ECHO: COUPLED ACTION AND TASK STATUS SIGNAL DENOISING FOR VISION LANGUAGE ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recognizing task completion and failure enables robots to stop upon success and initiate recovery or replanning after failure. Existing methods augment Vision Language Action (VLA) policies with task status signal prediction to support these execution decisions. However, learning and integrating these signals can require fine-grained supervision, external monitoring models, or complex architectural designs. Moreover, the potential of task status signals to directly guide action generation remains underexplored. We present Echo, a framework that combines task status signal learning without frame-level progress annotations with a compact design for status prediction and action generation within a single VLA. Specifically, Echo represents task status with stop and failure signals, added as two dimensions of each action chunk and denoised together with robot actions. Their combination distinguishes ongoing execution, successful completion, and failure. To learn these signals without fine-grained progress annotations, Echo constructs successful, failed, and recovery trajectories with complementary supervision properties. To enable status signal-guided action generation, Echo controls interactions between the signals and actions and denoises the signals faster than the actions. These allow later action denoising steps to condition on cleaner status estimates while preserving accurate signal prediction. Extensive real-world manipulation experiments show that Echo accurately predicts task status and consistently improves policy performance.

1 INTRODUCTION

Vision Language Action (VLA) models have shown strong capabilities in robotic manipulation by mapping visual observations and language instructions to robot actions (Intelligence et al., 2025; Bjorck et al., 2025; Team et al., 2026). Reliable execution also requires recognizing whether a task is ongoing, has succeeded, or has failed to determine when to continue, stop, or intervene. Existing methods predict progress, completion, or failure signals either within VLA policies or through external monitoring models (Apanasevich et al., 2026; Feng et al., 2026).

However, existing approaches face two limitations. First, learning and integrating these signals can require fine-grained frame-level supervision, external monitoring models, or complex architectural designs (Pacaud et al., 2025; Feng et al., 2026; Yan et al., 2026; Deng et al., 2026). Second, the predicted signals are typically used only for monitoring, leaving their potential to improve action generation underexplored (Pacaud et al., 2025; Huang et al., 2026).

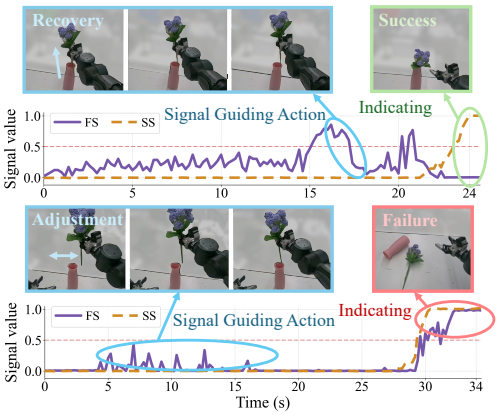

Figure 1: A low stop signal (SS, orange) value indicates ongoing execution, where a high failure signal (FS, purple) value marks recoverable failure and guides correction. High SS with low or high FS values indicates success (top) or terminal failure (bottom) respectively.

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.

We present Echo, a compact framework that jointly denoises robot actions and task status signals within a single VLA, learning the signals without frame-level progress annotations and using them to improve action generation. Echo represents task status through a stop signal (SS), indicating whether execution should terminate, and a failure signal (FS), indicating whether execution is in a failure state. Low SS indicates ongoing execution, while high SS with low FS indicates successful completion, and high SS with high FS indicates terminal failure. Within ongoing execution, elevated FS indicates a recoverable failure, as shown in Figure 1. Echo appends SS and FS as two additional dimensions of each action chunk and denoises them together with robot actions. This formulation reuses the VLA's visual, language, and robot-state features with little additional parameter overhead.

To learn these signals without frame-level progress annotations, Echo combines successful demonstrations, failed policy rollouts, pre-intervention failure segments, and recovery demonstrations using subtask-level prompts together with success or failure labels. Failed rollouts and pre-intervention failure segments supervise terminal and recoverable failures, respectively, while their actions are excluded from imitation. Recovery demonstrations pair similar status supervision with corrective actions at corresponding recoverable failure states, linking failure recognition to recovery behavior.

To use the learned status information during action generation, Echo controls interactions among SS, FS, and robot actions while preserving accurate and complementary signal predictions. It also introduces asynchronous denoising, denoising signal tokens faster than action tokens so that later action denoising steps condition on cleaner status estimates. When the policy recognizes a recoverable failure with low SS and elevated FS, these estimates, together with the current observations, can guide action generation toward corrective behavior learned from recovery demonstrations.

Experiments on five real-world manipulation tasks demonstrate accurate task-status prediction and improved performance over action-only baselines, with larger out-of-distribution gains. Component ablations, signal interventions, and failure-data scaling analyses examine how status signals influence action generation and how different data sources contribute to prediction and execution.

Our contributions are threefold: 1) We introduce a compact task status representation that incorporates stop and failure signals into VLA action chunks with little additional parameter overhead. 2) We develop a supervision scheme that learns these signals from successful, failed, and recovery data without frame-level progress annotations. 3) We design controlled interactions and asynchronous denoising to couple status prediction with action generation.

2 Related Works

Task Status Signal Modeling. Task status signals characterize task progress, completion, and failure (Kim et al., 2026; Liu et al., 2026b; Dai et al., 2026; Liu et al., 2026a; Liufu et al., 2026; Zhao et al., 2026). Visual-language reward models learn execution quality through representation learning, temporal supervision, and trajectory comparisons (Ma et al., 2023; Alakuijala et al., 2024; Ayalew et al., 2025; Zhang et al., 2025; Lee et al., 2026; Liang et al., 2026). When deployed as standalone evaluators, these models introduce additional evaluation alongside the control policy. Complementary approaches predict completion (Yang et al., 2025) or detect failures using policy features, temporal consistency, uncertainty, and visual reasoning (Gu et al., 2026; Park et al., 2026; Agia et al., 2024; Xu et al., 2025; Duan et al., 2024). Progress can also be extracted through lightweight probes (Bhardwaj et al., 2026). Although feature-based monitoring reduces overhead, accurate signal extraction alone does not ensure action feedback. Jointly achieving reliable task status prediction and signal-guided action adjustment with low overhead remains underexplored.

Task Status Signals for Action Improvement. Task status feedback improves policies through several mechanisms. VLAC (Zhai et al., 2025) uses critic tokens to estimate rewards for online reinforcement learning, while GRAPE (Zhang et al., 2024) optimizes policies using trajectory preferences; both require additional policy updates. Other methods use execution assessments to trigger backtracking, intervention, or corrective control (Ma et al., 2026; Deng et al., 2026; Yang et al., 2026; Lin et al., 2025; Li et al., 2026; Liu et al., 2024). More directly, VLA-Corrector (Pan et al., 2026) monitors visual dynamics deviations and applies gradient-guided corrective replanning, while ProgressVLA (Yan et al., 2026) propagates progress gradients through a world model to guide action denoising. These mechanisms demonstrate the utility of execution feedback, but introduce additional policy optimization, monitoring and recovery components, or inference-time gradient

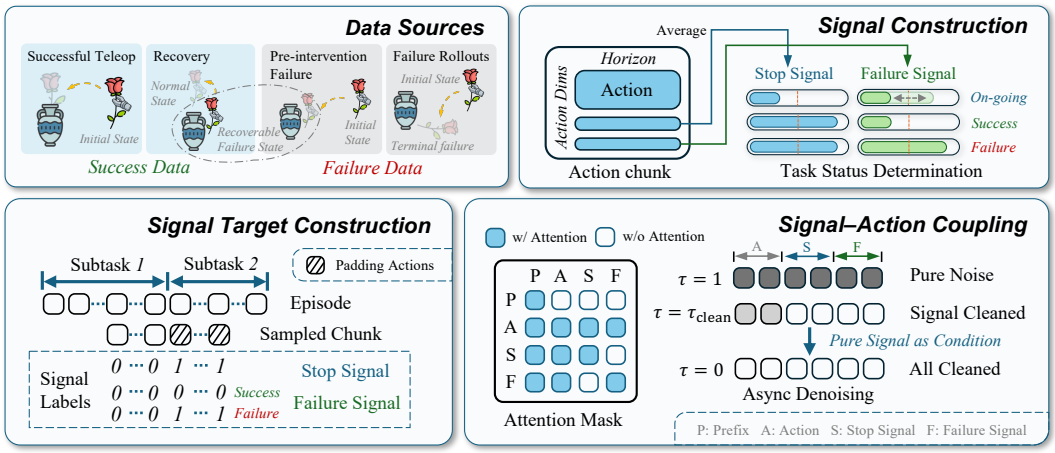

Figure 2: Overview of Echo. Four data types provide complementary action and signal supervision, with failed actions excluded from imitation. Stop (SS) and failure (FS) signals are appended to action chunks, with padding-based target construction illustrated on the left. The attention mask couples actions with both signals while blocking direct SS–FS interactions. Asynchronous denoising provides cleaner signals to guide subsequent action updates.

computation. Preserving accurate status estimates while directly improving action generation with minimal additional training and inference overhead remains an open challenge.

3 Methodology

3.1 Preliminaries

Action chunk representation and padding. An episode contains either a single task or multiple consecutive subtasks. For multi-subtask episodes, we manually annotate subtask boundaries and associate each segment with a language instruction. At episode timestep t, the policy takes the current observation, robot state, and instruction as input and predicts an action chunk $\mathbf{A}_t \in \mathbb{R}^{H \times d_a}$. Here, t is the starting timestep of the chunk, H is its prediction horizon, and d_a is the action dimension. Each action $\mathbf{a}_{t,i} \in \mathbb{R}^{d_a}$ corresponds to timestep $t+i$ and is expressed relative to the robot state at the beginning of the chunk.

A sampled chunk may extend beyond the current segment, entering a subsequent subtask or exceeding the episode length. Let T denote the segment's last valid action timestep. We identify positions beyond this boundary using a padding mask:

$$p_{t,i} = \begin{cases} 0, & t+i \le T, \\ 1, & t+i > T, \end{cases} \qquad i = 0, \ldots, H-1. \tag{1}$$

We replace the corresponding action targets with zeros:

$$\tilde{\mathbf{a}}_{t,i} = (1 - p_{t,i})\mathbf{a}_{t,i}. \tag{2}$$

Besides excluding actions beyond the current segment, padding naturally provides boundary-based signal labels. For example, when a segment ends in task completion, its padding mask indicates where stopping should be predicted. Echo combines this mask with the segment type to construct task status signal targets without additional frame-level progress annotations.

Flow matching. Echo builds on a flow-based VLA policy. Given a clean sample $\mathbf{x}_0$ and Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we construct

$$\mathbf{x}_\tau = (1-\tau)\mathbf{x}_0 + \tau\boldsymbol{\epsilon}, \qquad \tau \in [0,1], \tag{3}$$

where $\tau = 0$ denotes clean data and $\tau = 1$ denotes noise. The model predicts a velocity $\mathbf{v}_\theta(\mathbf{x}_\tau, \tau, \mathbf{c})$, conditioned on the observations, robot state, and instruction, collectively denoted by $\mathbf{c}$. Training matches the predicted velocity to $\boldsymbol{\epsilon} - \mathbf{x}_0$. Inference integrates the predicted velocity from $\tau = 1$ to $\tau = 0$.

Table 1: Action and task-status supervision for four data types. H is the chunk horizon, n counts non-padding positions, and n_t is defined in Eq. 8. $\mathbf{0}_k$ and $\mathbf{1}_k$ denote k zeros and ones. Blue text marks padding positions. "All", "Non-pad", and "–" denote supervision over the full chunk, non-padding positions only, and no supervision, respectively.

Sampled action chunks:		*No padding:* $[\mathbf{a}_1, \ldots, \mathbf{a}_H]$		*With padding:* $[\mathbf{a}_1, \ldots, \mathbf{a}_n, \text{pad}, \ldots, \text{pad}]$	
Data type	**Padding?**	**Action Supervision**	**Stop Signal (SS)**		**Failure Signal (FS)**	
			Target	**Supervision**	**Target**	**Supervision**
Success	No	All	$[\mathbf{0}_H]$	All	$[\mathbf{0}_H]$	All
	Yes	All	$[\mathbf{0}_n, \mathbf{1}_{H-n}]$	All	$[\mathbf{0}_n, \mathbf{0}_{H-n}]$	All
Recovery	No	All	$[\mathbf{0}_H]$	All	$[\mathbf{0}_{n_t}, \mathbf{1}_{H-n_t}]$	All
	Yes	Non-pad	$[\mathbf{0}_n, \mathbf{0}_{H-n}]$	Non-pad	$[\mathbf{0}_{n_t}, \mathbf{1}_{H-n_t}]$	All
Failure	No	–	$[\mathbf{0}_H]$	All	–	–
	Yes	–	$[\mathbf{0}_n, \mathbf{1}_{H-n}]$	All	$[\mathbf{0}_n, \mathbf{1}_{H-n}]$	All
Pre-intervention failure	No	–	$[\mathbf{0}_H]$	All	–	–
	Yes	–	$[\mathbf{0}_n, \mathbf{0}_{H-n}]$	All	$[\mathbf{0}_n, \mathbf{1}_{H-n}]$	All

3.2 Echo Framework

3.2.1 Task-Status Representation

Echo uses two complementary signals as shown in Figure 2: a stop signal (SS), indicating whether execution should terminate, and a failure signal (FS), indicating whether execution is in a failure state. Both signals share the action horizon and are represented as column vectors $\mathbf{S}_t, \mathbf{F}_t \in \mathbb{R}^{H\times 1}$:

$$\mathbf{S}_t = [s_{t,0}, \ldots, s_{t,H-1}]^\top, \qquad \mathbf{F}_t = [f_{t,0}, \ldots, f_{t,H-1}]^\top. \tag{4}$$

Concatenating them with the action matrix along the feature dimension gives the joint target

$$\mathbf{X}_t = [\tilde{\mathbf{A}}_t, \mathbf{S}_t, \mathbf{F}_t] \in \mathbb{R}^{H\times(d_a+2)}. \tag{5}$$

We average each predicted signal within the current chunk to obtain scalar estimates for timestep t:

$$\bar{s}_t = \frac{1}{H}\sum_{i=0}^{H-1} \hat{s}_{t,i}, \qquad \bar{f}_t = \frac{1}{H}\sum_{i=0}^{H-1} \hat{f}_{t,i}. \tag{6}$$

Given thresholds δ_s and δ_f, the execution status is

$$z_t = \begin{cases} \text{Ongoing}, & \bar{s}_t < \delta_s, \\ \text{Success}, & \bar{s}_t \geq \delta_s \wedge \bar{f}_t < \delta_f, \\ \text{Terminal failure}, & \bar{s}_t \geq \delta_s \wedge \bar{f}_t \geq \delta_f. \end{cases} \tag{7}$$

3.2.2 Data Construction and Supervision

We collect successful demonstrations through teleoperation and terminal failure rollouts during policy testing. During policy execution, operators intervene when recoverable failures occur, yielding pre-intervention failure segments before human takeover and recovery demonstrations afterward. Recovery demonstrations may complete a local correction or the entire instructed task. The collected data are aggregated to update the policy for subsequent collection rounds.

We construct action, SS, and FS supervision from segment types, boundaries, and positions within recovery segments, without additional frame-level progress annotations. Table 1 summarizes the targets and supervision regions. For a chunk with n non-padding positions, the padding-mask target is displayed as $[\mathbf{0}_n, \mathbf{1}_{H-n}]$. Blue identifies padding positions, independently of their target values or whether they are supervised.

Action supervision. Successful demonstrations and recovery demonstrations provide task-execution and corrective actions, respectively. We exclude actions from both terminal and pre-intervention failure data to avoid imitating unsuccessful behavior. Successful demonstrations receive action supervision over the entire chunk, including padding. Recovery actions are supervised only at non-padding positions: a recovery segment may end before the instructed task is complete, and learning its padded actions could cause the policy to stop prematurely.

Stop signal supervision. Chunks without padding receive zero SS targets at all positions. For chunks containing padding, the targets depend on whether the segment boundary marks the end of execution. Successful demonstrations end upon completion, while terminal failure rollouts end after an unrecoverable failure. Both therefore use the padding mask as the SS target, with supervision over the entire chunk.

Pre-intervention segments end at operator takeover, and task execution continues through recovery. Their SS targets are therefore zero everywhere, including padding, and are fully supervised. For recovery segments, we also set the full SS target to zero but supervise only non-padding positions. Recovery may end after a local correction or upon full task completion, so its boundary does not consistently indicate whether execution should stop. Excluding padded positions avoids assigning an incorrect termination target.

Failure signal supervision. Successful demonstrations contain successful execution without failure, so FS targets are zero over the entire chunk. For terminal and pre-intervention failure data, the segment-level labels indicate failure but do not identify its onset. We therefore omit FS supervision for chunks without padding. The segment endpoint, however, is known to be in a failure state. For chunks containing padding, we use the padding mask as the FS target and supervise the full chunk, assigning one to padded positions and zero elsewhere. This provides a boundary-based target without requiring the precise failure onset to be annotated.

Recovery demonstrations describe a transition from a recoverable failure state to a state from which successful task execution can resume. We encode this transition using binary chunk targets whose mean decreases with recovery progress. For a recovery segment spanning frames t_0 to t_1, with $t_1 > t_0$, define

$$u_t = \frac{t - t_0}{t_1 - t_0}, \qquad n_t = \lfloor H u_t \rfloor . \tag{8}$$

The FS target is $\mathbf{r}_t = [r_{t,0}, \ldots, r_{t,H-1}]^\top$, where

$$r_{t,i} = \begin{cases} 0, & i < n_t, \\ 1, & i \geq n_t, \end{cases} \qquad i = 0, \ldots, H-1. \tag{9}$$

The first frame's chunk is all ones, and the last frame's chunk is all zeros. Intermediate chunks contain n_t leading zeros followed by $H - n_t$ ones, shown as $[\mathbf{0}_{n_t}, \mathbf{1}_{H-n_t}]$ in Table 1. The mean target, $1 - n_t/H$, decreases approximately linearly with recovery progress. Unlike the padding boundary n, n_t is determined by the starting frame's position within the recovery segment. FS is supervised over the full chunk, including padding.

3.2.3 Status-Guided Action Generation

We next use the learned status signals to guide action generation, including corrective behavior during ongoing execution. To this end, Echo combines controlled signal–action interactions with asynchronous denoising, allowing later action updates to condition on cleaner status estimates.

Joint token processing. We project actions, SS, and FS into separate token groups and concatenate them with the multimodal prefix:

$$\mathbf{H} = [\mathbf{H}^P; \mathbf{H}^A; \mathbf{H}^S; \mathbf{H}^F], \qquad \mathbf{H}^q = \phi_q(\mathbf{x}^q_{\tau_q}), \quad q \in \{A, S, F\}, \tag{10}$$

where $\mathbf{H}^P$ denotes the multimodal prefix tokens carrying the conditioning context, ϕ_q is the input projection for group q, and τ_q is its flow time. The concatenated tokens are processed together by a shared Transformer architecture. Action, SS, and FS tokens use separate MLPs for their feature transformations and separate output projections for velocity prediction.

To support different noise levels within the same sequence, we extend the flow-time input from one value per sample to one value per generated token. For batch size B and concatenated generated-token length L, the time tensor has shape $B \times L$ (Black et al., 2025). Tokens belonging to the same

group receive that group's flow time. The resulting time embeddings are passed to AdaRMS within the shared Transformer.

Controlled signal-action interactions. Action tokens attend to the multimodal prefix and both signal groups, while SS and FS tokens attend to action tokens. These bidirectional interactions allow status estimates to guide action generation and action features to inform status prediction. We block direct SS–FS attention in both directions to reduce cross-signal interference and preserve their distinct termination and failure semantics.

Motivating experiment: earlier status estimates. To examine when status information becomes available, we train a baseline that denoises signals and actions synchronously. At an intermediate flow time, its predicted velocity gives a clean-sample estimate:

$$\hat{\mathbf{x}}_0(\tau) = \mathbf{x}_\tau - \tau \mathbf{v}_\theta(\mathbf{x}_\tau, \tau, \mathbf{c}). \tag{11}$$

As shown in Figure 3(b), both SS and FS approach their final estimates early in denoising. Action estimates, however, continue to change even between steps 8 and 9, the final two plotted steps (Figure 3(a)). These observations motivate resolving signals earlier so that cleaner status estimates can guide the remaining action updates. Details are provided in Appendix A.5.

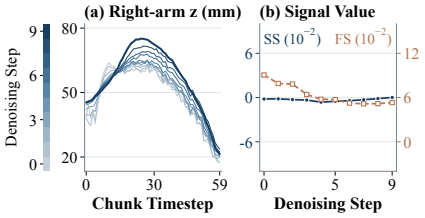

Figure 3: Intermediate clean-sample estimates under synchronous denoising. (a) Right-arm vertical action estimates; darker curves indicate later denoising steps. (b) Stop (SS) and failure (FS) signal estimates, plotted on separate scales. Both signals converge earlier than actions, with SS already near its zero target at the earliest denoising step. This motivates resolving signals earlier to guide subsequent action updates.

Asynchronous denoising. Let τ_A be the action flow time. For signal group $q \in \{S, F\}$, we use

$$\tau_q = \max\left(0, \frac{\tau_A - \tau_{\text{clean}}}{1 - \tau_{\text{clean}}}\right), \qquad 0 \le \tau_{\text{clean}} < 1, \tag{12}$$

where τ_{clean} is the action flow time at which signal q reaches its clean endpoint. Setting $\tau_{\text{clean}} = 0$ gives synchronous denoising. With $\tau_{\text{clean}} > 0$, the signal becomes clean before action denoising is complete.

During training, we sample τ_A, obtain the signal times from Eq. 12, and perturb each group.

$$\mathcal{L} = \mathbb{E}\left[\sum_{q \in \{A,S,F\}} \lambda_q \left\| \mathbf{M}^q \odot (\hat{\mathbf{v}}_\theta^q - (\boldsymbol{\epsilon}^q - \mathbf{x}_0^q)) \right\|_F^2\right]. \tag{13}$$

Here, $\lambda_A = 1$, while λ_S and λ_F weight signal losses for chunks without padding to balance the numbers of padded and unpadded samples; both are set to 1 for chunks containing padding. The binary masks $\mathbf{M}^q$ select the supervised positions in Table 1 and are broadcast over feature dimensions.

At inference, each group starts from Gaussian noise and is updated according to its own flow-time schedule. Once a signal reaches its clean endpoint, its value is held fixed while action denoising continues. Through attention, later action updates can therefore use the cleaner status estimates. The final signal means provide the scalar predictions used for execution decisions.

4 Experiments

We first report main results and comparisons with existing methods to evaluate the accuracy of Echo's task status signals and their benefit to task execution. We then analyze in-distribution (ID) and out-of-distribution (OOD) performance and ablate the model components. Finally, we intervene on the signals to verify their effect on action generation and vary the amount of failure data to examine the role of data sources.

4.1 Experimental Setup

Tasks and environments. We evaluate five real-world tasks (Figure 4): *Insert Flower*, *Capsule Toy*, *Insert Pen*, *Make Tanghulu*, and *Open Drawer*. These tasks cover grasping, precise insertion, and articulated-object interaction. Task, hardware, and data details are provided in Appendices A.1 and A.3.

Table 2: Main results on five real-world tasks. Score: Task Completion Score (0–10); Frame F1 and Traj. F1: frame-level and trajectory-level Macro-F1 (%), respectively. Mean ± SE over 30 trials per model and task.

Task	Score		Status prediction of Echo	
	$\pi_{0.5}$	Echo	Frame F1	Traj. F1
Insert Flower	5.67 ± 0.92	$\mathbf{8.33 \pm 0.69}$	92.18 ± 0.71	96.67 ± 3.33
Capsule Toy	8.40 ± 0.18	$\mathbf{8.57 \pm 0.16}$	88.31 ± 0.57	100.00 ± 0.00
Insert Pen	6.87 ± 0.84	$\mathbf{8.60 \pm 0.46}$	89.33 ± 1.45	96.67 ± 3.33
Make Tanghulu	4.33 ± 0.92	$\mathbf{5.33 \pm 0.93}$	91.11 ± 1.13	93.33 ± 4.63
Open Drawer	5.40 ± 0.77	$\mathbf{7.90 \pm 0.26}$	88.01 ± 1.22	90.00 ± 5.57

Table 3: Comparison on Insert Flower. Score: 0–10; F1: Macro-F1 (%). Mean ± SE over 30 trials. †: binary success/non-success evaluation; other F1 results use three-state evaluation. "–": not evaluated.

Method	Score	Status prediction	
		Frame F1	Traj. F1
CycleVLA	3.00 ± 0.85	43.10 ± 1.80	36.67 ± 8.95
VLA-Corrector	5.67 ± 0.92	–	–
Echo	$\mathbf{8.33 \pm 0.69}$	$\mathbf{92.18 \pm 0.71}$	$\mathbf{96.67 \pm 3.33}$
Robometer†	–	75.54 ± 7.05	90.00 ± 5.57
Echo†	$\mathbf{8.33 \pm 0.69}$	$\mathbf{93.09 \pm 0.81}$	$\mathbf{96.67 \pm 3.33}$

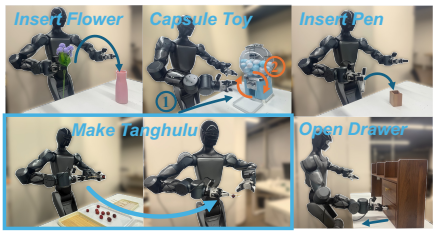

Figure 4: Real-world experimental setups for five manipulation tasks. Arrows illustrate the intended manipulations.

Evaluation metrics. We report three metrics. *Task Completion Score* measures task execution quality on a normalized scale of $[0, 10]$. *Frame-level Macro-F1* compares per-frame predictions with ongoing, success, and failure annotations throughout the trajectory, assessing fine-grained status recognition. *Trajectory-level F1* measures whether an annotated success or failure outcome is correctly recognized within a fixed window starting at the onset of the corresponding interval, assessing the timeliness and practical usability of outcome detection. Within each window, any failure prediction yields a failure decision; otherwise, any success prediction yields success, and detecting neither counts as a missed outcome. For each F1 metric, we average class-wise F1 over the ground-truth classes present in each rollout, then average across rollouts.

Unless otherwise specified, evaluations use 30 trials per model and task, thresholds of 0.5 for the stop signal (SS) and failure signal (FS), and a 15-frame decision window (3 s at a 0.2 s inference interval). We report mean ± standard error (SE), with F1 in percentages. Scoring, annotation, and outcome-decision rules are detailed in Appendix A.2.

Models and training protocol. All experiments use $\pi_{0.5}$ (Intelligence et al., 2025) as the policy backbone. Echo and the action-only baseline use the same successful and recovery demonstrations for action supervision, with matched training steps and optimizer settings. Echo additionally uses failure trajectories and pre-intervention failure segments for status supervision, excluding their actions from imitation. Training details are provided in Appendix A.4.

4.2 Main Results

Task-status prediction. Table 2 shows that Echo learns accurate task-status predictions without additional frame-level progress annotations. Strong frame-level performance indicates that the learned signals track execution states throughout the trajectory, while strong trajectory-level performance on most tasks supports their use for timely outcome recognition. Overall, the results support the proposed supervision scheme for learning signals that capture both local execution states and task outcomes.

Task execution performance. Echo outperforms the action-only $\pi_{0.5}$ baseline on every task in Table 2. Since both models use the same action-supervision data, the consistent improvement supports jointly predicting task status and using the learned signals to condition action generation. The benefit therefore extends beyond monitoring execution to improving task completion. The analyses below examine when these gains are most pronounced and how the signals affect action generation.

4.3 Comparison with Existing Methods

We compare Echo with CycleVLA (Ma et al., 2026), VLA-Corrector (Pan et al., 2026), and Robometer (Liang et al., 2026) on Insert Flower. CycleVLA augments action generation with stop

Table 4: ID/OOD breakdown on Insert Flower and Open Drawer. Each condition contains 15 trials from the main evaluation. Score: 0–10; F1: Echo's status metrics (%). Results are mean ± SE.

Metric / Model	Insert Flower		Open Drawer	
	ID	OOD	ID	OOD
Score / $\pi_{0.5}$	9.33 ± 0.67	2.00 ± 1.07	8.40 ± 0.13	2.40 ± 1.06
Score / Echo	$\mathbf{8.67 \pm 0.91}$	$\mathbf{8.00 \pm 1.07}$	$\mathbf{8.93 \pm 0.18}$	$\mathbf{6.87 \pm 0.29}$
Frame F1 / Echo	93.19 ± 0.93	91.17 ± 1.03	87.27 ± 1.02	88.74 ± 2.24
Traj. F1 / Echo	100.00 ± 0.00	93.33 ± 6.67	93.33 ± 6.67	86.67 ± 9.09

Table 5: Component ablations on Insert Flower. $S \to A$ denotes action attention to both signal groups. Score: 0–10; F1: %. Mean ± SE over 30 trials per variant.

Variant	Score	Frame F1	Traj. F1
Echo	$\mathbf{8.33 \pm 0.69}$	$\mathbf{92.18 \pm 0.71}$	96.67 ± 3.33
w/o $S \to A$	5.00 ± 0.93	90.78 ± 0.49	$\mathbf{100.00 \pm 0.00}$
Full Attention	7.67 ± 0.79	86.14 ± 1.40	100.00 ± 0.00
w/o Async.	7.33 ± 0.82	87.29 ± 1.73	86.67 ± 6.31
Shared MLP	6.67 ± 0.88	78.68 ± 3.31	83.33 ± 6.92

and progress prediction. VLA-Corrector detects discrepancies between predicted and observed visual features to trigger corrective replanning. Robometer is an external reward model that estimates task progress and success from execution videos. Without a termination prediction, Robometer is evaluated using binary success/non-success Macro-F1; we also report Echo under this protocol. Implementations and adaptations to our evaluation protocol are detailed in Appendix A.6. Echo achieves the strongest status-prediction and task-completion performance on the applicable metrics (Table 3).

4.4 Ablation and Mechanism Analysis

We next investigate why Echo improves execution and when its learned signals are most useful. We compare in-distribution and out-of-distribution conditions, ablate the signal–action architecture, directly intervene on the signals, and vary the amount of terminal-failure data.

4.4.1 In-Domain and Out-of-Distribution Analysis

We break down the main results for Insert Flower and Open Drawer into in-distribution (ID) and out-of-distribution (OOD) conditions, with 15 trials each (Table 4). Details are in Appendix A.1.

Echo's gains are larger under OOD conditions, while ID differences are modest and task-dependent (Table 4). Status prediction remains broadly stable: frame-level performance changes modestly, and trajectory-level performance declines slightly under distribution shift.

Successful demonstrations emphasize nominal execution, whereas recovery demonstrations emphasize adjustment and correction. We hypothesize that status signals steer the policy toward the latter behaviors under distribution shift. The signal interventions below directly test this hypothesis.

4.4.2 Component Ablations

We ablate signal–action attention, signal separation, and asynchronous denoising to examine how these choices contribute to status prediction and task execution. All ablations use Insert Flower, with each variant modifying one component while holding the remaining settings fixed (Table 5).

Signal-to-action attention. Blocking action attention to both signal groups (*w/o* $S \to A$) preserves signal quality but reduces task completion to near the action-only baseline. This indicates that the execution gains depend on actions conditioning on the learned signals.

Signal separation. Allowing direct SS–FS attention (*Full Attention*) reduces frame-level accuracy and task completion while preserving strong trajectory-level performance. We observe copying between the signals that weakens their distinct semantics. Sharing multilayer perceptrons (MLPs) across action, SS, and FS causes larger degradation in both prediction and execution, further supporting separate feature transformations.

Asynchronous denoising. Synchronous denoising uses the same flow time for actions and signals during training and inference. It causes only a modest decrease in frame-level accuracy but degrades task completion and timely outcome recognition. This supports making cleaner status estimates available earlier to guide subsequent action updates.

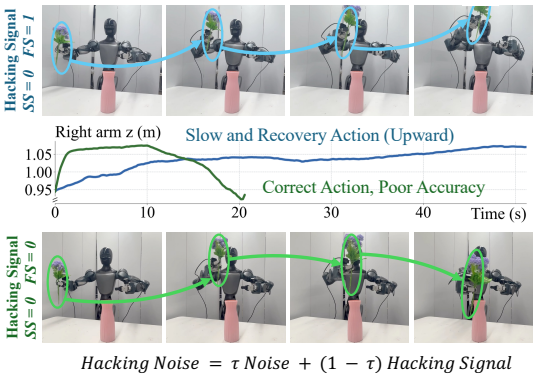

Figure 5: Signal interventions on Insert Flower. With SS $= 0$, imposing FS $= 1$ elicits upward recovery motions (blue), while FS $= 0$ produces task-directed motions (green). Curves show right-arm height over time.

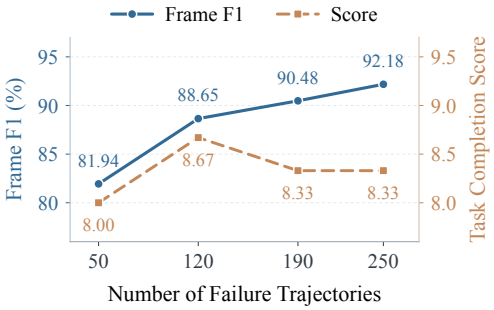

Figure 6: Effect of terminal-failure data quantity on Insert Flower, with its sampling proportion fixed. Curves show mean Frame-level Macro-F1 (left axis) and Task Completion Score (right axis) over 30 trials.

4.4.3 Do Task-Status Signals Influence Action Generation?

On Insert Flower, we impose prescribed signal targets at the corresponding noise level before each denoising step:

$$\mathbf{x}^q_{\tau_q} \leftarrow (1-\tau_q)\mathbf{h}^q + \tau_q \boldsymbol{\epsilon}^q, \qquad q \in \{S, F\}, \tag{14}$$

Here, $\mathbf{h}^q \in \mathbb{R}^{H\times 1}$ is the prescribed target and $\boldsymbol{\epsilon}^q$ is Gaussian noise. We fix SS to zero and FS to either zero or one throughout the chunk, imposing normal execution or recoverable failure, respectively.

In Figure 5, FS $= 0$ produces task-directed motions, whereas sustained FS $= 1$ elicits repeated upward recovery motions. This qualitative intervention shows that status inputs can directly steer action generation toward corrective behavior, supporting their use for biasing the policy toward adjustment and recovery when needed.

4.4.4 Effect of Failure Data Quantity

On Insert Flower, we vary the number of terminal-failure trajectories using nested subsets of 50, 120, 190, and 250 trajectories, with 250 used in the main experiments. The other three data types, training settings, and failure-data sampling proportion remain fixed. Thus, larger subsets provide more diverse failure examples without increasing their share of training samples.

Figure 6 shows that frame-level status prediction improves as the failure dataset grows, while task-completion scores remain broadly stable. This is consistent with the role of terminal-failure data: they supervise the distinction between terminal failure and successful completion, while their actions are excluded from imitation. Broader coverage of terminal-failure states thus enriches status supervision without directly supplying additional corrective actions.

The stable completion scores are also consistent with adjustment and recovery depending primarily on the pre-intervention and recovery data held fixed in this experiment. Pre-intervention failures supervise signals at recoverable errors, while recovery demonstrations provide corrective actions. Together, these observations support the complementary roles of terminal-failure supervision and recovery-related data.

5 Conclusions

We presented Echo, a compact framework that jointly denoises actions and task status signals within a single VLA. Echo learns from successful, failed, and recovery data without frame-level progress annotations, using controlled interactions and asynchronous denoising to guide action generation. Experiments on five real-world tasks demonstrate accurate status prediction and improved task completion, supporting the use of task status signals for both monitoring and action generation.

AI USE STATEMENT

We used generative AI tools to improve the clarity, grammar, and readability of the manuscript. All AI-assisted edits were reviewed by the authors, who take full responsibility for the final content of this work.

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. *arXiv preprint arXiv:2410.04640*, 2024.

Minttu Alakuijala, Reginald McLean, Isaac Woungang, Nariman Farsad, Samuel Kaski, Pekka Marttinen, and Kai Yuan. Video-language critic: Transferable reward functions for language-conditioned robotics. *arXiv preprint arXiv:2405.19988*, 2024.

I Apanasevich, M Artemyev, R Babakyan, P Fedotova, Denis Grankin, E Kupryashin, A Misailidi, D Nerus, A Nutalapati, G Sidorov, et al. Green-vla: Staged vision-language-action model for generalist robots. *arXiv preprint arXiv:2602.00919*, 2026.

Tewodros W Ayalew, Xiao Zhang, Kevin Yuanbo Wu, Tianchong Jiang, Michael Maire, and Matthew R Walter. Progressor: A perceptually guided reward estimator with self-supervised online refinement. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10297–10306. IEEE, 2025.

Atiksh Bhardwaj, Edward Weiyi Duan, Prithwish Dan, Wei-Chiu Ma, and Preston Culbertson. Decoding task progress from vla representations. *arXiv preprint arXiv:2608.13474*, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

Tingjun Dai, Mingfei Han, Tingwen Du, Zhiheng Liu, Zihao Zhang, Zhihui Li, Salman Khan, Jun Yu, and Xiaojun Chang. See, plan, rewind: Progress-aware vision-language-action models for robust robotic manipulation. *arXiv preprint arXiv:2603.09292*, 2026.

Haoyuan Deng, Yitong Gao, Yudong Lin, Haichao Liu, Zhenyu Wu, and Ziwei Wang. Uniintervene: Agentic intervention for efficient real-world reinforcement learning. *arXiv preprint arXiv:2606.12372*, 2026.

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.

Youhe Feng, Hansen Shi, Haoyang Li, Xinlei Guo, Yang Wang, Chengyang Zhang, Jinkai Zhang, Xiaohan Zhang, Jie Tang, and Jing Zhang. Procvlm: Learning procedure-grounded progress rewards for robotic manipulation. *arXiv preprint arXiv:2605.08774*, 2026.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40041–40076, 2026.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Yangzheng Wu, Tengyue Ba, Zhanguang Zhang, and Yingxue Zhang. Stageguard: Learning stage transitions for long-horizon robot tasks via agentic distillation. *arXiv preprint arXiv:2609.20791*, 2026.

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

Seungsu Kim, Jinyoung Choi, Seungmin Baek, and Jean-Michel Renders. Progvla: Progress-aware robot manipulation skill learning. *arXiv preprint arXiv:2605.28231*, 2026.

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

Wenhao Li, Xiu Su, Dan Niu, Yichao Cao, Hongyan Xu, Zhe Qu, Lei Fan, Shan You, and Chang Xu. Sentinel-vla: A metacognitive vla model with active status monitoring for dynamic reasoning and error recovery. *arXiv preprint arXiv:2605.01191*, 2026.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S Huang, Luke Zettlemoyer, Dieter Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bihan Wen. Failsafe: Reasoning and recovery from failures in vision-language-action models. *arXiv preprint arXiv:2510.01642*, 2025.

Chenyv Liu, Wentao Tan, Lei Zhu, Fengling Li, Jingjing Li, Guoli Yang, and Heng Tao Shen. Self-correcting vla: Online action refinement via sparse world imagination. *arXiv preprint arXiv:2602.21633*, 2026a.

Huihan Liu, Shivin Dass, Roberto Martín-Martín, and Yuke Zhu. Model-based runtime monitoring with interactive imitation learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4154–4161. IEEE, 2024.

Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, et al. Palm: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *arXiv preprint arXiv:2601.07060*, 2026b.

Weijia Liufu, Xiaoyu Guo, Ruiyi Chen, Jingzhi Liu, Kaidong Zhang, Xiwen Liang, Jianqi Lin, Dawei Sun, Yuze Wang, Rongtao Xu, et al. Repo-vla: Recovery-driven policy optimization for vision-language-action models. *arXiv preprint arXiv:2605.09410*, 2026.

Chenyang Ma, Kai Lu, Guangyu Yang, Jiuming Liu, Shitong Xu, Bill Byrne, Ioannis Havoutis, 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.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning*, pp. 23301–23320. PMLR, 2023.

Paul Pacaud, Ricardo Garcia, Shizhe Chen, and Cordelia Schmid. Scaling cross-environment failure reasoning data for vision-language robotic manipulation. *arXiv preprint arXiv:2512.01946*, 2025.

Yi Pan, Miao Pan, Qi Lu, Jiaming Huang, Man Zhang, Siteng Huang, Xin Li, Jie Zhang, Yongliang Shen, Xuhong Zhang, et al. Vla-corrector: Lightweight detect-and-correct inference for adaptive action horizon. *arXiv preprint arXiv:2607.01804*, 2026.

Seongheon Park, Wendi Li, Changdae Oh, Samuel Yeh, Zsolt Kira, Michael Hagenow, and Sharon Li. Hide-and-seek in trajectories: Discovering failure signals for vla runtime monitoring. *arXiv preprint arXiv:2605.30834*, 2026.

Xiaomi Robotics Team, Jun Guo, Piaopiao Jin, Jason Li, Peiyan Li, Yingyan Li, Futeng Liu, Wanli Peng, Optimus Qin, Yifei Su, et al. Xiaomi-robotics-1: Scaling vision-language-action models with over 100k hours of real-world trajectories. *arXiv preprint arXiv:2607.15330*, 2026.

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. *arXiv preprint arXiv:2503.08558*, 2025.

Hongyu Yan, Qiwei Li, Jiaolong Yang, and Yadong Mu. Progressvla: Progress-guided diffusion policy for vision-language robotic manipulation. *arXiv preprint arXiv:2603.27670*, 2026.

Ran Yang, Zijian An, Lifeng Zhou, and Yiming Feng. Seqvla: Sequential task execution for long-horizon manipulation with completion-aware vision-language-action model. *arXiv preprint arXiv:2509.14138*, 2025.

Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Chenyang Zhao, Piaopiao Jin, Guokang Sun, Shaoqing Xu, et al. Fpc-vla: A vision-language-action framework with a supervisor for failure prediction and correction. *Expert Systems with Applications*, pp. 131742, 2026.

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.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. *arXiv preprint arXiv:2505.10911*, 2025.

Zijian Zhang, Kaiyuan Zheng, Zhaorun Chen, Joel Jang, Yi Li, Siwei Han, Chaoqi Wang, Mingyu Ding, Dieter Fox, and Huaxiu Yao. Grape: Generalizing robot policy via preference alignment. *arXiv preprint arXiv:2411.19309*, 2024.

Ganlong Zhao, Zijia Tang, Xingping Chen, Zhanghui Kuang, Ye Tian, and Guanbin Li. Flare: A failure-aware framework for autonomous correction and recovery in visual-language robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22391–22401, 2026.

A Appendix

A.1 Robot Platform and Task Details

Robot platform. Data collection and policy deployment use a wheeled humanoid robot with two seven-degree-of-freedom arms, gripper end effectors, and a waist–leg mechanism with six degrees of freedom. The mobile base remains stationary during task execution. The policy controls a total of 20 degrees of freedom through the two arm end effectors, the grippers, and the waist–leg mechanism. Visual observations come from three RGB cameras: one on the head and one on each wrist.

Common evaluation protocol. All tasks start from the same initial robot state. Task objects are placed at the evaluation positions described below. Each episode has a maximum duration of five minutes. An episode is classified as a failure if the task remains incomplete at the time limit, or if the robot reaches an unsafe posture indicating an imminent risk of tipping before completing the task.

Task objectives and evaluation positions. We evaluate five tasks: Insert Flower, Capsule Toy, Insert Pen, Make Tanghulu, and Open Drawer. Table 6 summarizes their objectives and evaluation positions.

Table 6: Task objectives and evaluation positions. All tasks share the initial robot state and the five-minute episode limit. ID and OOD denote in-domain and out-of-distribution positions, respectively.

Task	Objective	Evaluation positions
Insert Flower	Grasp a flower and insert it into a vase.	3 random ID and 3 OOD tabletop positions; 5 trials each (30 total).
Capsule Toy	Rotate the machine's knob until a capsule is dispensed.	–
Insert Pen	Grasp a pen and insert it into a pen holder.	6 random tabletop positions; no explicit ID/OOD split (30 trials).
Make Tanghulu	Insert a bamboo skewer into a hawthorn fruit.	–
Open Drawer	Pull the drawer open.	1 ID and 1 OOD position; 15 trials each (30 total).

Task outcomes and annotation boundaries. For Insert Flower, successful insertion constitutes task success. A dropped flower followed by the arm returning to its reset position without attempting to grasp it again constitutes failure. Insert Pen follows the analogous insertion and failure protocol.

For Make Tanghulu, success requires inserting the bamboo skewer into the hawthorn and retaining the fruit on the skewer after release. An episode is classified as a failure if insertion is not achieved or if the hawthorn falls off after release. The latter receives partial credit of 5 points, as specified in Table 7. Open Drawer succeeds when the drawer is pulled open, and Capsule Toy succeeds when knob rotation causes a capsule to be dispensed. Episodes that remain unsuccessful are subject to the common termination conditions above.

A.2 Evaluation Protocol and Task Scoring

A.2.1 Task Completion Score

Each episode has a maximum task score of 10. Successful episodes receive 10 points before task-specific deductions. Failed episodes receive 0 points, except in Make Tanghulu when insertion is achieved but the hawthorn subsequently falls off the skewer after release; these failed episodes receive 5 points as partial credit. For Insert Flower and Insert Pen, each contact event incurs a one-point deduction, capped at three points. For Capsule Toy and Open Drawer, each empty grasp attempt incurs a one-point deduction, capped at five points.

Table 7: Task-specific scoring rules. Successful episodes receive 10 points minus the applicable deductions. Failed episodes receive 0 points, except for Make Tanghulu episodes in which the hawthorn falls off after insertion and release; these receive 5 points as partial credit.

Task	Deduction rule	Maximum deduction
Insert Flower	1 point per contact with the vase wall.	3 points
Capsule Toy	1 point per empty grasp attempt.	5 points
Insert Pen	1 point per contact with the pen holder.	3 points
Make Tanghulu	5 points if the hawthorn falls off the skewer after release.	5 points
Open Drawer	1 point per empty grasp attempt.	5 points

A.2.2 Frame-Level and Trajectory-Level Metrics

Predicted execution states follow the SS/FS threshold rule defined in the main text. We compute each metric separately for each episode and then report its mean and standard error across episodes.

Frame-level Macro-F1. For episode i and class c, we compute

$$F1_{i,c} = \frac{2TP_{i,c}}{2TP_{i,c} + FP_{i,c} + FN_{i,c}}, \qquad F_i^{\text{frame}} = \frac{1}{|\mathcal{C}_i|} \sum_{c \in \mathcal{C}_i} F1_{i,c}, \tag{15}$$

where $\mathcal{C}_i$ contains the execution-state classes present in the ground-truth annotations of episode i.

Trajectory outcome recognition. For each annotated success or failure interval, the evaluator examines the first W available predictions starting at its ground-truth boundary. If any prediction is failure, the predicted outcome is failure. Otherwise, it is success if any prediction is success, and ongoing if neither is detected. When the annotated interval contains fewer than W frames, all available frames in that interval are used.

We compute outcome F1 within each episode over the outcome classes present in its ground truth. An ongoing prediction counts as a missed outcome. The resulting per-episode values are then aggregated across episodes. For the results reported in the main text, we use $W = 15$ frames. The policy is queried at 5 frames per second, so this window corresponds to 3 seconds.

Aggregation across episodes. Each model is evaluated for 30 episodes per task. For an episode-level metric q_i, we report

$$\bar{q} = \frac{1}{N} \sum_{i=1}^{N} q_i, \qquad \text{SE}(\bar{q}) = \sqrt{\frac{\sum_{i=1}^{N} (q_i - \bar{q})^2}{N(N-1)}}. \tag{16}$$

The same episode-first aggregation is used for task score, frame-level Macro-F1 and trajectory-level F1.

A.3 Data Sampling Ratio

For every task, we use the same training sampling ratio of $20 : 1 : 2 : 20$ for Success, Failure, Pre-intervention failure, and Recovery data, respectively.

A.4 Implementation, Hyperparameters, and Inference Cost

A.4.1 Training and Deployment Settings

The main experiments and most ablations use $\pi_{0.5}$. For each task, Echo and its action-only baseline use matched optimizer settings and training steps. Actions, SS and FS use separate token groups

Table 8: Training, model and deployment hyperparameters.

Hyperparameter	Description	Value
H	Action chunk length	60
λ_A	Action loss weight	1
λ_S	Stop signal loss weight (non-padding chunks)	0.1
λ_F	Failure signal loss weight (non-padding chunks)	1
τ_{clean}	Signal clean-endpoint time (SS and FS)	0.5
δ_s	Stop signal threshold	0.5
δ_f	Failure signal threshold	0.5
W	Trajectory outcome window	15 frames (3 s)
Query rate	Policy inference frequency	5 Hz

with separate feature-transformation MLPs and output projections. Signal flow times follow the asynchronous schedule in the method section, with $\tau_{\text{clean}} = 0.5$ shared by SS and FS. Both signals reach their clean endpoints when the action flow time reaches $\tau_A = 0.5$. Table 8 summarizes the training and inference hyperparameters.

For chunks containing padding, both signal loss weights are set to 1, following the supervision rule in the main text.

A.4.2 Inference Overhead

Despite increasing the generated-token sequence from 60 to 180 tokens and adding separate MLPs, Echo increases inference latency on an NVIDIA RTX 4090 by only 5 ms, from 115 ms to 120 ms, introducing little additional inference overhead.

A.5 Implementation of the Motivating Experiment

The motivating experiment examines how intermediate action and status estimates evolve during a synchronous denoising process. Its purpose is to determine when status information becomes available before introducing the asynchronous schedule.

Synchronous reference model. We use a baseline that denoises actions, SS, and FS synchronously, assigning the same flow time to all three groups:

$$\tau_A = \tau_S = \tau_F = \tau. \tag{17}$$

This corresponds to $\tau_{\text{clean}} = 0$ during both training and inference, instead of the $\tau_{\text{clean}} = 0.5$ used by Echo. All other architectural designs are the same as Echo, including the signal–action attention mask, separate MLPs, and group-specific projections. In particular, action tokens and each signal group can attend to one another, while direct attention between SS and FS remains blocked. The reference model jointly predicts actions and both signals; it is distinct from the action-only $\pi_{0.5}$ baseline.

Intermediate clean-sample estimates. We record all ten denoising steps, indexed from 0 to 9, within a single inference query. The observation, robot state, and instruction are held fixed throughout that query. At each step k, we record the current noisy state $\mathbf{x}_{\tau_k}$ and the velocity predicted by the model. For each group $q \in \{A, S, F\}$, we reconstruct its clean-sample estimate as

$$\hat{\mathbf{x}}^q_{0,k} = \mathbf{x}^q_{\tau_k} - \tau_k \mathbf{v}^q_\theta(\mathbf{x}_{\tau_k}, \tau_k, \mathbf{c}). \tag{18}$$

The velocity prediction depends on the joint noisy input and the conditioning context $\mathbf{c}$. This reconstruction follows from the linear flow path: if the predicted velocity equals $\boldsymbol{\epsilon}^q - \mathbf{x}^q_0$, subtracting τ_k times that velocity recovers $\mathbf{x}^q_0$. With a learned velocity, it provides an estimate of the clean output at the current step.

The reconstruction is used for analysis and does not replace the noisy state in the inference procedure. This distinction is important: directly plotting $\mathbf{x}_{\tau_k}$ would mix the predicted content with

the noise remaining at that flow time. The experiment instead compares clean-sample estimates produced at different denoising steps.

Action and signal visualizations. Figure 3(a) plots the right-arm vertical action component across the $H = 60$ chunk positions, expressed in millimeters. Each curve corresponds to a denoising step, with darker curves indicating later steps. Figure 3(b) summarizes each reconstructed signal chunk using its mean:

$$\hat{\bar{s}}_k = \frac{1}{H}\sum_{j=0}^{H-1} \hat{s}_{0,k,j}, \qquad \hat{\bar{f}}_k = \frac{1}{H}\sum_{j=0}^{H-1} \hat{f}_{0,k,j}. \tag{19}$$

The plotted steps are indexed from 0 to 9. SS and FS use separate vertical axes, both marked with a 10^{-2} scale, so their displayed magnitudes should be read from the corresponding axes.

Interpretation. In the plotted example, SS is close to zero from the earliest step, and FS approaches its later estimate within the first few steps. The action curves continue to change, including between the last two plotted steps. The figure therefore illustrates that useful status estimates can stabilize earlier than the continuous action sequence. It motivates making cleaner signal estimates available to subsequent action updates; it does not imply that every signal prediction is correct or that all action dimensions require the same number of denoising steps.

A.6 Baseline Implementations and Performance Analysis

We compare action-only $\pi_{0.5}$, CycleVLA, VLA-Corrector, Robometer and Echo on Insert Flower. This section distinguishes original methods from adaptations in our setup.

Action-only $\pi_{0.5}$. The baseline uses the same successful and recovery action-supervision data as Echo, with matched optimizer settings and training steps. It predicts actions without SS or FS, so status metrics are not assigned to it. During implementation, we ensured that key settings including training data, model learning rate, and optimizer were consistent with Echo. The model was trained for the same number of steps until convergence.

CycleVLA. CycleVLA combines a progress-aware VLA with external failure prediction and subtask backtracking. When reproducing its performance, we trained using demonstration success data and DAgger recovery data. And we adopted a learning rate of 5e-5 and the Adam optimizer for 20k training steps (aligned with the training protocol of Echo). Since its stop signal is only supervised on the final frame, and success/failure judgment mainly relies on the progress signal, we take the maximum stop signal within the model's actually executed frames and the average of the corresponding progress signals, apply a threshold of 0.5 to both quantities, and construct frame-level and trajectory-level state prediction discrimination accordingly. As illustrated in Figure 7, since CycleVLA does not leverage real-world policy failure data, it exhibits weak discriminative capability for real-world failures.

VLA-Corrector. The VLA-Corrector considered here compares predicted and observed visual feature evolution using a latent-space vision monitor. Persistent deviations trigger action-chunk truncation and corrective replanning with Online Gradient Guidance. It does not itself establish terminal failure, so status metrics are not assigned to it.

When reproducing its performance, we adopted the frozen baseline $\pi_{0.5}$ model and trained the corrector for 10k iterations following the recommended configuration in its original code. In practical testing, we corrected the VLA action chunk via the Online Gradient Guidance approach and the cutoff-only method, then we compared their performance and reported the better-performing one.

Robometer. Robometer is an external video-language reward model with progress and success predictions trained using trajectory-based supervision, including preference comparisons. When reproducing its performance, we trained the officially released Robometer-4B base model via LoRA fine-tuning. We trained for 20k iterations on all data of the Insert Flower task using its officially recommended configuration and evaluated its performance offline. Since Robometer does not provide estimations for task termination, we classify all "in-progress" samples as failures during evaluation,

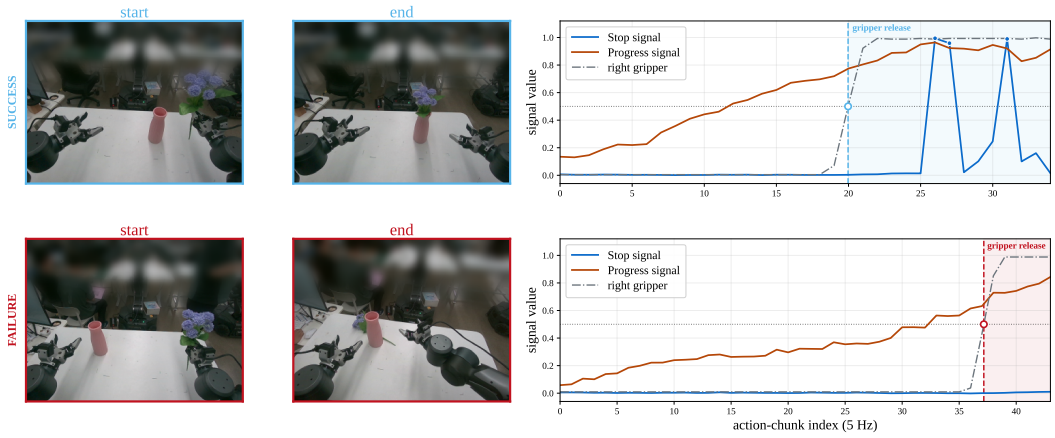

Figure 7: Signal values estimated by CycleVLA. The model exhibits weak discriminative capability for real-world failures.

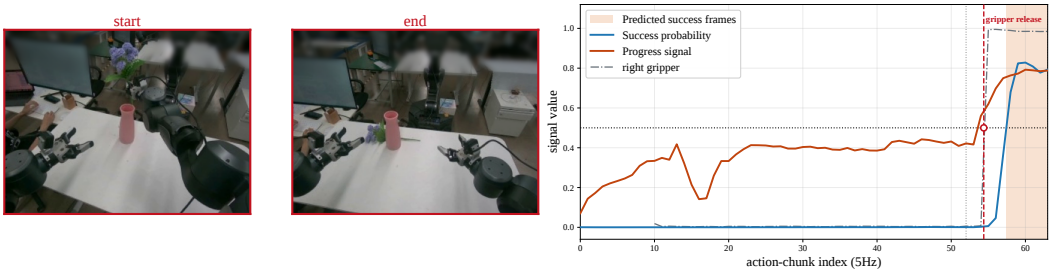

Figure 8: Signal values estimated by Robometer. The model sometimes lacks discriminative capability for real-world failures.

reducing the original 3-class classification problem to binary classification. We compute the final metrics by applying a threshold of 0.5 on the success probability output by the model.

As shown in Figure 8, similar to CycleVLA, Robometer sometimes lacks discriminative ability for real-world failures.

A.7 Detailed Task-Status Prediction Results

Table 9 provides the complete numerical results for Echo under the default evaluation settings: $\delta_s = \delta_f = 0.5$ and $W = 15$ frames. In addition to the frame-level and trajectory-level F1 scores reported in the main text, we provide overall frame accuracy, frame-level recall for each execution state, and trajectory-level success and failure recall.

Ongoing-state recall exceeds 99% on every task, while frame-level recognition of success and failure is lower. The trajectory-level results separately show whether each outcome is recognized within the evaluation window. The counts accompanying trajectory recall make the different numbers of successful and failed episodes explicit.

A.8 Hyperparameter Sensitivity

Figure 9 evaluates status-prediction sensitivity across all five tasks using two parameter grids. For frame-level evaluation, we independently vary the SS and FS thresholds over $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. For trajectory-level evaluation, we use a shared threshold $\delta_s = \delta_f = \delta$ with $\delta \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and vary the outcome window over $W \in \{5, 10, 15, 20, 25\}$ frames. At 5 Hz, these windows correspond to 1–5 seconds. Each setting is evaluated on the recorded episodes, with F1 computed per episode and then averaged across episodes.

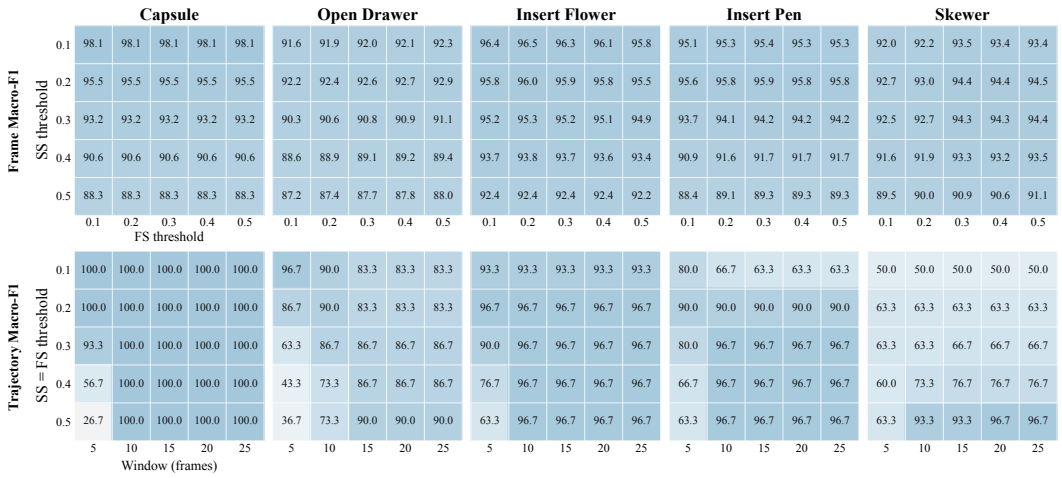

Figure 9: Hyperparameter sensitivity across five tasks. Top: frame-level Macro-F1 under independently varied SS and FS thresholds. Bottom: trajectory-level F1 under a shared SS/FS threshold and varying outcome-window lengths. Values are percentages. Capsule and Skewer denote Capsule Toy and Make Tanghulu, respectively. The main results use $\delta_s = \delta_f = 0.5$ and $W = 15$ frames.

Table 9: Detailed task-status prediction results for Echo on five tasks. All metrics are percentages. Frame F1 denotes frame-level Macro-F1 and Traj. F1 denotes trajectory-level F1; both are reported as mean $\pm$ SE over 30 episodes per task. Trajectory recall includes the number of correctly recognized outcomes over the number of ground-truth outcomes in parentheses. A dash indicates that no ground-truth failure examples are present.

(a) Overall performance

Task	Frame F1	Traj. F1	Frame Acc.
Insert Flower	92.18 $\pm$ 0.71	96.67 $\pm$ 3.33	96.00
Capsule Toy	88.31 $\pm$ 0.57	100.00 $\pm$ 0.00	94.58
Insert Pen	89.33 $\pm$ 1.45	96.67 $\pm$ 3.33	92.18
Make Tanghulu	91.11 $\pm$ 1.13	93.33 $\pm$ 4.63	95.67
Open Drawer	88.01 $\pm$ 1.22	90.00 $\pm$ 5.57	96.51

(b) Class-wise recognition

	Frame-level recall			Trajectory-level recall	
Task	Ongoing	Success	Failure	Success	Failure
Insert Flower	99.96	77.25	79.78	96.00 (24/25)	100.00 (5/5)
Capsule Toy	100.00	68.02	–	100.00 (30/30)	–
Insert Pen	100.00	77.17	75.00	96.43 (27/28)	100.00 (2/2)
Make Tanghulu	99.96	78.60	77.28	93.75 (15/16)	92.86 (13/14)
Open Drawer	99.79	66.06	–	90.00 (27/30)	–

Frame-level threshold sensitivity. Frame-level performance is generally more sensitive to the SS threshold than to the FS threshold over the tested ranges. For example, Capsule Toy changes from 98.1 at an SS threshold of 0.1 to 88.3 at 0.5, while remaining unchanged across the tested FS thresholds. The other tasks also show relatively modest changes along the FS-threshold axis, although the best SS threshold varies across tasks. Thus, a single threshold setting does not maximize frame-level F1 for every task.

Trajectory-level threshold and window sensitivity. Short outcome windows can miss predictions that become correct later in the outcome interval. At the default shared threshold of 0.5,

increasing W from 5 to 15 frames raises trajectory-level F1 from 26.7 to 100.0 on Capsule Toy, from 36.7 to 90.0 on Open Drawer, and from 63.3 to 96.7 on both Insert Flower and Insert Pen. Make Tanghulu increases from 63.3 to 93.3 over the same range and reaches 96.7 with a 20-frame window.

Longer windows do not compensate for every threshold choice. For example, Make Tanghulu remains at 50.0 across all tested window lengths when the shared threshold is 0.1. The heatmaps therefore show a task-dependent interaction between thresholds and the time allowed for outcome recognition. Around the default setting, several tasks exhibit a plateau once the window is sufficiently long, whereas Make Tanghulu benefits from a slightly longer window. These results support reporting the common default configuration together with its sensitivity, rather than assuming that larger windows or lower thresholds always improve recognition.

The window W is used only for trajectory-level evaluation and does not change the frame-level predictions. These offline comparisons characterize status evaluation; they do not alter the actions or task-completion scores of the recorded episodes.

A.9 Compatibility with External Monitoring and Planning

Echo improves the VLA policy itself by allowing learned task-status information to participate in action generation. This objective is compatible with external monitoring, recovery and planning modules. SS and FS can also provide execution-status information to a higher-level controller using the signal semantics defined in the main text.

An external controller could combine these signals with broader temporal context or additional observations to verify completion, request intervention or replan. Echo supplies policy-internal status estimates and corrective action generation, while the external module adds its own decision logic. This is architectural compatibility; improved performance from a particular combined system requires a separate evaluation.