generation, execution monitoring, outcome verification, and memory construction through role-specific VLM calls. The separation is enforced by both the information supplied to each call and its structured output schema. In particular, the executor receives one subgoal rather than the full planning problem, while the motion supervisor

Model	Translation (48)	Rotation (19)	Gripper (13)
Qwen3.8-Flash-Next-FP8	47.92% (23/48)	0.00% (0/19)	46.15% (6/13)
HY-Embodied-0.5 MoT-2B	20.83% (10/48)	15.79% (3/19)	23.08% (3/13)
Hy-Embodied-VLM-1.0 A3B	31.25% (15/48)	0.00% (0/19)	0.00% (0/13)
Cosmos3-Nano (understanding tower)	27.08% (13/48)	0.00% (0/19)	15.38% (2/13)
GLM-5.3-Flash	41.67% (20/48)	15.79% (3/19)	53.85% (7/13)
GPT-6 Astra (medium)	58.33% (28/48)	63.16% (12/19)	61.54% (8/13)

Table 7: **Action-selection results by primitive type.** Accuracy is reported separately for translation, rotation, and gripper questions.

and scene monitor have no action field and therefore cannot silently replace an action already being executed.

Planner Prompt

```
You turn a task told in one sentence into a list of small subgoals
for a robot arm.  You do not move the robot:  another agent, the
executor, takes one subgoal at a time and proposes the motions itself.
Everything you have worked out must reach it as the wording of a
subgoal, because the wording and the criterion are the only things
it is given.  You answer with one JSON object and nothing else.
```

Replanner Prompt

```
A plan you wrote is part-way through and something has gone wrong.
You write the steps that are still to come.  The finished ones are
kept by the caller and must not appear again.  You answer with one
JSON object and nothing else.
```

Executor Prompt

```
You decide what a robot arm should do next.  You are given a subgoal,
what the cameras see now, and what the robot reports about itself.
You answer with one JSON object and nothing else.  You are not the
planner:  choose the next few actions that move the subgoal forward,
not the whole task.  They may be a move across the table, a descent, a
lift, a turn, a gripper command or a wait -- whatever the situation in
front of you actually calls for.
```

Motion Supervisor Prompt

```
You are watching one motion that is already running on a robot arm.
Your only decision is whether it should carry on or stop now.  You
cannot start, change, extend or replace a motion.  You answer with one
JSON object and nothing else.
```

Scene Monitor Prompt

```
You are watching a robot arm while one step of a task runs.  You are
the only one looking at the task as a whole; another agent is already
judging each individual motion, so that is not your job.  You cannot
start, stop, change or suggest a motion.  You answer with one JSON
object and nothing else.
```

Verifier Prompt

```
You decide whether one step of a robot task actually happened.  You
are shown the scene before the step and the scene now, and what the
robot that performed it claims.  You are not that robot and you are
not obliged to agree with it.  You answer with one JSON object and
nothing else.
```

Memory Manager Prompt

```
You keep the memory of a robot run.  You are given what the robot was
asked to do, the plan it is following, the note you wrote last time,
and the log of what has happened since -- one line per motion, with
the numbers the robot measured.  You answer with one JSON object and
nothing else.  You do not decide what the robot does next and you
never write an action:  you write down what is now known, so that the
planner can decide better.
```

Target Locator Prompt

```
You look at pictures of a table and say what is on it and where.  You
answer with one JSON object and nothing else.  You never describe a
motion and you never mention the robot.
```

The prompts above are incomplete excerpts that summarize the role definitions. We will provide the complete prompt files in the supplementary material.

D Simulation Evaluation Details

D.1 LIBERO-Pro Tasks and Perturbations

We evaluate three LIBERO task families: Spatial, Object, and Goal. Each family contains 10 tasks evaluated under five conditions: Base, Language, Object, Position Swap, and Task. Base is the unperturbed reference condition; the other four conditions apply the corresponding benchmark-provided perturbations.

Language perturbations vary the wording of the task instruction. Object perturbations modify scene objects according to the benchmark definitions. Position Swap perturbations alter object placements, while Task perturbations change the instructed target or goal. We use the provided configurations for each condition. Table 8 summarizes the evaluation conditions.

The full evaluation matrix comprises 4 task families $\times$ 5 conditions $\times$ 10 tasks, yielding 200 task configurations: 40 base configurations and 160 perturbed configurations. These are task configurations rather than rollout counts; evaluation across multiple initialization seeds produces additional rollouts. Success is determined by the environment's predicate-based checks for the evaluated task.

D.2 Baseline Configurations

We evaluate direct vision-language-action (VLA) policies and agent frameworks that combine planning with learned or programmatic execution. The direct VLA baselines comprise OpenPI $\pi_{0.5}$(Black et al., 2025a), OpenVLA(Kim et al., 2025b), OpenVLA-OFT(Kim et al., 2025a), GR00T N1.5(NVIDIA, 2025), and MolmoAct2(Fang et al., 2026). The agent baselines comprise Harness VLA(Zhang et al., 2026), VoLo(Chen et al., 2026a), and CAP-X(Fu et al., 2026). All evaluated VLA weights remain frozen; we perform no additional policy training or fine-tuning. For all LIBERO-Pro configurations that use $\pi_{0.5}$, including its use within Harness VLA and VoLo, we use the same OpenPI `libero_base` checkpoint. Table 9 summarizes their control and planning mechanisms.

Condition	Configuration key	Description
Base	`base`	Original, unperturbed task setting
Language	`lan`	Instruction wording perturbations
Object	`object`	Benchmark-defined object perturbations
Position Swap	`swap`	Object placement perturbations
Task	`task`	Target or goal perturbations

Table 8: **LIBERO-Pro evaluation conditions.** Each condition is applied to every evaluated task family.

Baseline	Control mechanism	Planning and memory
OpenPI $\pi_{0.5}$	Direct VLA action prediction	No external planner or task memory
OpenVLA	Autoregressive action-token prediction	No external planner or task memory
OpenVLA-OFT	Continuous action-chunk prediction	No external planner or task memory
GR00T N1.5	Native action-chunk prediction	No external planner or task memory
MolmoAct2	Continuous action prediction	No external planner or task memory
Harness VLA	Tool-based execution with frozen $\pi_{0.5}$	VLM planning and task-specific memory
VoLo	VLM-guided execution with frozen $\pi_{0.5}$	VLM planning with visual perception tools
CAP-X	Generated Python programs over robot tools	Program generation and visual execution feedback

Table 9: **Baseline configurations.** Direct policies predict robot actions, whereas agent frameworks add planning, tools, or programmatic execution around their control policy.

Direct VLA Policies: Shared Setup. Direct VLA policies receive the task instruction and each model's supported observations, without MotorMind-generated subgoals, memory, or action corrections. We preserve model-specific image preprocessing, state encoding, action decoding, and normalization. Predicted actions are converted to LIBERO's seven-dimensional control interface. Each policy follows its configured action execution schedule before acquiring a new observation; chunk lengths are not assumed to be identical across architectures.

OpenPI $\pi_{0.5}$. We use the OpenPI `libero_base` $\pi_{0.5}$ checkpoint as the direct $\pi_{0.5}$ baseline and as the shared $\pi_{0.5}$ execution policy in the agent frameworks. The policy receives visual observations, the language instruction, and robot state, and predicts continuous action chunks. We retain the corresponding LIBERO observation and action transforms and normalization. In direct evaluation, the policy is conditioned on the task instruction; within an agent framework, it receives the instruction selected by that framework. Its weights remain frozen in both cases.

OpenVLA and OpenVLA-OFT. OpenVLA uses the published `openvla/openvla-7b` checkpoint with its autoregressive action decoder. It receives a single scene image and the task instruction, without proprioceptive input, and executes one action before the next observation. The implementation retains the original action decoder and uses suite-specific LIBERO normalization statistics.

OpenVLA-OFT uses the published `moojink/openvla-7b-oft-finetuned-libero-spatial-object-goal-10` checkpoint with its continuous L1 action head and proprioceptive projector. It receives scene and wrist images, the instruction, and robot state, and executes eight-action chunks. We retain the model-specific image preprocessing, proprioceptive normalization, and LIBERO action transforms. Both policies are evaluated without additional weight updates.

GR00T N1.5. For the LIBERO-adapted configuration, we use `youliangtan/gr00t-n1.5-libero-spatial-posttrain` across task families, without switching to a different checkpoint for each family. The policy receives scene and wrist images together with robot state and language. We retain its native LIBERO normalization and eight denoising steps. It predicts 16 actions, of which the first is executed before observing again.

The raw-base configuration uses `nvidia/GR00T-N1.5-3B` with its pretrained DROID embodiment and statistics. A DROID-to-LIBERO adapter converts the observation and action interfaces,

and 16 actions are executed between observations. This configuration evaluates embodiment transfer and is distinct from the LIBERO-adapted policy; its results retain a separate baseline label.

MolmoAct2. We use the published `allenai/MolmoAct2` and `allenai/MolmoAct2-LIBERO` checkpoints for the base and LIBERO-adapted configurations, respectively. Both use continuous-action inference with ten flow steps, normalized language, and no depth-reasoning mode. The input contains scene and wrist images, the task instruction, and robot state; ten predicted actions are executed before the next observation.

The LIBERO-adapted policy uses its own LIBERO statistics. The base-weight configuration uses LIBERO-derived normalization and robot metadata while retaining the base model weights. Accordingly, the base configuration involves no additional policy training, but is not free of LIBERO-derived interface information. These VLA policy can also be the ablation for mid-level action representation.

Harness VLA. Harness VLA combines a Qwen3.8-Flash-Next-FP8 planner with perception and execution tools, including SAM3 and the frozen OpenPI `libero_base` $\pi_{0.5}$ policy. The planner selects tool calls and execution instructions, while the VLA provides the learned manipulation component. We retain the framework's native tool interface rather than replacing it with MotorMind's semantic-action executor.

Harness VLA additionally constructs task-specific memory through exploration. Each task starts from empty memory on seed 0, and the resulting memory is frozen at the designated exploration budget before evaluation on new initial states. The evaluated memory-budget configurations use snapshots after five or ten exploration attempts. Exploration changes the stored memory rather than the VLA weights. Exploration and evaluation costs are treated separately, and memory budgets are identified with their corresponding results.

VoLo. VoLo retains its Qwen3.8-Flash-Next-FP8 planner and SAM3/Molmo perception tools. The planner uses visual context and tool outputs to guide execution by the frozen OpenPI `libero_base` $\pi_{0.5}$ policy. We retain the framework's LIBERO observation and action transforms and normalization, with five actions executed before the next policy observation. No additional policy training is performed. VoLo is evaluated as an agent framework with its own planning and perception workflow; it does not receive MotorMind's memory, task-clause ledger, or geometric action corrections. VoLo is sequential execution agent, which can also serve as the ablation for our asynchronous harness system.

CAP-X. CAP-X is a program-generating baseline. It retains the agent0 `ReducedSkillLibrary` interface and generates executable Python code from visual observations and the task instruction. Generated programs invoke the available robot tools, and subsequent visual feedback supports correction. Unlike the VLA-based frameworks above, CAP-X does not use $\pi_{0.5}$ as its execution policy.

Our implementation provides RGB-D observations and uses joint-position control. Candidate program generation uses GPT-5.6-terra and Claude-Opus-5, with four candidates from each model; GPT-5.6-terra performs synthesis and visual differencing. The method retains its code-generation and feedback workflow rather than adopting MotorMind's planner or action interface. Evaluation uses the applicable environment task definition and predicate-based success checks.

E Adaptive Reasoning Tasks

We evaluate adaptive reasoning in four online manipulation suites comprising 30 tasks. The suites test responses to changes in object location, continuously moving targets, semantic and temporal target-selection requirements, and revised instructions. Task definitions and intervention logic are implemented in the environment. Object identities and state variables used to trigger interventions or score outcomes are not supplied as privileged policy inputs. This study be framed as stress tests rather than precise quantitative evidence.

Suite	Tasks	Evaluation focus
Interactive	10	Adapt to object or receiver displacement with an unchanged instruction.
Pure Dynamic	5	Identify, intercept, and place a named moving target.
Dynamic Reasoning	10	Resolve semantic, relational, and temporal references to moving targets.
Prompt Change	5	Respond to revised instructions without resetting the physical scene.

Table 10: **Adaptive reasoning task suites.** The four suites contain 30 online manipulation tasks in total.

Task	Instruction	Intervention
1	put the ketchup in the wooden tray	Receiver during transport; 1 displacement
2	put the bowl on the plate	Receiver during transport; 1 displacement
3	put the wine bottle on the rack	Target object before grasping; 1 displacement
4	put the cream cheese in the bowl	Receiver during transport; 1 displacement
5	put the bbq sauce in the wooden tray	Target object before grasping; 2 displacements
6	put the milk in the basket	Receiver during transport; 2 displacements
7	put the chocolate pudding in the bowl	Receiver during transport; 2 displacements
8	put the salad dressing in the wooden tray	Target object before grasping; 3 displacements
9	put the orange juice in the basket	Target object before grasping; 3 displacements
10	put the butter in the bowl	Receiver during transport; 3 displacements

Table 11: **Scene shift tasks.** The instruction remains unchanged while the target object or receiving container is displaced.

E.1 Details of the Four Task Suites

Table 10 summarizes the four suites and their evaluation focus.

Scene Shift. The instruction remains fixed while the environment displaces either the object to be grasped or its receiving container. Interventions are triggered by physical progress, such as approaching an object before grasping or approaching a receiver after lifting the payload. Tasks include one, two, or three displacements, testing both initial adaptation and repeated correction. The number of displacements in Table 11 specifies the configured maximum; later interventions require the corresponding progress triggers to be reached.

Dynamic Manipulation. These tasks use a single-pass conveyor. The instruction names the target, and the agent must locate it among the moving objects, grasp it, and place it in the designated receiver. The selected configurations include conveyor motion at 1.5 mm/s. Objects follow scripted conveyor motion until a two-finger grasp transfers them to ordinary rigid-body dynamics. Targets that are not grasped can leave the available conveyor region; the motion does not loop to provide repeated passes.

Dynamic Reasoning. These tasks combine conveyor manipulation with target selection from semantic properties, exclusion, spatial relations, or observation history. Spatial references explicitly refer to the initial arrangement.

Prompt Shift. The environment replaces the instruction after a predefined physical trigger while preserving the robot and scene state. The tasks cover destination redirection, insertion of an intermediate operation, an obstacle reminder, and target-object replacement. The agent must adapt from its current physical state rather than restart the original task.

The first three instruction changes occur during transport after the object has been lifted and approached toward its original destination. Task 4 introduces an obstacle and issues a reminder after the gripper-closing event. Task 5 redirects the target during the pre-grasp approach. The trigger for Task 4 uses a positive closing command and three consecutive steps with less than 1 mm change in gripper width; this event alone does not certify a successful grasp. We note that the diagnostic is not intended to establish reliable low-level control, but to characterize VLMs' local decision-making capabilities and limitations, motivating a mid-level action interface with continuous feedback and correction.

Task	Instruction
1	pick up the red mug from the conveyor belt and place it in the basket
2	pick up the alphabet soup from the conveyor belt and place it in the basket
3	pick up the ketchup from the conveyor belt and place it in the red basket
4	pick up the blue-and-white cream cheese box from the conveyor belt and place it in the wooden tray
5	pick up the green ketchup from the conveyor belt and place it in the basket

Table 12: **Dynamic manipulation tasks.** Each instruction identifies a named target moving on a single-pass conveyor.

Task	Reasoning requirement	Instruction
1	Contents: liquid	pick up the packaged product containing liquid from the conveyor belt and place it in the basket
2	Category: edible item	pick up the edible item from the conveyor belt and place it in the basket
3	Purpose: beverage	pick up the beverage meant for drinking from the conveyor belt and place it in the basket
4	Category exclusion: not a bowl	pick up the drinking vessel that is not a bowl from the conveyor belt and place it in the basket
5	Attribute exclusion: not red	pick up the cup that is not red from the conveyor belt and place it in the basket
6	Initial spatial relation	pick up the object that is between the two bowls in the initial arrangement on the conveyor belt and place it in the basket
7	Cumulative temporal counting	Select the second distinct food item over the entire episode, then pick it up downstream of the yellow line and place it in the basket.
8	Purpose: seasoning, not drinking	pick up the bottled product used to season food rather than to drink from the conveyor belt and place it in the basket
9	Contents: fruit drink	pick up the drink made from fruit from the conveyor belt and place it in the basket
10	Category and initial spatial relation	pick up the food item that is between the two cups in the initial arrangement on the conveyor belt and place it in the basket

Table 13: **Dynamic reasoning tasks.** The target is specified through semantic, exclusion-based, spatial, or temporal constraints.

E.2 Dynamic Environment Protocol

Simulation stepping during planning. The dynamic protocol advances the environment according to elapsed wall-clock time while the agent is planning or waiting for model inference. At the 20 Hz control rate, the clock schedules one simulation step for each accumulated 0.05 s of non-action wall time. These steps keep the conveyor and environment dynamics progressing while the robot receives a zero-motion command. Planning latency therefore contributes to elapsed scene evolution rather than providing a deliberately paused environment. Robot actions retain their native execution schedule.

Observations and interventions. Agents receive the task instruction, their supported camera observations, and robot measurements through their respective interfaces. They are not given ground-truth target identities, scripted object trajectories, intervention triggers, or evaluator state. Interactive interventions alter the physical scene while retaining the instruction. Prompt Change retains the physical scene and updates the instruction at the trigger. All methods for a given task use the same task definition and intervention rule, although the trigger may be reached at different times.

Instruction-update handling. Direct VLA policies discard queued actions associated with the old instruction and use the revised text on subsequent inference. CAP-X interrupts the old program at an action boundary and updates the generation goal. ***MotorMind*** retains the same mission and interrupts only the active subgoal. The planner generates a new plan for the revised instruction using the current observation, the previous plan, accumulated memory, and recent execution evidence. The physical environment is not reset, and cancellation and switching time remain part of the episode budget.

Episode setup and budget. The online suites use a 600 s wall-clock budget. Interactive and Prompt Change use ten settling steps; the dynamic suites use 32 settling steps and an additional 20,000-step horizon.

Success criteria. Task success is assessed using environment predicates rather than an agent's declaration of completion. For tray placement in Prompt Change Tasks 2 and 5, scoring requires the object origin to lie within the oriented tray region, upward load-bearing support from the tray, and

Task	Initial instruction	Replacement instruction
1	put the ketchup in the wooden tray	Change of plan: put the ketchup on the plate instead of in the wooden tray.
2	Put the chocolate pudding on the plate.	Change of plan: put the chocolate pudding in the wooden tray instead of on the plate.
3	Put the chocolate pudding on the plate.	First put the chocolate pudding down on the table, turn on the stove and keep it on, and then continue putting the chocolate pudding on the plate.
4	put the chocolate pudding on the plate	Avoid the obstacle and continue putting the chocolate pudding on the plate.
5	Put the BBQ sauce in the wooden tray.	Change of plan: leave the BBQ sauce on the table and put only the chocolate pudding in the wooden tray.

Table 14: **Prompt shift tasks.** The instruction changes after a physical trigger without resetting the scene or robot.

Budget	Goal	Spatial	Object	Avg. SR	Time (s)	Cost-Eff.
300 s	40.0%	70.0%	90.0%	**66.7%**	**159.8**	**25.03**
450 s	50.0%	70.0%	80.0%	**66.7%**	210.0	19.05
900 s	40.0%	60.0%	50.0%	50.0%	320.0	9.37
3,600 s	50.0%	60.0%	70.0%	60.0%	296.4	12.14

Table 15: **Execution-budget ablation.** Average success does not increase monotonically with the maximum execution budget. The 300 s setting matches the success of the 450 s setting while requiring less wall time.

release from the gripper. The implemented stove-interruption check records whether the stove was on before final placement; it does not independently verify the requested intermediate table placement or that the stove remains on at termination. For intervention suites, task success and whether the intervention trigger was reached are separate outcomes; episodes that fail before triggering an intervention remain part of the task evaluation.

F Ablation Study Detail

Effect of Execution Budget. Table 15 shows that both the 300 s and default 450 s settings achieve 66.7% average success. The shorter budget reduces mean wall time from 210.0 s to 159.8 s, a 23.9% reduction, and increases cost-efficiency from 19.05 to 25.03. Extending the budget to 900 s lowers average success to 50.0%; the 3,600 s setting reaches 60.0%. Thus, additional time alone does not consistently improve task completion in this evaluation. An effective harness must use new observations to decide when further action is useful and when execution should stop.

G Error Analysis Details

Data. The analysis covers the 300 LIBERO-PRO episodes behind the ***MotorMind*** rows of Tables 3: ten tasks per suite, two seeds each, for the Goal, Spatial, and Object suites, unperturbed and under each of the four perturbations. Episodes run with a 450 s budget and stop as soon as the environment's goal predicate holds. The evaluation records the simulator state at every executor observation and after every subgoal: each object's pose, the finger gap, and each goal predicate. Nothing from this record reaches a model. The labels below are computed from this record and from the harness's own event log.

Each failed episode is assigned its last unrecovered failure in Figure 7; *Dropped* there includes the four episodes that end with the target knocked over.

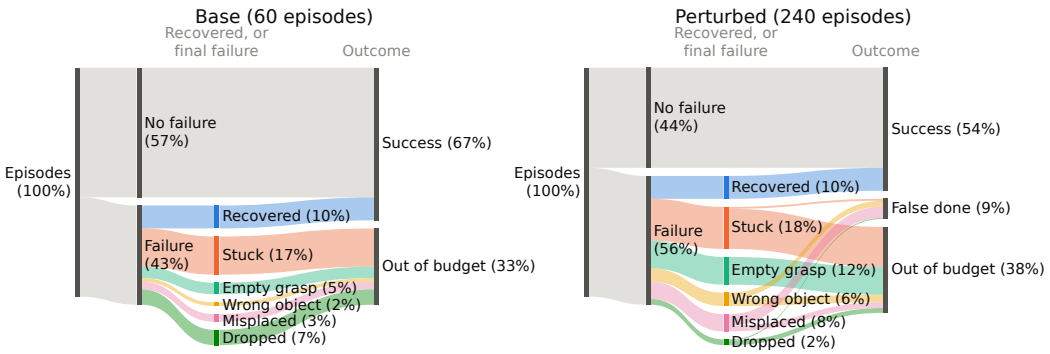

Figure 7: **Physical failure analysis** tracing ***MotorMind*** episodes through failures, recovery, and outcomes on unperturbed (left) and perturbed (right) LIBERO-PRO tasks. Each failed episode is assigned its last unrecovered failure; *Recovered* episodes had failures and recovered from all of them. *False done*: the model declared the task complete while its goal predicate was false. Band thickness is proportional to the share of episodes within each panel.