EM-VLA: CLOSING THE LOOP BETWEEN EXECUTION MONITORING AND ACTION GENERATION FOR LONG-HORIZON MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models have demonstrated strong capabilities in robotic manipulation, yet reliable long-horizon execution remains challenging. Existing approaches introduce high-level reasoning or execution monitoring, but execution feedback is often used primarily to trigger intervention rather than continuously inform both reasoning and control. We propose EM-VLA, a hierarchical VLA that coordinates high-level reasoning and low-level control through a persistent state maintained by a lightweight execution monitor. The monitor integrates observed transitions, visual-language context, and internal action-model responses to update this state across interactions. The state guides subtask transitions and replanning while adapting action generation to the current execution stage. Internal responses from action generation then inform subsequent state updates, closing the loop between monitoring and control. This design enables the VLA to revise its plan and behavior as execution conditions change without requiring high-level reasoning at every action query. To support training, we derive temporally aligned subtask plans and supervision for reasoning and monitoring from successful demonstrations. Evaluations on LIBERO, CALVIN, RoboTwin 2.0, and real-world robotic tasks show improved long-horizon performance over the compared baselines and stronger recovery from execution-time perturbations.

1 INTRODUCTION

Large-scale vision-language pretraining has produced models with broadly transferable visual and semantic representations (Radford et al., 2021; Alayrac et al., 2022). Building on these advances, vision-language-action (VLA) models have been developed to generate robot actions from images and language instructions, from discrete action tokens (Zitkovich et al., 2023; Kim et al., 2024) to continuous action chunks (Black et al., 2025; Physical Intelligence et al., 2025). In real-world manipulation scenarios, long-horizon tasks require the coordination of sequences of interdependent actions under uncertainty in physical outcomes. For example, a failed grasp may leave an object in place while subsequent actions assume it has been moved. Conversely, an unexpected displacement may invalidate later steps. Because instantaneous observations are inadequate for inferring the efficacy of prior actions, robust long-horizon control requires maintaining and continually updating an internal state representation of interaction history, and conditioning subsequent action selection on this evolving task context.

Existing approaches address the challenges from two complementary perspectives. One line of work focuses on structuring the task before and during execution. Language-model-based planners decompose high-level instructions into executable skills, while feedback-aware planners incorporate observations to revise subsequent plans (Ahn et al., 2022; Huang et al., 2023). Hierarchical VLAs and embodied reasoning methods further introduce language motions, semantic subtasks, and grounded predictions to guide low-level actions (Belkhale et al., 2024; Physical Intelligence et al., 2025; Huang et al., 2025; Shi et al., 2025; Zhao et al., 2025). A complementary line of work focuses on monitoring the execution process itself. Rather than directly generating actions, these methods estimate task success, learn dense value signals, or detect execution failures, providing feedback for further reasoning or intervention (Du et al., 2023; Ma et al., 2023; Xu et al., 2025; Yang et al., 2025; Ma et al., 2026). Despite these advances, the roles of planning and execution feedback remain

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.

largely separated. Execution information is typically reduced to a score, warning, or trigger for subsequent reasoning, rather than maintained as a persistent source of information for action generation. This motivates a central question: *how can execution information be maintained and reused to jointly guide high-level reasoning and low-level control throughout long-horizon execution?*

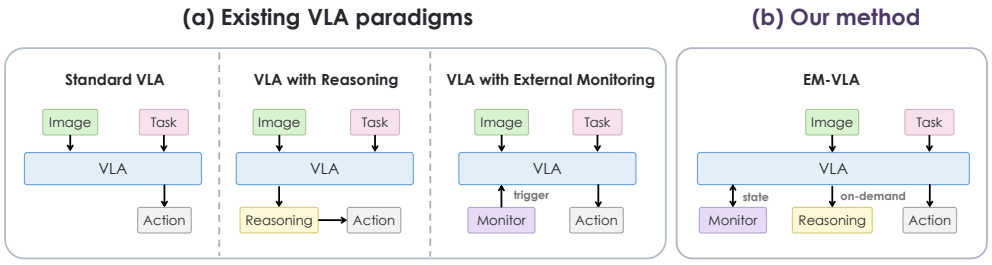


Figure 1: Comparison of VLA execution paradigms. (a) Existing approaches directly predict actions, introduce intermediate reasoning, or use external monitoring to trigger intervention. (b) EM-VLA maintains a persistent execution state through monitor–VLA interaction to guide high-level reasoning and low-level control.

To address this question, we investigate the existing VLA paradigms, as illustrated in Figure 1, which either repeatedly invoke high-level reasoning or use external monitoring mainly to trigger further reasoning or correction. Therefore, we introduce EM-VLA, a hierarchical VLA that coordinates high-level reasoning and low-level control through a persistent state maintained by a lightweight execution monitor. Firstly, the vision-language backbone generates a subtask plan when needed and caches it in plan memory, while the VLA generates actions conditioned on the active subtask. After action execution, a lightweight monitor updates the state from the observed transition, the VLA's visual-language context, and adapter-routing and residual-magnitude signals produced during action generation. The resulting state estimates progress within the current subtask, whether execution matches that subtask, and overall task completion. These estimates guide subtask transitions and replanning, while the execution state routes lightweight adapters in the action expert for stage-dependent action generation. Adapter responses then inform the next state update, closing the loop between monitoring and control without requiring high-level reasoning at every action query.

We evaluate EM-VLA on LIBERO, CALVIN, RoboTwin 2.0, and real-world robotic manipulation tasks. It improves over the evaluated VLA baselines across these settings, achieving 97.6% average success on LIBERO and 90.9% success in RoboTwin's randomized setting. For long-horizon execution, it achieves 96.6% success on LIBERO-Long, completes an average of 4.35 tasks per CALVIN sequence, and attains 80% average success on real robots. These results indicate more reliable long-horizon execution across simulated and physical settings. Our contributions are summarized as follows:

- We introduce a lightweight monitor that maintains a persistent execution state across interactions and uses it to guide subtask transitions and replanning, which bridges the gap between execution monitoring and action generation.
- We couple the execution monitor with the VLA through a routing and feedback mechanism. The execution state guides adapter routing, while the VLA's internal responses feed back into the monitor, closing the loop between monitoring and control.
- We develop a pipeline that turns successful demonstrations into temporally aligned subtask plans, providing supervision for both intermediate reasoning and execution monitoring. Evaluations across simulation and real-world tasks show that our method outperforms the compared baselines, particularly in long-horizon manipulation.

2 Related Work

Vision-Language-Action (VLA) models leverage pretrained vision-language representations to map visual observations and language instructions to robot actions (Radford et al., 2021; Alayrac et al., 2022). Existing VLAs mainly follow two action-generation paradigms: autoregressively generate

discrete action tokens (RT-2 (Zitkovich et al., 2023) and OpenVLA (Kim et al., 2024)) and generate continuous actions (π_0 (Black et al., 2025), OpenVLA-OFT (Kim et al., 2025)). As manipulation tasks scale from short-horizon single-stage settings to long-horizon multi-stage execution, direct observation-to-action mapping struggles to balance high-level planning with low-level control. Recent work introduces intermediate representations and hierarchical reasoning. For example, Hi Robot (Shi et al., 2025) and $\pi_{0.5}$ (Physical Intelligence et al., 2025) organize low-level control through high-level language or semantic subtasks. ECoT (Zawalski et al., 2024), CoT-VLA (Zhao et al., 2025), and ThinkAct (Huang et al., 2025) use reasoning traces, visual subgoals, and reinforced visual latent planning, respectively, to improve task decomposition and action generation. OneTwoVLA (Lin et al., 2025) switches between recomputing and reusing reasoning results to reduce redundancy. Complementarily, another line of work predicts future states through world-action modeling to support action generation. GR-1 (Wu et al., 2023) and GR-2 (Cheang et al., 2024) jointly model future images and actions. Motus (Bi et al., 2025) unifies video generation and action modeling. VPP (Hu et al., 2025) and UP-VLA (Zhang et al., 2025) use future prediction to enhance policy representations or training objectives. However, these methods lack continual verification and correction of high-level reasoning or future-state predictions using execution feedback, allowing reasoning or stage-transition errors to accumulate over long-horizon execution.

Robot execution monitoring provides feedback on task outcomes, progress, and deviations. SuccessVQA (Du et al., 2023) assesses goal completion, while LIV (Ma et al., 2023), RoboDopamine (Tan et al., 2025), and VLAC (Zhai et al., 2025) provide denser value, reward, or progress signals. These signals characterize different aspects of execution, but do not directly specify how subsequent plans or actions should be adjusted. Runtime detectors such as FAIL-Detect (Xu et al., 2025), Sentinel (Agia et al., 2025), and SAFE (Gu et al., 2025) identify deviations using observations, actions, or internal model signals. They can flag execution failures, yet a warning alone offers limited guidance on the subsequent plan or action. Recovery-oriented methods such as AHA (Duan et al., 2024) and RACER (Dai et al., 2024) use visual or language feedback to diagnose failures and guide corrective behavior. More closely related, FPC-VLA (Yang et al., 2025) employs a supervisor to assess and correct predicted actions, whereas CycleVLA (Ma et al., 2026) combines progress-aware execution with failure-triggered subtask backtracking and retries. However, these methods use execution feedback chiefly for local correction or recovery decisions, leaving its potential to continuously shape subsequent action generation underexplored.

3 METHOD

3.1 PROBLEM FORMULATION

In vision-language-action (VLA) tasks, an agent must execute a language-specified task through a sequence of actions while continuously adapting to the evolving execution state. The task should first be decomposed into sequential subtasks, and we manage them as a cached task plan memory $\mathcal{P} = (p_1, \ldots, p_M)$, where M denotes the number of subtasks. During action generation, each episode consists of multiple decision rounds. In each round, the policy generates an action chunk, executes a prefix of it, and receives a new observation before the next round. At decision round t, the VLA model receives a language instruction ℓ, the active subtask p_j from the cached task plan $\mathcal{P}$, and the current observation o_t, which contains visual and proprioceptive information. In addition, an execution state h_t summarizes the execution evidence accumulated in previous decision rounds. Conditioned on these inputs, the model generates an H-step action chunk as follows:

$$A_t = (a_{t,1}, \ldots, a_{t,H}) \sim \pi_\theta(\cdot \mid o_t, \ell, p_j, h_t). \tag{1}$$

After generating the action chunk, the policy executes only a prefix $A_t^{\text{exec}} = (a_{t,1}, \ldots, a_{t,H_e})$, where $H_e \leq H$, before querying the model again with an updated observation. The executed prefix A_t^{exec} and resulting observation o_{t+1} provide the evidence for the next execution-state update.

3.2 ARCHITECTURE OVERVIEW

We introduce EM-VLA, a hierarchical VLA policy built around a persistent execution state that closes the loop between execution monitoring and action generation, as illustrated in Figure 2. The proposed framework comprises three components: (i) a vision–language backbone, (ii) an action expert, and (iii) a lightweight execution monitor. The vision–language backbone generates and caches

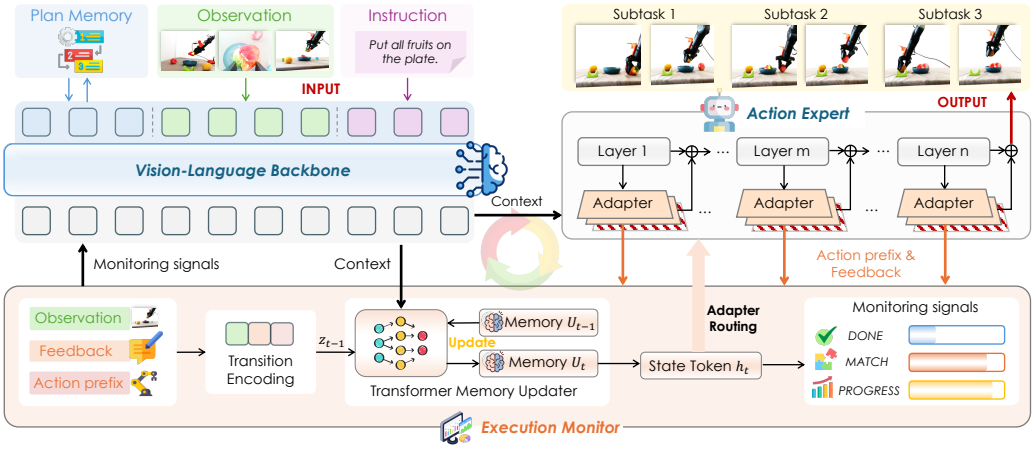


Figure 2: Overview of EM-VLA. The vision-language backbone provides plans and context. The monitor updates execution state from observed transitions and action-expert feedback to guide subtask transitions, replanning, and adapter routing, closing the loop between monitoring and control.

an ordered subtask plan for high-level reasoning and provides multimodal context for action generation and monitoring. During execution, an explicit execution state guides subtask transitions and replanning. The action expert uses lightweight routing adapters to adjust actions to the current phase. After each step, the execution monitor updates the state from observed transitions, vision–language context, and the action expert's internal signals. This bidirectional coupling forms a closed loop: state guides action generation, and the action expert's activations inform the updated state.

3.3 Coupling Execution Monitoring with Action Generation

As aforementioned, we couple execution monitoring with action generation through a closed-loop process. Specifically, at each decision round, the execution state guides plan-level decisions and action generation, while the resulting actions and execution feedback inform the subsequent state update. To explain how the execution monitor and action expert interact, we divide the description into the following three parts.

3.3.1 Execution-state monitoring

The execution monitor is a lightweight Transformer that maintains a fixed-size recurrent memory U_t, including a dedicated state token. For $t > 0$, it receives the observation o_t resulting from the previously executed action prefix A_{t-1}^{exec}, together with the action-model feedback $\mathbf{f}_{t-1}$ produced during that action query. It also retrieves the visual-language context c_{t-1}^{vl}, consisting of the VLM prefix attention keys and values cached when A_{t-1} was generated. We encode the executed transition as

$$z_{t-1} = \text{Enc}(A_{t-1}^{\text{exec}}, o_t, \mathbf{f}_{t-1}). \tag{2}$$

The monitor updates its memory by jointly processing the transition tokens and attending to the cached visual-language context:

$$U_t = \mathcal{T}_\phi(U_{t-1}, z_{t-1}; c_{t-1}^{\text{vl}}), \qquad h_t = \text{LN}(U_t^{\text{state}}). \tag{3}$$

Here, $\mathcal{T}_\phi$ is the Transformer memory updater with parameters ϕ. The recurrent memory preserves execution evidence across decision rounds, while its state token provides h_t to condition the current action query. At the beginning of each episode, U_0 is initialized with learned tokens and $h_0 = \text{LN}(U_0^{\text{state}})$, so the first action query requires no prior execution feedback.

The execution state is further decoded into three monitoring signals:

$$(\hat{d}_t, \hat{m}_t, \hat{p}_t) = \sigma(C(h_t)), \tag{4}$$

where the *DONE* signal $\hat{d}_t$ estimates completion of the overall task; the *MATCH* signal $\hat{m}_t$ measures the consistency of the observed transition with the subtask under execution; and the *PROGRESS*

signal $\hat{p}_t$ measures progress within that subtask. Here, C denotes the prediction heads and σ is the element-wise sigmoid function. These signals share the same execution state while retaining distinct semantic roles.

These signals guide plan-level decisions. While unfinished subtasks remain, persistently low match $\hat{m}_t < \tau_m$ triggers replanning; otherwise, persistently high progress $\hat{p}_t \geq \tau_p$ advances the active subtask. Once the plan is exhausted, global completion $\hat{d}_t$ determines whether to invoke reasoning: $\hat{d}_t < \tau_d$ requests a new plan, while a higher estimate preserves the current execution condition. Here, τ_m, τ_p, and τ_d are the respective thresholds. When replanning is triggered, the vision-language backbone conditions on the current observation, task instruction, and completed prefix $\mathcal{C}$ to autoregressively generate a revised subtask suffix. The resulting p_j conditions action generation in Equation 1, while h_t additionally modulates the action expert through the adapter routing described next.

3.3.2 State-conditioned action generation

Beyond guiding plan-level decisions, the execution state h_t conditions action generation through sparse adapter routing. Specifically, for each action-expert layer, a router maps h_t to a categorical distribution over K bottleneck adapters:

$$g_t = \mathrm{softmax}(R(h_t)), \qquad \tilde{x}_t = x_t + \alpha \sum_{k=1}^{K} \tilde{g}_{t,k} \mathcal{A}_k(x_t). \tag{5}$$

where R is the router, x_t denotes the action-expert hidden features, and $\mathcal{A}_k$ a bottleneck adapter (Houlsby et al., 2019). $\tilde{g}_t$ retains and renormalizes only the two largest weights, and α scales the residual. Following sparse gating (Lepikhin et al., 2020), routing and adapter parameters are layer-specific, with routing conditioned on execution state rather than individual language tokens. This residual pathway allows the action generator to adapt to the current execution stage.

3.3.3 Action-to-monitor feedback

The action expert produces a compact feedback vector $\mathbf{f}_t$ from two types of internal signals: adapter routing weights and weighted residual magnitudes. At each action-expert layer, the latter is computed for adapter k as:

$$\delta_{t,k} = \left\| \alpha \tilde{g}_{t,k} \mathcal{A}_k(x_t) \right\|_2 . \tag{6}$$

The other signal is the adapter routing weight. For each adapter, we compute the mean and standard deviation of its routing weight and the fraction of layers where it receives the largest weight. We additionally include the mean routing entropy and the mean, maximum, and sum of the weighted residual norms across layers and adapters. These statistics are concatenated into a compact feedback vector $\mathbf{f}_t$, summarizing the selected adaptation modes and feature-update magnitudes without exposing full layer-wise activations.

The feedback is paired with the generated action chunk A_t and incorporated into the monitor only after A_t^{exec} is executed and o_{t+1} becomes available. Thus, h_t conditions the action generation, while the feedback and execution outcome contribute to h_{t+1}, closing the monitoring–control loop.

Algorithm 1 summarizes how these components interact during online execution, including state updates, action generation, subtask transitions, and replanning.

3.4 Supervision Construction and Training

We train the monitor on contiguous transition sequences within each episode, allowing its recurrent state to persist across subtask boundaries while changing the active-subtask input. Let y_t^d, y_t^m, y_t^p denote global-completion, match, and subtask-progress labels. We first describe supervision construction and then present the training objectives.

For each successful demonstration, we combine keyframes selected from gripper and motion changes with uniform samples. Given the task instruction, initial observation, and ordered keyframes, a VLM proposes a subtask list and assigns keyframes to its subtasks to determine segment boundaries. We enforce monotonic order and full trajectory coverage, revising and realigning missing, repeated, or unsupported steps. We then spot-check the annotations and manually correct

Algorithm 1 Online execution of EM-VLA

Initialize the cached plan $\mathcal{P} \leftarrow \emptyset$, completed prefix $\mathcal{C} \leftarrow ()$, and monitor state h_0.
for $t = 0, 1, \ldots$ **do**
  if no plan is cached or replanning was requested **then**
    Generate a subtask suffix from $(o_t, \ell, \mathcal{C})$ and cache $\mathcal{P} \leftarrow \mathcal{C} \,\|\, \text{suffix}$.
    Set j to the first uncompleted subtask.
  end if
  Generate action chunk A_t and feedback $\mathbf{f}_t$ from (o_t, ℓ, p_j, h_t).
  Execute A_t^{exec} and observe o_{t+1}.
  Update h_{t+1} from the observed transition, $\mathbf{f}_t$, and visual-language context.
  Predict $(\hat{d}_{t+1}, \hat{m}_{t+1}, \hat{p}_{t+1})$ from h_{t+1}.
  if no uncompleted subtask remains **then**
    if $\hat{d}_{t+1}$ is low **then**
      Request replanning.
    end if
  else if $\hat{m}_{t+1}$ is persistently low **then**
    Request replanning without updating $\mathcal{C}$.
  else if $\hat{p}_{t+1}$ is persistently high **then**
    Append p_j to $\mathcal{C}$; advance j or mark the plan exhausted.
  end if
end for

trajectories that remain inconsistent. The resulting list supervises high-level reasoning, while the aligned segments identify active subtasks for action training and provide monitor labels. For a subtask segment $[b, e]$, the progress label is $y_t^p = (f_t - b)/(e - b)$, where f_t is the frame index at decision round t. This label also supervises routing for the current action query. The global-completion label y_t^d is 1 within the final W frames of a successful trajectory and 0 otherwise. The match label y_t^m is 1 for a transition paired with its annotated subtask and 0 for counterfactual pairs formed using an incorrect subtask or a transition from another subtask segment or task. Further details on annotation and label construction are provided in Appendix B.

The vision-language backbone learns intermediate reasoning by autoregressively predicting the annotated subtask plan. Given multimodal input c and target plan tokens $s_{1:L}$, the reasoning objective is

$$\mathcal{L}_{\text{reason}} = -\mathbb{E}\left[\sum_{i=1}^{L} \log P_\theta(s_i \mid s_{<i}, c)\right]. \tag{7}$$

For execution monitoring, we use binary cross-entropy for completion and match ($\mathcal{L}_{\text{done}}$ and $\mathcal{L}_{\text{match}}$), and smooth-$\ell_1$ loss for progress ($\mathcal{L}_{\text{prog}}$):

$$\mathcal{L}_{\text{mon}} = \mathcal{L}_{\text{done}} + \lambda_m \mathcal{L}_{\text{match}} + \lambda_p \mathcal{L}_{\text{prog}}. \tag{8}$$

Counterfactual match examples contribute only to $\mathcal{L}_{\text{match}}$, because their substituted subtask or transition does not define a valid done or progress target.

The action expert retains its conditional flow-matching objective $\mathcal{L}_{\text{FM}}$ (Black et al., 2025). To encourage stage-dependent adapter use while avoiding route collapse, we additionally supervise the dense routing weights with a progress-conditioned target and a balance regularizer:

$$\mathcal{L}_{\text{stage}} = -\mathbb{E}_t \sum_{k=1}^{K} \pi_{t,k}^{\text{stage}} \log \bar{g}_{t,k}, \qquad \mathcal{L}_{\text{bal}} = \sum_{k=1}^{K} \mu_k \log(K \mu_k). \tag{9}$$

Here, $\pi_{t,k}^{\text{stage}}$ is a Gaussian-smoothed target derived from y_t^p over uniformly spaced progress centers. $\bar{g}_{t,k}$ averages dense routes across layers, and μ_k further averages them over the batch. The balance term encourages uniform aggregate adapter use. Both losses use dense routes, while action generation uses top-2 routes. The resulting action objective is

$$\mathcal{L}_{\text{act}} = \mathcal{L}_{\text{FM}} + \lambda_s \mathcal{L}_{\text{stage}} + \lambda_b \mathcal{L}_{\text{bal}}. \tag{10}$$

Further details are provided in Appendix C.

4 EXPERIMENTS

We evaluate whether EM-VLA improves long-horizon manipulation across simulation and real-world settings, and examine how execution state monitoring and its coupling to action generation affect performance. Beyond task success, we analyze the monitor's predictions, stage-dependent action generation, and inference cost.

Table 1: Success rate (%) on LIBERO. Long denotes LIBERO-10.

Method	Spatial	Goal	Object	Long	Avg.
OpenVLA (Kim et al., 2024)	84.7	79.2	88.4	53.7	76.5
CoT-VLA (Zhao et al., 2025)	87.5	87.6	91.6	69.0	83.9
ThinkAct (Huang et al., 2025)	88.3	87.1	91.4	70.9	84.4
FPC-VLA (Yang et al., 2025)	87.0	86.2	92.0	82.2	86.9
π_0 (Black et al., 2025)	96.8	**98.8**	95.8	85.2	94.2
UniVLA (Bu et al., 2025)	96.5	95.6	96.8	92.0	95.2
CycleVLA (Ma et al., 2026)	97.6	91.7	98.1	93.6	95.3
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	**98.8**	98.2	98.0	92.4	96.8
StarVLA (StarVLA Community, 2026)	97.8	96.2	98.6	93.8	96.6
OpenVLA-OFT (Kim et al., 2025)	97.6	97.9	98.4	94.5	97.1
EM-VLA	96.4	98.6	**98.8**	**96.6**	**97.6**

4.1 EXPERIMENTAL SETUP

Simulation benchmarks. We evaluate on three simulation benchmarks spanning task diversity, sequential manipulation, and bimanual robustness. LIBERO (Liu et al., 2023) comprises the Spatial, Object, Goal, and Long (LIBERO-10) suites, each with ten tasks; we report mean success for each suite. On CALVIN (Mees et al., 2022), we follow the ABC→D protocol, training on environments A, B, and C, and evaluating in unseen D over 1,000 five-task sequences. We report success rates for completing at least 1–5 tasks and the average completed sequence length. RoboTwin 2.0 (Chen et al., 2025) contains 50 bimanual tasks. We report task-averaged success over 100 rollouts per task in its Clean and Randomized settings.

Real-world setup. Our real-world evaluation uses a Piper platform with multi-view RGB observations. We consider four multi-step tasks: **Dish Stacking**, **Shape Placement**, **Bowl Stacking**, and **Fruit Plating**. These tasks require grounding object attributes and target locations, following instruction-specified ordering, and maintaining progress across successive transfers. We evaluate each task in 10 standard trials and three additional robustness settings. **Object Perturbation** substitutes manipulation objects, while **Background Perturbation** changes the backdrop and tabletop covering; both are applied before execution. **Execution Perturbation** applies two interventions per rollout: interrupting the active step in the early-to-middle stage and invalidating a completed step in the middle-to-late stage. We report task success and normalized completion length.

Baselines. We compare 15 baselines grouped by their primary emphasis: **(i) General-purpose VLAs**, including OpenVLA, OpenVLA-OFT, π_0, $\pi_{0.5}$, UniVLA, StarVLA, X-VLA, and LingBot-VLA; **(ii) Reasoning- or prediction-augmented methods**, including CoT-VLA, ThinkAct, UP-VLA, VPP, and Motus; and **(iii) Feedback-driven methods**, including FPC-VLA and CycleVLA.

4.2 SIMULATION BENCHMARK RESULTS

LIBERO. Table 1 shows EM-VLA achieves the highest average success rate of 97.6%, with its most pronounced advantage on LIBERO-Long, where it reaches 96.6%. Compared with reasoning-augmented baselines such as CoT-VLA and ThinkAct, EM-VLA demonstrates stronger long-horizon task execution, consistent with the motivation for maintaining a persistent execution state across interactions.

CALVIN. Table 2 shows that EM-VLA achieves the highest average completed sequence length of 4.35 and completes all five tasks in 75.5% of sequences. It outperforms strong predictive and

Table 2: CALVIN ABC→D results.

Method	1	2	3	4	5	Avg. Len.
OpenVLA (Kim et al., 2024)	91.3	77.8	62.0	52.1	43.5	3.27
π_0 (Black et al., 2025)	93.8	85.0	76.7	68.1	59.9	3.84
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	93.0	85.7	79.2	73.9	68.0	4.00
UP-VLA (Zhang et al., 2025)	92.8	86.5	81.5	76.9	69.9	4.08
VPP (Hu et al., 2025)	95.7	91.2	86.3	81.0	75.0	4.29
UniVLA (Bu et al., 2025)	95.5	85.8	75.4	66.9	56.5	3.80
StarVLA (StarVLA Community, 2026)	92.5	83.9	74.4	67.9	59.9	3.79
X-VLA (Zheng et al., 2025)	95.7	90.9	86.0	78.8	70.5	4.22
EM-VLA	**98.3**	**92.4**	**86.8**	**81.7**	**75.5**	**4.35**

Table 3: Average task success rate (%) on the 50-task RoboTwin 2.0 benchmark.

Method	Clean	Randomized
π_0 (Black et al., 2025)	65.9	58.4
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	82.7	76.8
X-VLA (Zheng et al., 2025)	72.9	72.8
StarVLA (StarVLA Community, 2026)	88.2	88.3
Motus (Bi et al., 2025)	88.7	87.0
LingBot-VLA (Wu et al., 2026)	88.6	86.7
EM-VLA	**91.6**	**90.9**

generalist baselines, including UP-VLA, X-VLA, and VPP, in average sequence length and attains the highest completion rates from three through five tasks.

RoboTwin 2.0. The Clean setting uses scenes without domain randomization, whereas Randomized introduces clutter and variations in lighting, background textures, and tabletop height. Table 3 shows that EM-VLA achieves 91.6% success on Clean and 90.9% on Randomized. It outperforms the most robust prior results from Motus on Clean and StarVLA on Randomized by 2.9 and 2.6 points, respectively. The small gap between the two settings demonstrates strong robustness across diverse bimanual tasks and randomized scene conditions.

4.3 Real-World Experiments

Table 4: Real-world results (SR / Len.): success rate (%) / normalized completion length. Perturbation results are averaged over four tasks.

	Standard Tasks					Perturbations		
Method	Dish	Shape	Bowl	Fruit	Avg.	Object	Background	Execution
OpenVLA	40 / 0.63	30 / 0.40	10 / 0.35	0 / 0.22	20 / 0.40	7.5 / 0.24	17.5 / 0.37	5 / 0.30
π_0	80 / 0.85	**80 / 0.85**	50 / 0.67	40 / 0.60	62.5 / 0.74	37.5 / 0.55	60 / 0.71	32.5 / 0.60
StarVLA	80 / 0.90	70 / 0.70	50 / 0.73	30 / 0.53	57.5 / 0.72	40 / 0.60	52.5 / 0.68	30 / 0.57
EM-VLA	**100 / 1.00**	**80** / 0.80	**70 / 0.88**	**70 / 0.82**	**80 / 0.88**	**55 / 0.72**	**75 / 0.84**	**62.5 / 0.81**

To assess performance beyond simulation, we compare methods on four physical-robot tasks requiring multiple ordered manipulations. Table 4 shows that EM-VLA achieves the highest average success rate of 80.0% and normalized completion length of 0.875, outperforming the strongest baseline by 17.5 percentage points and 0.133, respectively. It completes all Dish Stacking trials and shows particularly clear gains on Bowl Stacking and Fruit Plating, where success requires maintaining progress across multiple ordered manipulations.

Table 4 further evaluates robustness to object substitutions, background changes, and execution-time perturbations. EM-VLA achieves the highest success rate and completion length in all three settings. The largest advantage appears under execution perturbations, where EM-VLA outper-

forms the strongest baseline by 30.0 points while retaining a completion length of 0.806. This result highlights the importance of runtime feedback when the task state changes during execution.

4.4 Ablations and Execution-State Analysis

Table 5: Ablations on CALVIN ABC→D.

Method	1	2	3	4	5	Avg. Len.
Base VLA ($\pi_{0.5}$)	93.0	85.7	79.2	73.9	68.0	4.00
Full-plan execution	94.4	87.5	80.1	74.6	66.8	4.03
VLA-predicted progress	96.0	90.3	82.7	75.9	68.0	4.13
Decoupled execution monitoring	97.2	91.5	84.4	79.3	71.7	4.24
EM-VLA	**98.3**	**92.4**	**86.8**	**81.7**	**75.5**	**4.35**

Component ablation. To assess the effects of execution design and monitoring, we compare several variants on CALVIN (Table 5). The full-plan baseline conditions each action chunk on the complete subtask sequence instead of an active subtask. Its limited improvement suggests that global planning context alone does not resolve the current execution stage or transition timing. The VLA-progress variant predicts subtask progress alongside actions to guide subtask transitions, raising the average sequence length to 4.13. However, progress estimation and action generation involve distinct learning objectives, making their optimization within the same output space potentially suboptimal. Introducing the execution-state monitor raises the average length to 4.24 even without monitor–action coupling. The complete method further reaches 4.35 and a five-task success rate of 75.5%, showing that persistent monitoring and its coupling to action generation provide complementary benefits.

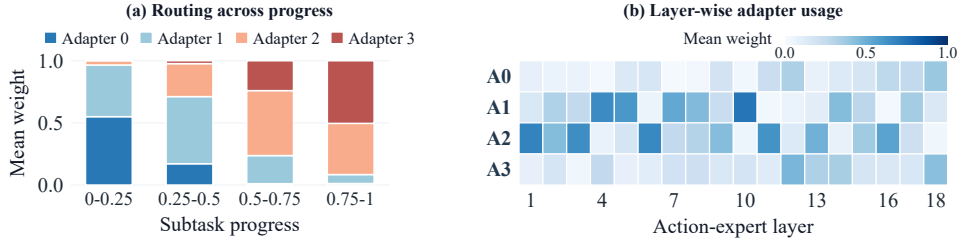


Figure 3: Adapter routing analysis. **Left:** mean top-2 routing weights within subtask-progress bins. **Right:** mean routing weights at each action-expert layer.

Adapter routing analysis. Figure 3 examines whether the execution-state-conditioned router learns distinct action-generation patterns across subtask progress. Routing shifts from Adapters 0 and 1 early in execution, through Adapters 1 and 2, to Adapters 2 and 3 near completion. Overlap between adjacent progress bins suggests gradual rather than rigid stage transitions. The layer-wise heatmap shows that all four adapters remain active across the network, with no global collapse to a single adapter.

5 Conclusion

We presented EM-VLA, a hierarchical VLA that coordinates high-level reasoning and low-level control through a persistent execution state. A lightweight monitor maintains this state to guide subtask transitions, replanning, and stage-dependent action generation, while action-model feedback closes the loop between monitoring and control. Supervision derived from successful demonstrations supports both reasoning and monitoring. Evaluations across simulation and real-world tasks demonstrate improved long-horizon execution and recovery from execution-time perturbations.

AI USE STATEMENT

In this work, we used generative AI tools to assist in generating supervision datasets. Additionally, we used these tools to polish the manuscript. The authors reviewed the AI-assisted work and made corrections where needed. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Christopher Agia, Rohan Sinha, Jingyun Yang, Zi-ang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 689–723. PMLR, 2025. URL https://proceedings.mlr.press/v270/agia25a.html.

Michael Ahn 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.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quan Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024. URL https://arxiv.org/abs/2403.01823.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A unified latent action world model. *arXiv preprint arXiv:2512.13030*, 2025. URL https://arxiv.org/abs/2512.13030.

Kevin Black et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, 2025. URL https://arxiv.org/abs/2410.24164.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Learning to act anywhere with task-centric latent actions. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.014. URL https://www.roboticsproceedings.org/rss21/p014.html.

Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, Hanbo Zhang, and Minzhao Zhu. GR-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024. URL https://arxiv.org/abs/2410.06158.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. RACER: Rich language-guided failure recovery policies for imitation learning. *arXiv preprint arXiv:2409.14674*, 2024. URL https://arxiv.org/abs/2409.14674.

Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In *Proceedings of the 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pp. 120–136. PMLR, 2023. URL https://proceedings.mlr.press/v232/du23b.html.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. *arXiv preprint arXiv:2410.00371*, 2024. URL https://arxiv.org/abs/2410.00371.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. SAFE: Multitask failure detection for vision-language-action models. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2506.09937.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2412.14803.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. ThinkAct: Vision-language-action reasoning via reinforced visual latent planning. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2507.16815.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

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.

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

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020. URL https://arxiv.org/abs/2006.16668.

Fanqi Lin, Ruiqian Nai, Yingdong Hu, Jiacheng You, Junming Zhao, and Yang Gao. OneTwoVLA: A unified vision-language-action model with adaptive reasoning. *arXiv preprint arXiv:2505.11917*, 2025. URL https://arxiv.org/abs/2505.11917.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Chenyang Ma, Guangyu Yang, Kai Lu, Shitong Xu, Bill Byrne, Niki Trigoni, and Andrew Markham. CycleVLA: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding. *arXiv preprint arXiv:2601.02295*, 2026. URL https://arxiv.org/abs/2601.02295v1.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23301–23320. PMLR, 2023. URL https://proceedings.mlr.press/v202/ma23b.html.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. URL https://arxiv.org/abs/2112.03227.

Physical Intelligence, Kevin Black, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8748–8763, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025. URL https://arxiv.org/abs/2502.19417.

StarVLA Community. StarVLA: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026. URL https://arxiv.org/abs/2604.05014.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, Shaoxuan Xie, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025. URL https://arxiv.org/abs/2512.23703.

Hongtao Wu, Ya Jing, Chilam Cheang, Guangzeng Chen, Jiafeng Xu, Xinghang Li, Minghuan Liu, Hang Li, and Tao Kong. Unleashing large-scale video generative pre-training for visual robot manipulation. *arXiv preprint arXiv:2312.13139*, 2023. URL https://arxiv.org/abs/2312.13139.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026. URL https://arxiv.org/abs/2601.18692.

Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. In *Robotics: Science and Systems*, 2025. URL https://arxiv.org/abs/2503.08558.

Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Piaopiao Jin, Guokang Sun, Shaoqing Xu, Yangwei You, and Jingtai Liu. FPC-VLA: A vision-language-action framework with a supervisor for failure prediction and correction. *arXiv preprint arXiv:2509.04018*, 2025. URL https://arxiv.org/abs/2509.04018.

Michal Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024. URL https://arxiv.org/abs/2407.08693.

Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025. URL https://arxiv.org/abs/2509.15937.

Jianke Zhang, Yanjiang Guo, Yucheng Hu, Xiaoyu Chen, Xiang Zhu, and Jianyu Chen. UP-VLA: A unified understanding and prediction model for embodied agent. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2501.18867.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Tsung-Yi Lin, Gordon Wetzstein, Ming-Yu Liu, and Donglai Xiang. CoT-VLA: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1702–1713, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zhao_CoT-VLA_Visual_Chain-of-Thought_Reasoning_for_Vision-Language-Action_Models_CVPR_2025_paper.html.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025. URL https://arxiv.org/abs/2510.10274.

Brianna Zitkovich et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, 2023. URL https://arxiv.org/abs/2307.15818.

A Model Details

Following π_0 and $\pi_{0.5}$ (Black et al., 2025; Physical Intelligence et al., 2025), the VLA pairs a PaliGemma-based vision-language backbone with a separately parameterized action expert. We add the execution monitor as a third Transformer stack. All three use 18 layers, 8 attention heads, one key-value head, and a head dimension of 256. Their hidden and feed-forward widths are $(2048, 16384)$ for the VLM, $(1024, 4096)$ for the action expert, and $(512, 2048)$ for the monitor, respectively.

The visual encoding of o_t is shared with the current VLM prefix computation, allowing that pass and the monitor update to overlap before action decoding. Memory persists across subtask changes but resets to learned initial tokens at each episode boundary; the saved context is cleared at the same time.

B Subtask Annotation and Supervision

B.1 Subtask Annotation

For each successful demonstration, we construct an ordered subtask list and a temporal partition of the trajectory. We first select candidate keyframes around gripper opening and closing and at robot-motion onsets and pauses. We estimate motion from smoothed end-effector displacement between consecutive observations. Onsets and pauses are marked by sustained transitions across separate moving and stationary thresholds; normalized joint-state changes are used when end-effector poses are unavailable. Uniformly spaced frames provide coverage when these events are sparse, and temporally adjacent candidates are merged to avoid redundant images. This produces a compact, ordered visual record while retaining frames near likely changes in manipulation stage.

Annotation uses two VLM passes. Given the task instruction, initial observation, and ordered keyframes, the first proposes a subtask sequence $\mathcal{P} = (p_1, \ldots, p_M)$; the second assigns each keyframe to a subtask. We convert these assignments into contiguous segments $[b_j, e_j]$, placing each boundary at the midpoint between adjacent keyframes assigned to successive subtasks. We enforce monotonic subtask order, full trajectory coverage, and a nonempty interval for every retained subtask. Missing, redundant, or unsupported steps are revised and realigned, merging or splitting subtasks when needed. A single-subtask demonstration receives the full trajectory as its segment. We inspect a sample and manually correct trajectories that remain inconsistent.

VLM prompt templates. The complete instructions for the two VLM passes are shown below. Angle-bracketed fields denote the task text or attached images. The first prompt generates the subtask list:

Prompt 1: Subtask Plan Generation

```
You are a robotic manipulation planning expert.

You are given:
- The FIRST FRAME of a robot manipulation episode
- ORDERED KEYFRAMES from that episode
- A GLOBAL TASK instruction

Your goal is to decompose the task into a sequence of valid
robotic subtasks based on the task instruction and images.

---
ALLOWED ACTIONS AND DEFINITIONS

You MUST use ONLY the following verbs:

1. Pick
- The robot moves toward a target object and establishes
  control of it.
```

Prompt 1: Subtask Plan Generation (continued)

```
- This includes both reaching the object and grasping it.
- Use Pick when the goal is to acquire or hold an object.

2. Place
- The robot transports a held object to a target location
  and releases it there.
- This includes both moving the object and placing it.
- MUST include a destination.

3. Push
- The robot applies force to an object without necessarily
  grasping it.
- The object is moved by pushing, pressing, or sliding.
- Use Push when the robot interacts with an object mainly
  through contact force rather than grasping.
- MUST include a destination.

4. Open
- The robot manipulates an articulated object to open it,
  such as pulling a drawer or opening a cabinet door.

5. Close
- The robot manipulates an articulated object to close it,
  such as pushing a drawer or closing a cabinet door.

6. Pour
- The robot tilts a held container to empty its contents.
- MUST include a destination where the contents are poured.

7. Press
- The robot presses, taps, or clicks a button, switch,
  or control.
- MUST include the target being pressed.

8. Strike
- The robot uses a held object or tool to hit a target.
- MUST include the tool and the target.

9. Shake
- The robot repeatedly moves a held object back and forth.
- MUST include the object being shaken.

10. Rotate
- The robot changes the orientation of an object.
- MUST preserve any orientation requirement from the task.

11. Handover
- The robot transfers an object from one arm to the other.
- MUST specify the source arm and destination arm.

12. Scan
- The robot uses a scanner or scanning device to scan
  a target object.
- MUST include the scanner and the target object.

---
STRICT CONSTRAINTS

1. Do NOT hallucinate objects or destinations.
2. The sequence must be logically executable.
3. Each step must follow physical feasibility.
4. Avoid redundant actions, but retain repeated operations
```

Prompt 1: Subtask Plan Generation (continued)

```
   when required by the task and supported by the episode.
5. Preserve object names and attributes specified in the
   task. When objects are not individually named, use
   visible attributes to distinguish them consistently;
   do not invent identities or properties.
6. Preserve explicitly specified left-arm, right-arm,
   or both-arm requirements.
7. Preserve important orientation and spatial constraints.
8. Use one subtask per distinct manipulation stage. Pick
   includes reaching and grasping; Place includes transport
   and release. Do NOT split routine reaching or transit
   into separate subtasks.
9. Keep independent object transfers as separate steps in
   execution order; do NOT combine multiple pick/place
   cycles into a single subtask.

---
DESTINATION RULE

For the following actions:
- Place
- Push
- Pour
- Press
- Strike
- Handover
- Scan

You MUST specify the destination, target, or receiving arm
explicitly.

---
OUTPUT FORMAT (STRICT JSON ONLY)

Return ONLY valid JSON:
{
  "subtasks": [
    "Pick object",
    "Place object on destination",
    "Pick tool with the right arm",
    "Strike target with tool"
  ]
}

No explanations. No extra text.

INPUTS
Global Task: <task instruction>
First Frame: <image>
Ordered Keyframes:
frame_index=<index>: <image>
...
```

The second prompt assigns the ordered keyframes to the fixed subtask list:

Prompt 2: Trajectory Alignment

```
You are a robotic manipulation trajectory annotation expert.

You are given:
- a global task instruction
- an ordered list of subtasks with fixed IDs
```

Prompt 2: Trajectory Alignment (continued)

```
- multiple ordered keyframes from one robot manipulation
  episode

Your job is to assign each keyframe to exactly one subtask_id.

Rules:
1. You MUST choose subtask_id only from the provided
   subtask list.
2. Do NOT create new subtasks.
3. Do NOT rewrite subtask names.
4. The assigned subtask_id sequence should be temporally
   monotonic non-decreasing.
5. If a frame is ambiguous, choose the most likely current
   subtask based on the robot state, object state, gripper
   state, and task progress.
6. At a boundary, switch to the next subtask at the first
   keyframe that visibly shows its execution underway.
   Do not switch solely because the previous step appears
   complete.
7. If no keyframe supports a proposed subtask, do not
   force an assignment to its ID; leave it for validation.
8. Return strict JSON only, with one entry per supplied
   keyframe in chronological order. Preserve the supplied
   frame indices; do not omit, duplicate, or invent frames.
9. Do not output confidence.
10. Do not output explanations.

Output format:
{
  "frame_labels": [
    {"frame_index": 0, "subtask_id": 0},
    {"frame_index": 10, "subtask_id": 0},
    {"frame_index": 20, "subtask_id": 1}
  ]
}

INPUTS
Global Task: <task instruction>
Subtasks:
0: <subtask 0>
1: <subtask 1>
...
Keyframes (total <n>):
frame_index=<index>: <image>
...
```

B.2 Training Labels and Counterfactual Pairs

Plans and aligned segments provide the supervision described in Section 3.4. For matching states, the annotated task–subtask–transition pairing is positive. We sample three types of negatives: a different subtask from the same episode with the original transition; a transition from a different subtask of the same task, possibly in another episode; or a transition from another task. The latter two retain the anchor's task and subtask, and identical subtask texts are excluded when selecting a different subtask. By default, we sample one negative of each type when available. Each sampled transition keeps its pre-observation, executed action prefix, and post-observation together. Counterfactual pairs supervise match only; done and progress losses are masked. Their visual-language prefix context is computed from the transition's pre-observation and the resulting task–subtask pair rather than copied from the source pair.

C Training Details

Table 6: Training and data configurations. S1/S2/S3 denotes the three VLA training stages. Merged entries indicate shared settings across benchmarks.

Category	Hyperparameter	LIBERO	CALVIN	RoboTwin	Real-world
Optimization	Optimizer	AdamW			
	Optimizer betas	$(0.9, 0.95)$			
	Learning rate	5×10^{-5}			
	Gradient clipping	1.0			
	LR scheduler	Cosine			
	Warmup ratio	0.1	0.1	0.1	0.1
Training	Steps (S1/S2/S3)	20k/10k/5k	40k/20k/10k	100k/50k/10k	20k/10k/5k
	Batch size (S1/S2/S3)	64 / 64 / 128			
	Action horizon	10	10	10	20
	Monitor steps	30k	30k	30k	30k
	Monitor batch size	512 (sequence) / 1,024 (match)			
Loss weights	Reasoning	0.2			
	Stage routing λ_s	0.05			
	Balance λ_b	0.005			
Data	Demonstrations	1,693	18,957	27,500	400
	Frames	273,465	1,136,536	6,075,103	230,261
	Cameras	2	2	3	3
	Robot action dimension	7	7	14	7

Training proceeds through three stages. Stage 1 jointly optimizes the vision-language backbone and action expert for intermediate reasoning and action generation. Stage 2 freezes the backbone and specializes the action expert for current-subtask control, followed by training an initial execution monitor without action-expert feedback. Stage 3 freezes both the vision-language backbone and the action expert, trains the execution-state-conditioned router and adapters, and refines feedback-aware monitoring using the resulting action-expert responses. Table 6 summarizes the VLA training and data configurations across benchmarks. Stage-specific training steps and batch sizes are listed in Stage 1/2/3 order.

For routing supervision, we convert the current-frame progress label y_t^p into a smooth target over adapters. For $K > 1$, the stage centers are $c_k = (k-1)/(K-1)$, and the target is

$$\pi_{t,k}^{\text{stage}} = \frac{\exp\big(-(y_t^p - c_k)^2/(2\sigma_s^2)\big)}{\sum_{k'=1}^{K} \exp(-(y_t^p - c_{k'})^2/(2\sigma_s^2))}. \tag{11}$$

We use $K = 4$ adapters. The width $\sigma_s = 0.22$ controls overlap between neighboring stages, allowing routing preferences to change smoothly with progress. This target supervises the layer-averaged dense routing distribution through Equation 9; action generation uses the normalized top-2 weights.

D Evaluation Protocols

Simulation evaluation. On LIBERO, we evaluate each task over 50 rollouts, report success rates for the four suites separately, and average them to obtain the overall score. On CALVIN, evaluation uses 1,000 five-task sequences under the ABC→D protocol. Let $L_i \in \{0, \ldots, 5\}$ denote the number of consecutive tasks completed in sequence i before the first failure. Success at length n is the fraction of sequences with $L_i \geq n$, and average completion length is the mean of L_i. On RoboTwin 2.0, we evaluate each of the 50 tasks over 100 rollouts in each of the Clean and Randomized settings and average the per-task success rates within each setting.

Real-world tasks and scoring. Figure 4 shows the real-world platform. The four tasks in Figure 5 require successive object manipulations under a single instruction. Table 7 lists their instructions and

numbers of scoring steps. Instruction-specified ordering constraints are part of the success criterion. Each method is evaluated over 10 standard trials per task. Task success is the fraction of trials satisfying all requirements; normalized completion length measures partial task completion. We report each task separately and take an unweighted average over the four tasks.

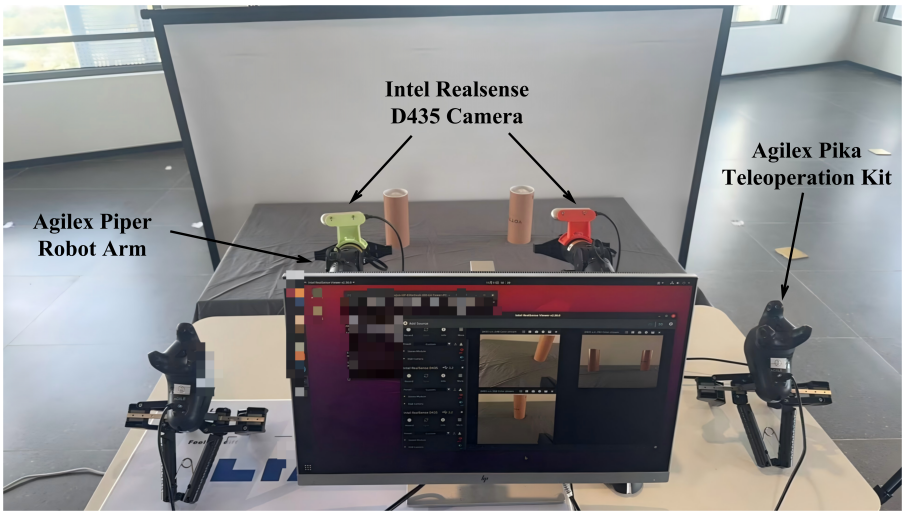
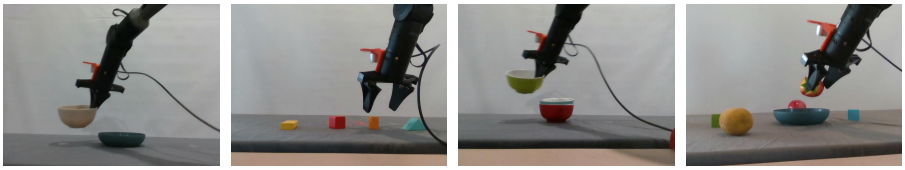


Figure 4: Real-world experimental platform with AgileX Piper robot arms, Intel RealSense D435 cameras, and the AgileX Pika teleoperation kit used for demonstration collection.



(a) Dish Stacking (b) Shape Placement (c) Bowl Stacking (d) Fruit Plating

Figure 5: Four real-world long-horizon tasks.

Table 7: Real-world task instructions and scoring steps.

Task	Instruction	Steps
Dish Stacking	Place the blue plate on the table, then place the bowl on the plate.	4
Shape Placement	Place the [X]-shaped and [Y]-shaped blocks in the red region.	4
Bowl Stacking	Stack bowls in the order of color [X], [Y], [Z] from top to bottom.	6
Fruit Plating	Put all fruits on plate.	6

Each real-world trial has a time limit of 150 seconds. For normalized completion length, grasping and placing are scored as separate steps. The score is the number of steps completed at the end of the trial divided by the total number of required steps for that task. Scoring reflects final completion rather than historical achievement: a step invalidated by an intervention receives no credit unless it is restored, and repeating a step does not earn additional credit.

Perturbation protocols. We evaluate three additional settings, with 10 trials per task and method in each setting. Object Perturbation replaces manipulation objects with new instances before execution. Background Perturbation changes the backdrop and tabletop covering before execution. Execution Perturbation applies two interventions per rollout: the first, during the early-to-middle stage, interferes with the ongoing manipulation; the second, during the middle-to-late stage, undoes a previously completed step. The policy must respond to the changed scene and satisfy the original task instruction within the 150-second trial limit. These settings distinguish changes to object appearance and background from changes to the physical task state during execution.

E ADDITIONAL EXPERIMENTS

Monitor prediction analysis. We evaluate the Transformer execution monitor on LIBERO using an episode-disjoint 80/20 training/test split. Done denotes global task completion, match denotes execution–subtask consistency, and progress measures advancement within the active subtask. We report accuracy and F1 for done and match, and mean absolute error (MAE) and root mean squared error (RMSE) for progress on the held-out test set. Table 8 summarizes the results. Done and match achieve F1 scores of 79.81% and 76.66%, respectively, while progress prediction has an MAE of 0.06326 on the normalized $[0, 1]$ scale. These prediction metrics complement the downstream task evaluations rather than directly measuring recovery success.

Table 8: Monitor prediction results on LIBERO.

Prediction	Metric	Result
Done	Accuracy (%)	92.12
	F1 (%)	79.81
Match	Accuracy (%)	87.95
	F1 (%)	76.66
Progress	MAE	0.06326
	RMSE	0.08108

Table 9: Mean component inference time on one NVIDIA RTX 4090

Component	Time (ms)
Visual embedding	13.8
VLM prefix	37.4
Action decoding	23.8
Execution monitor	2.1
Adapters	0.56
High-level reasoning (optional)	≈ 389

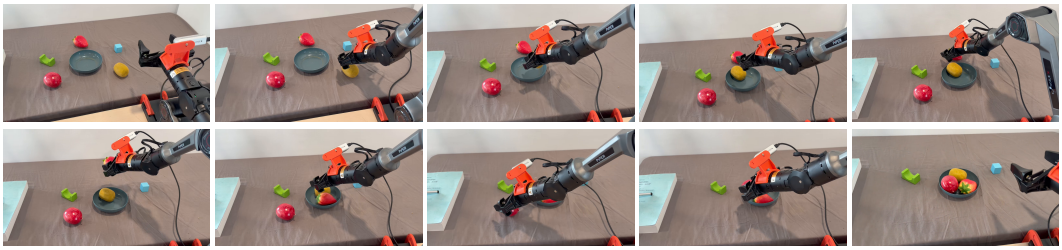
Inference efficiency. We measure component-level inference latency on a single NVIDIA RTX 4090 (Table 9). During routine action queries, the execution monitor and routed adapters take 2.1 and 0.56 ms, respectively, compared with 37.4 ms for VLM prefix computation and 23.8 ms for action decoding. High-level planning takes approximately 389 ms per invocation but is triggered only when a subtask plan needs to be generated or revised. These measurements characterize component-level costs rather than end-to-end latency under overlapping execution.

F Execution Case Studies

Figure 6 presents a continuous real-world Fruit Plating rollout with the instruction "Put all fruits on plate." The robot first transfers the three fruits onto the plate. After the task has been completed, a human removes the yellow fruit from the plate and changes the surrounding object layout. The robot subsequently retrieves the displaced fruit and returns it to the plate, restoring the task goal.

This case illustrates recovery when an externally induced change invalidates a previously achieved goal. After completing the original task, the robot responds to the altered scene and restores the goal by returning the displaced fruit to the plate.

(a) Normal execution



(b) Perturbation and recovery



Figure 6: Normal execution and recovery under perturbation in a continuous Fruit Plating rollout. Frames are ordered chronologically from left to right and top to bottom. The first two rows show successive grasping, transport, and placement of the three fruits. The last row shows the perturbed layout and the robot returning the displaced fruit to the plate.