PACE: Progress Actively-internalized Conditioning Execution for Long-horizon Manipulation

Fei Ni, Zhuo Chen, Xianze Yao, Zerui Chen, Changrui Chen,
Yifu Yuan, Shan Luo, Jianye Hao, Jiankang Deng
MVP Lab, Imperial College London King's College London
https://PACE-embodied-robotics.github.io

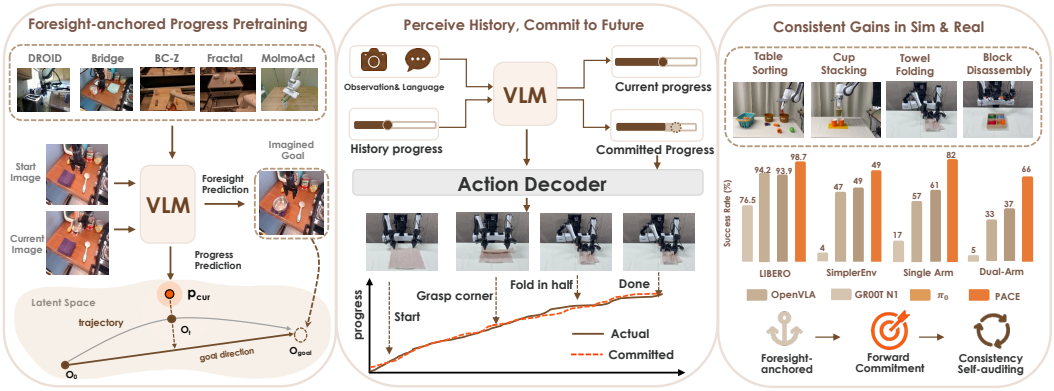

Figure 1: **Overview of PACE,** which actively internalizes progress as an interface bridging commitment and execution. The model predicts a foresight and its anchored progress, then extends with historical and forward increments, to equip policy with a calibrated *pace* sense for long-horizon tasks.

Abstract

Vision-language-action (VLA) models have made rapid progress in robotic manipulation, yet the policy acts with no sense of the *pace* at which its own execution unfolds, whether it is advancing, stalling, or has quietly gone wrong, especially for long-horizon tasks. Recent methods supply progress through external reward models or as an auxiliary prediction inside the policy, but the signal runs alongside action generation rather than actively shaping it. Moreover, progress is inherently tied to the terminal state, making estimation fragile when the policy lacks an internal awareness of where the task should end. To this end, we propose **PACE**, which actively internalizes task progress as a predictive interface bridging commitment and execution, equipping the policy with a calibrated sense of the *pace* at which it advances toward task completion. Specifically, we first equip the VLA with a *foresight anchor*, a predicted terminal representation from observation and instruction, from which progress emerges as a geometric projection. Building on this anchor, the model perceives a compact past–present–future progress state capturing *how much the last step advanced*, *how far execution has come*, and *how far the next step should go*. This predicted progress further actively conditions the action decoder to guide the prediction, while the gap between committed and realized increments provides a built-in consistency check for lightweight failure detection. Experiments across LIBERO, SimplerEnv, and real-world single-arm and bimanual platforms show that PACE achieves 98.7% on LIBERO and 74% on real-world manipulation, with consistent gains especially on long-horizon tasks. Robustness and stable generalization across diverse scenarios further confirm the advantage of internalizing progress within the policy, validating a calibrated sense of execution pace as an effective paradigm for long-horizon robot control.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Large-scale pretraining has substantially advanced robotic manipulation, enabling Vision-Language-Action (VLA) models to map multimodal observations and language instructions directly to robot actions [20, 5, 3, 7]. These systems demonstrate strong semantic generalization on short-horizon tasks, yet long-horizon manipulation exposes a critical blind spot: current VLAs have no internal representation of task progress. Consider a robot instructed to "open the drawer and place the bowl inside". The policy generates plausible actions at every step, yet without knowing how far the overall task has progressed, it cannot tell whether it should still be reaching for the handle, has already opened the drawer, or needs to recover from a missed grasp. This lack of progress awareness compounds over long horizons, causing robots to repeat ineffective actions, switch subtasks prematurely, or continue executing long after the task has quietly derailed. Action supervision teaches which motor chunk follows an observation, but never explicitly teaches what task-level advancement that chunk should produce, leaving the policy without any sense of the *pace* at which its own execution unfolds.

A growing body of work has begun to supply progress or evaluative signals to address this gap. *External progress and reward models* such as RoboReward [24], Robometer [27], TOPReward [9], and Robo-Dopamine [44] train standalone estimators that score trajectories offline, producing dense labels for weighted behavioral cloning [8, 9] or demonstration filtering. Building on such estimators, *external guidance methods* feed progress into action generation from outside the policy: the concurrent ProgressVLA [46] uses a trained progress classifier to steer diffusion sampling at inference, and EVOLVE-VLA [2] repurposes the VLAC critic [53] as a dense reward for test-time reinforcement learning. These approaches do shape actions through progress, but the signal originates from a separate model rather than from the policy's own forward pass. A third strategy moves the progress signal inside the model as an *auxiliary prediction*. SeqVLA [47] predicts subtask completion to trigger stage transitions, keeping progress at a coarse discrete level that cannot capture continuous advancement within a stage. PALM [29] goes further by co-decoding a continuous progress value alongside actions within the same forward pass but the predicted progress does not re-enter the decoder to shape the action, leaving it a parallel output that the motor system cannot act on. Producing progress inside the model is a necessary step toward internalization, but until the signal actively participates in shaping which actions are generated, it remains an observation about execution rather than a force that drives it. This raises a fundamental question: *Can a VLA learn to truly internalize task progress, perceiving it from observation and actively conditioning action generation on it?*

To this end, we propose PACE, which actively internalizes task progress as a predictive interface bridging commitment and execution within the VLA. Our key insight is that a robot cannot gauge how far it has come without first forming an internal representation of where the task is going. We therefore equip the VLA with a *foresight anchor*, a predicted terminal-state representation derived from observation and language instruction alone, requiring no goal image or reference demonstration at inference. From this anchor, progress emerges as a geometric projection that grounds where execution stands on the start-to-goal axis. Building on this foresight-anchored progress, the model perceives a compact past–present–future temporal state capturing *how much the last step advanced*, *how far execution has come*, and *how far the next step should go*, equipping the policy with a calibrated sense of its own execution *pace*. These progress variables directly condition the action generator, closing the perceive–commit–condition loop that distinguishes PACE from methods that treat progress as a parallel output alongside actions [29] or route it through an external classifier outside the policy forward pass [46]. The same forward commitment also yields a built-in consistency check at inference, where the gap between committed and realized increments exposes emerging failures without any additional learned component. Comprehensive evaluations across LIBERO, SimplerEnv, and real-world single-arm and bimanual platforms show that PACE achieves 98.7% on LIBERO and 74% on real-world tasks, substantially outperforming strong VLA baselines, with the internalization advantage persisting under distribution shift and the commit-realize gap providing reliable failure detection at zero additional cost. Our contributions are summarized as follows:

- We propose PACE, a VLA framework that internalizes task progress as a predictive interface by predicting a foresight anchor and deriving anchored progress as a geometric projection, equipping the policy with a calibrated sense of execution *pace* for long-horizon manipulation tasks.
- We design a perceive–commit–condition loop in which historical pace, current progress, and a forward commitment jointly condition action generation, closing the gap between progress perception and action so that progress actively shapes execution rather than running alongside it.

- PACE demonstrates substantial gains over strong VLA baselines across diverse tasks and platforms, with robust progress internalization that persists under distribution shift and a commit-realize progress gap that serves as a built-in consistency check for potential execution failures.

2 Related Work

VLA Models and Long-Horizon Control. General-purpose VLA models map multimodal observations and language instructions to robot actions [20, 5, 3, 7, 14, 37], but their policy interface remains agnostic to task-level advancement. Recent work addresses this from several angles: predicting goal images or subgoal frames to shorten planning horizons [4, 43, 35, 36], exposing chain-of-thought traces [56] or spatial milestones [13] as intermediate reasoning, learning future-facing representations through visual prediction or world models [15, 49], aligning execution with reference demonstrations [50, 11], and detecting subtask completion to trigger stage transitions [47]. These methods collectively show that making temporal structure visible to the policy benefits long-horizon control, yet continuous task progress, a signal that directly reflects how far execution has advanced and how far it should go, remains underexplored as a policy interface. Our work addresses this gap by equipping the VLA with an explicit sense of execution *pace*, where progress is perceived, committed to, and fed directly into action generation.

Progress Estimation and Reward Models. Dense evaluative signals have become an important complement to sparse success labels, with recent work approaching progress estimation from diverse angles [55, 17]. Embedding-based methods such as VIP [32], LIV [31], R3M [33], and DINOv2 [39] learn visual representations whose geometry correlates with task advancement, measuring progress as distance to a goal state in a learned latent space. VLM-based estimators scale this idea beyond fixed visual encoders: TOPReward [9] extracts progress from token probability distributions without additional training, while Robo-Dopamine [44] distills process-level supervision from step-aware multi-view annotations. Trained on large-scale trajectory data, RoboReward [24] and Robometer [27] fine-tune vision-language models such as Qwen3VL [1] into general-purpose reward predictors that transfer across embodiments, while VLAC [53] learns a vision-language-action critic for scalar progress regression. Despite their diversity, all these methods estimate progress through a standalone model and deliver it to the policy as an external signal, never integrating it into the action-generation pathway itself. Our progress interface can leverage any of these estimators as a supervision source while internalizing the resulting signal within the VLA's own forward pass.

Progress-Informed Robot Learning. Given a progress or reward signal, a natural question is how best to couple it with the policy. Offline methods such as SARM [8] and advantage-weighted behavioral cloning with TOPReward [9] use progress to weight or filter demonstrations before training, improving data quality but leaving the policy unaware of progress at inference. External guidance methods feed progress into action generation from outside the policy, where ProgressVLA [46] steers diffusion sampling through a progress classifier, EVOLVE-VLA [2] repurposes the VLAC critic [53] as a dense reward for test-time reinforcement learning, and RoVer [12], RoboMonkey [22], and FPC-VLA [48] apply learned reward models or error detectors [51] as inference-time verifiers. Moving inside the model, PALM [29] jointly decodes actions and a progress value, yet progress remains a co-decoded output that does not re-enter action generation as an explicit conditioning. Our framework PACE closes this gap by fully internalizing progress within a single forward pass, where the model predicts a foresight anchor, perceives progress as a geometric projection onto it, commits to a forward increment, and conditions action generation on both, while the same commitment yields a built-in consistency signal that compares realized progress against the policy's own prediction, especially for long-horizon tasks.

3 Method

PACE turns task progress into an internal predictive interface that bridges perception and action within the VLA. The policy perceives its current progress from observation, commits to a forward increment, and conditions its own actions on both, all within a single forward pass. We build this interface in four sections. Section 3.1 trains a foresight anchor and reads current progress p_{cur} as a geometric projection onto it. Section 3.2 extends this static readout with a realized historical increment Δp_{prev} and a forward commitment Δp_{cur}, forming a compact past–present–future interface. Section 3.3 feeds the interface into the action generator, so progress shapes the next action chunk

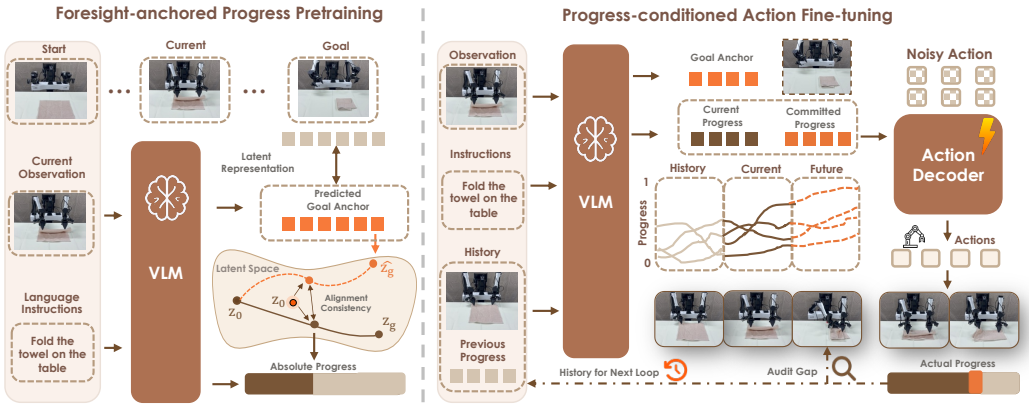

Figure 2: **Overview of PACE**, which internalizes progress as an interface bridging perception and action. The model first predicts a foresight anchor and its anchored current progress, then extends with historical and forward increments to condition action prediction on a unified past–present–future view, while the commit-realize gap doubles as a lightweight inference-time audit.

rather than running alongside it. Section 3.4 reuses the commitment at inference, where its gap with the realized increment yields an execution-consistency signal without any new learned component.

3.1 Foresight-Anchored Progress Pretraining

Progress is inherently a relative quantity that measures how far execution has advanced between a known start and a known goal. Without an internal goal representation, the VLA has no stable reference against which to anchor a progress reading, and the resulting prediction becomes fragile, sensitive to variation in pace, viewpoint, or initial configuration that a goal-anchored measure would absorb. We therefore equip the VLA with a *foresight anchor*, a predicted representation of the task's terminal visual state from the current observation and language instruction, which supplies the stable reference from which progress emerges as a geometric projection onto the start-to-anchor direction in latent space. The anchor and progress readout are produced together in a single forward pass.

Foresight anchor prediction. We append K learnable query tokens q_{goal} to the VLA input sequence. Their layer-18 hidden states are mean-pooled and projected through an MLP head to a predicted terminal embedding $\hat{z}_g \in \mathbb{R}^d$, supervised against the frozen visual embedding of the goal frame:

$$\mathcal{L}_{\text{goal}} \;=\; \big\|\hat{z}_g - f_\psi(o_g)\big\|_2^2, \tag{1}$$

where f_ψ is a frozen visual encoder. During training, $\mathcal{L}_{\text{goal}}$ shapes the query read-outs toward representations that track proximity to task completion, making intermediate states easier to localize along the task-specific progress direction. At inference, the anchor is re-predicted at each step from observation and language alone, requiring no goal image, and provides a consistent reference frame for progress computation throughout the rollout.

Foresight-anchored progress. With the anchor in hand, progress reduces to a single geometric operation. For a tuple (o_s, o_t, o_g), the progress of o_t is the signed projection of its frozen embedding onto the start-to-goal displacement,

$$p(z;\, z_s, z_g) \;=\; \frac{\langle z - z_s,\; z_g - z_s\rangle}{\|z_g - z_s\|_2^2}, \qquad z_i = f_\psi(o_i), \tag{2}$$

yielding a geometric ground truth $p_t^{\text{gt}} = p(z_t;\, z_s, z_g)$ that reads exactly 0 at the start and 1 at the goal by construction; intermediate frames can project outside this range, so the target is clamped to $[0, 1]$ (Appendix A.1). This projection captures signed displacement along the start-to-goal direction in latent space, reflecting both direction and magnitude of advancement regardless of trajectory duration. We additionally train a scalar readout head on a dedicated query token q_p to predict p_t^{pred} directly from the VLA hidden state, supervised by $\mathcal{L}_{\text{progress}} = \|p_t^{\text{pred}} - p_t^{\text{gt}}\|_2^2$. Since the deployed policy sees only its own $\hat{z}_g$ rather than the true goal frame, the effective target shifts to $p_t^{\text{anchor}} = p(z_t;\, z_s, \hat{z}_g)$; a consistency loss $\mathcal{L}_{\text{cons}} = \|p_t^{\text{pred}} - p_t^{\text{anchor}}\|_2^2$ ties the readout to this projection, keeping it calibrated

to the geometry downstream conditioning relies on. The Stage 1 objective is

$$\mathcal{L}_{\text{pretrain}} \;=\; \mathcal{L}_{\text{goal}} \;+\; \lambda_p\,\mathcal{L}_{\text{progress}} \;+\; \lambda_c\,\mathcal{L}_{\text{cons}}. \tag{3}$$

To expose the model to diverse partial-task prefixes that share the same endpoint, we augment each demonstration by sampling tuples (o_s, o_t, o_g) with $s \le t \le T$, where o_s varies while o_g remains fixed. This stage requires only demonstration videos paired with language instructions; no action labels are needed, making it compatible with large-scale robot video datasets.

3.2 Perceiving and Committing Progress

The progress reading p_{cur} locates the current observation along the start-to-goal direction in latent space, providing an absolute measure of task advancement. This localization is a necessary first step, but it leaves a critical gap: the same value of p_{cur} can arise from steady advancement, a sudden subtask transition, or recovery from a prior regression, yet each of these situations calls for a different motor response. Without knowing the recent execution rhythm or the intended near-future advancement, the policy cannot distinguish them and risks acting on an ambiguous signal. Absolute progress tells the policy *where the task stands*, but long-horizon execution also requires knowing *where it came from last step* and *how far the next step should go*. We therefore extend the progress representation with two relative quantities that together supply this missing temporal context.

Historical pace. The absolute reading alone cannot tell whether the policy is steadily advancing, has just broken through a plateau, or is recovering from a regression. To supply this missing context, the model receives the realized increment from the previous chunk, $\Delta p_{\text{prev}}(k) = p_{\text{cur}}(k) - p_{\text{cur}}(k{-}1)$, as an observed scalar that captures how much the last action actually changed the task state on the foresight-anchored scale. Its sign and magnitude reveal the local execution dynamics: a positive increment after a flat stretch signals resumed advancement, while a negative value flags temporary regression from a corrective motion. Because Δp_{prev} is computed directly from consecutive geometric projections rather than predicted, it stays on the same foresight-anchored scale at both training and inference, where it is fed as a scalar token to the next forward pass.

Forward commitment. Where p_{cur} and Δp_{prev} capture where the task stands and how it got there, the forward commitment $\Delta p_{\text{cur}}(k)$ looks ahead, declaring how much the next chunk should advance before the action executes. This transforms progress from a passive reading into an active signal that the model must subsequently deliver on. By staking an explicit claim about intended advancement before acting, the model binds its progress perception to the motor commands it generates, creating a verifiable link between plan and execution. A dedicated query token $q_{\Delta p}$ produces $\Delta p_{\text{cur}}^{\text{commit}}$ from the VLA hidden state, supervised by the expert's realized next increment:

$$\mathcal{L}_{\text{commit}} \;=\; \big\|\Delta p_{\text{cur}}^{\text{commit}} - (p_{k+1}^{\text{gt}} - p_k^{\text{gt}})\big\|_2^2. \tag{4}$$

The committed increment conditions the action decoder toward the declared advancement, and once the chunk completes, its gap with the realized increment exposes emerging failures without any additional model. Negative commitments are naturally accommodated, allowing the policy to plan corrective motions such as regrasping that temporarily reduce progress. Together, the absolute reading p_{cur} and the two relative increments Δp_{prev}, Δp_{cur} form a minimal past–present–future progress interface that the VLA reads and produces in a single forward pass, equipping the action decoder with a unified view of where execution has been, where it stands, and where it intends to go.

3.3 Progress-Conditioned Action Generation

The progress interface is useful only if it can shape what the policy does next. At each chunk, the VLA backbone produces a hidden representation $\mathbf{z}_k$ from observation history and language instruction, but this context carries no awareness of how far the task has progressed. We bridge this gap by projecting three progress signals through embedding MLPs and prepending them as conditioning tokens to the action DiT, so that perceived progress directly shapes the generated actions,

$$A_k \sim \pi_\theta\big(\,\cdot \mid \mathbf{z}_k,\ \phi_g(\hat{z}_g),\ \phi_p(p_{\text{cur}}),\ \phi_\Delta(\Delta p_{\text{cur}})\big), \tag{5}$$

where $A_k = (a_{kH}, \ldots, a_{(k+1)H-1})$ is the next action chunk generated via flow matching. The three tokens ground the generator with $\hat{z}_g$ anchoring it to the predicted task endpoint, p_{cur} situating the current chunk along the start-to-goal direction, and Δp_{cur} declaring how much this chunk should advance, ensuring that action generation reflects the policy's own progress pace.

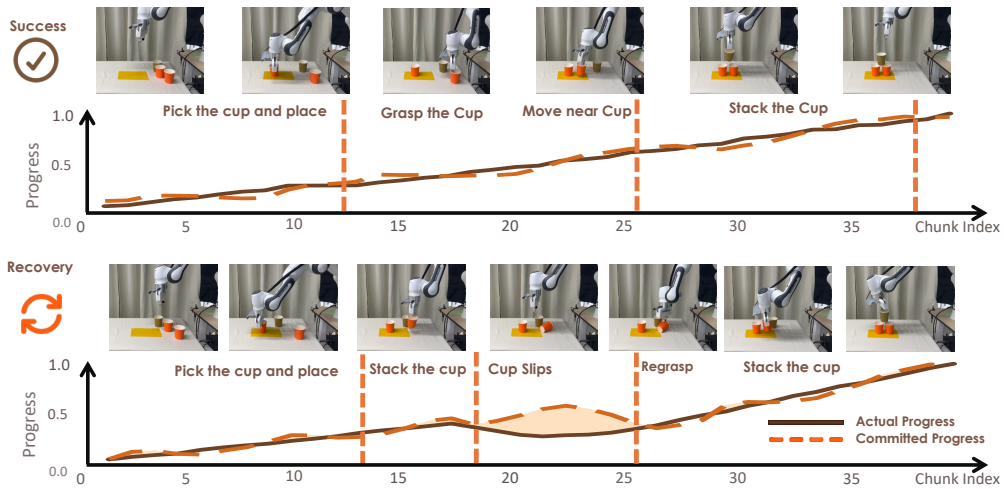

Figure 3: **Commitment verification examples.** *Top:* a successful trial where committed and realized progress track closely. *Bottom:* a recovery trial where the cup slips at k=18. The committed curve keeps rising while realized progress stalls, and the residual closes once the regrasp succeeds.

To preserve the geometric calibration established in Stage 1, the progress scalars are detached before entering the conditioning MLPs, preventing the flow-matching loss from back-propagating into the readout heads. The conditioning MLPs are zero-initialized, allowing the action DiT to converge under its standard objective before absorbing progress information. The Stage 2 objective $\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{flow}} + \lambda_{\Delta}\,\mathcal{L}_{\text{commit}} + \lambda_{\text{aux}}\,\mathcal{L}_{\text{pretrain}}$ combines flow matching with continued progress supervision (Eq. 3), keeping the readout heads calibrated as the backbone adapts. This read-then-condition design separates PACE from methods that treat progress as a parallel output alongside actions [29] or as guidance from an external classifier outside the policy forward pass [46]. In PACE, progress is first read out as a calibrated variable and then re-injected into action generation.

3.4 Commitment Verification at Inference

Every forward commitment is a testable prediction. The model declares $\Delta p_{\text{cur}}^{\text{commit}}(k)$ before chunk k executes, and the realized increment $\Delta p_{\text{cur}}^{\text{actual}}(k) = p_{k+1} - p_k$ becomes observable once the next observation arrives. Their difference defines the audit residual,

$$\Delta p_{\text{audit}}(k) = \Delta p_{\text{cur}}^{\text{actual}}(k) - \Delta p_{\text{cur}}^{\text{commit}}(k), \tag{6}$$

which measures the coherence between what the policy intended and what execution produced, requiring no additional learned component beyond the progress interface already in place. Under normal execution the policy maintains consistent commitments, intending to advance and delivering accordingly. When execution encounters an unexpected disturbance, such as a grasp slip or external perturbation, the commitment head continues to predict the advancement it expects, while the realized progress reflects what truly occurred. The residual thus grows precisely when the policy's intended trajectory diverges from the physical outcome, a reliable hallmark of emerging failure. Individual commitments carry prediction variance that can spike the per-step residual even during successful execution, making single-step detection prone to false alarms. We therefore accumulate $|\Delta p_{\text{audit}}|$ over a sliding window of W chunks and flag potential failures only when the windowed average exceeds a threshold τ, absorbing transient noise while preserving sustained deviations. Our framework PACE thus gains a form of execution self-awareness, recognizing when its own actions fail to produce the expected advancement without requiring any external evaluator.

4 Experiments

We conduct experiments across simulation benchmarks and real-world single-arm and bimanual platforms to investigate five properties of the progress interface. ***Progress effectiveness*** on simulation and real-world platforms reveals consistent gains on long-horizon tasks (Sections 4.1, 4.2). ***Progress internalization*** ablations show that conditioning on progress outperforms passive prediction and that the foresight anchor produces calibrated, phase-sensitive signals (Sections 4.3, 4.4). ***Progress consistency*** through the commit-realize gap exhibits strong correlation with execution failure, providing a lightweight self-monitoring signal at zero additional cost (Section 4.5). ***Progress robustness*** under distribution shift confirms the internalization advantage on generalization (Section 4.6). ***Progress***

Table 1: **Performance comparison on simulators.** Underlined scores mark the best results excluding PACE, daggered entries are finetuned from publicly released weights, and – denotes methods whose official weights are not released for finetuning on the corresponding benchmark.

Method	LIBERO (Franka)						SimplerEnv (WidowX)					
	Spatial	Object	Goal	Long	Avg.	Rank	Spoon	Carrot	Block	Eggplant	Avg.	Rank
Octo [14]	78.9	85.7	84.6	51.1	75.1	13	12.5	8.3	0.0	43.1	16.0	10
OpenVLA [20]	84.7	88.4	79.2	53.7	76.5	12	12.5	0.0	0.0	4.1	4.2	11
SpatialVLA [41]	88.2	89.9	78.6	55.5	78.1	11	16.7	25.0	29.2	100.0	42.7	7
CogACT [25]	96.8†	98.4†	89.8†	89.2†	93.6†	7	71.7	50.8	15.0	67.5	51.3	4
GR00T N1 [3]	94.4	97.6	93.0	90.6	93.9	6	59.5	43.0	45.0	48.5	49.0	5
π_0 [5]	96.8	98.8	95.8	85.2	94.2	5	49.1	20.0	36.6	82.5	47.1	6
UniVLA [7]	96.5	96.8	95.6	92.0	95.2	3	41.6	29.2	4.2	50.0	31.3	9
InternVLA-M1 [10]	98.0	99.0	93.8	92.6	95.9	2	87.5	67.9	31.3	100.0	71.7	2
MolmoAct [23]	87.0	95.4	87.6	77.2	86.6	9	70.3	50.0	45.3	60.0	51.4	3
ProgressVLA [46]	88.2	96.4	87.2	66.2	84.5	10	–	–	–	–	–	–
SPR [13]	93.2	95.4	93.2	85.4	91.8	8	33.3†	16.7†	25.0†	70.8†	36.5†	8
PALM [29]	95.2	96.7	94.3	91.8	94.5	4	–	–	–	–	–	–
PACE (Ours)	**99.0**	**99.8**	**97.8**	**98.0**	**98.7**	**1**	**87.5**	**79.2**	**33.3**	**100.0**	**75.0**	**1**

source compatibility shows that the interface works with alternative progress sources, whose quality modulates the gain (Section 4.7).

4.1 Simulation Benchmarks

We evaluate on two complementary simulation suites. LIBERO [28] provides language-conditioned manipulation across spatial, object, goal, and long-horizon task variations on Franka. SimplerEnv [26] probes cross-domain robustness through visual appearance shifts on short-horizon WidowX tasks.

LIBERO. We fine-tune a single checkpoint jointly across all four suites and evaluate over 500 trials with randomized initial states. As shown in Table 1, PACE achieves 98.7% average success rate, ranking first across all four suites. Unlike π_0 and GR00T N1 with no explicit progress sense, PACE equips the action decoder with a calibrated progress signal to guide the execution. PALM co-predicts progress but does not channel it back as an explicit condition, and PACE highlights the value of actively conditioning actions on progress. ProgressVLA [46] utilizes progress to guide diffusion sampling, but through a decoupled standalone classifier, and PACE's advantage especially on the Long suite further confirms the effectiveness of coupling progress with the VLA backbone.

SimplerEnv. We evaluate on the WidowX suite in SimplerEnv across four tasks under visual appearance shifts. PACE achieves 75.0% average success, substantially outperforming the competitive baselines including π_0 and GR00T N1, demonstrating that progress awareness provides effective guidance for manipulation under visual variation while preserving the underlying task advancement pace. The performance advantage over other progress-aware approaches further validates that actively internalizing progress into the action generation loop is key to unlocking its full potential.

4.2 Real-World Evaluation

Experiment settings. We deploy PACE on two physical platforms covering four tasks that each stress a different aspect of progress-aware control (Figure 4). Single-arm experiments use a Franka Research 3 with two RealSense D435 cameras on *Table Sorting*, which organizes scattered objects into designated categories across sequential subtasks, and *Cup Stacking*, which builds a two-layer pyramid from three cups requiring sustained alignment. Bimanual experiments use a pair of ARX Lift2S arms with two RGB cameras on *Towel Folding*, which grasps and folds a deformable cloth through multiple folding stages, and *Block Disassembly*, which separates interlocking blocks and sorts them into color-coded compartments. All models run on a single NVIDIA RTX 5090 GPU. We collect 60 teleoperated demonstrations per task and evaluate over 50 rollouts per method.

Results. As shown in Table 2, PACE achieves 74.0% average success across all four tasks, substantially outperforming all baselines. Progress-aware methods such as SPR [13] and PACE consistently outperform direct action prediction models including OpenVLA, SpatialVLA, and UniVLA, confirming the importance of progress awareness for long-horizon manipulation. Among progress-aware approaches, SPR exposes spatial milestones as intermediate planning targets but does not feed progress back to condition action generation, and the gap to PACE

Table 2: **Real-world performance.** Success rate (%) on single-arm and bimanual tasks.

Policy	Single-arm		Bimanual		Avg.
	Table Sorting	Cup Stacking	Towel Folding	Block Disasm.	
OpenVLA [20]	22	12	6	4	11
SpatialVLA [41]	30	20	8	6	16
UniVLA [7]	46	34	18	10	27
SPR [13]	50	38	22	14	31
π_0 [5]	62	52	38	28	45
GR00T N1 [3]	66	56	42	32	49
PACE (Ours)	**86**	**78**	**68**	**64**	**74**

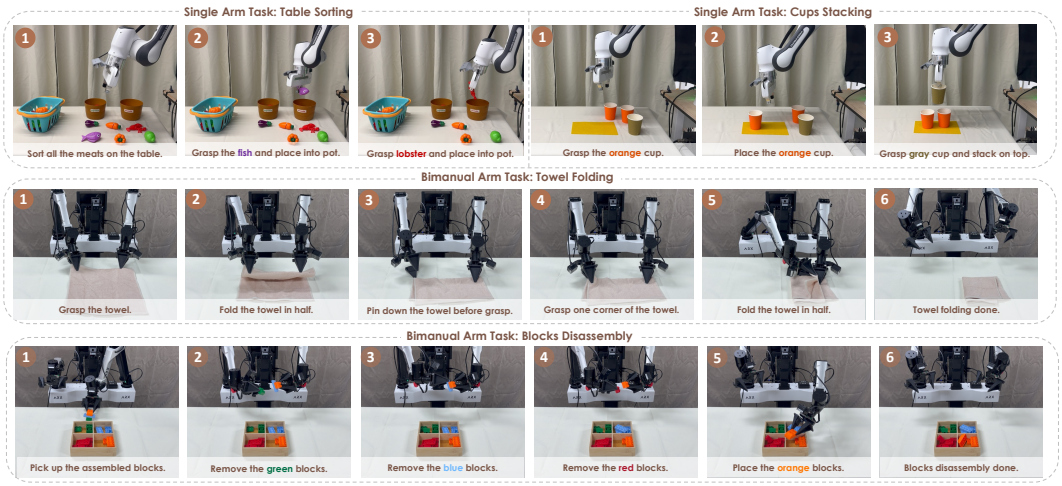

Figure 4: **Real-world robot experiments.** We evaluate PACE on two real-world platforms including single-arm tasks (Table Sorting and Cup Stacking) and bimanual tasks (Towel Folding and Block Disassembly), demonstrating robust performance across diverse long-horizon scenarios.

highlights the value of actively routing progress through the policy forward pass. Compared to π_0 and GR00T N1, PACE achieves consistent gains by equipping the action decoder with a calibrated progress signal, with the advantage most pronounced on bimanual tasks where temporary regression from regrasping or alignment plateaus compounds errors in policies that lack an explicit sense of intended advancement and cannot distinguish planned retreat from genuine stalling.

4.3 The Effectiveness of Progress Internalization

To investigate whether the progress interface is genuinely internalized into the action loop, we ablate PACE by removing one component at a time while keeping the backbone, training data, and action decoder identical across all variants (Figure 5). The most revealing variant is *co-decode*, which still perceives and predicts progress through the same readout heads but never feeds it into the action decoder. This single change costs 4.8% in simulation and over 17% on real robots, even though the predicted signal is identical, confirming that progress merely co-decoded alongside actions provides substantially weaker guidance than progress that explicitly shapes action generation. Removing the forward commitment while retaining perception and history costs a further 6.5% on real tasks, and removing history produces an even larger drop of 8.0%, indicating that temporal context from both directions contributes and that knowing recent execution *pace* matters. These ablations confirm that a VLA benefits from internalizing a complete sense of its own execution pace: *where* it stands, *how fast* it has been advancing, and *what the next chunk should accomplish*.

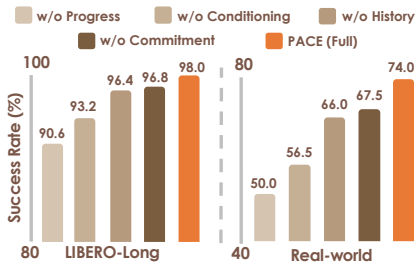

Figure 5: **Ablation of the progress interface.** Isolating each component separately to verify the effectiveness of progress genuinely internalized into action generation.

4.4 The Accuracy of Anchoring Progress and Commitment Prediction

We further investigate whether the predicted progress signals are themselves accurate. PACE produces two quantities at each chunk, an *absolute* progress reading p_{cur} that locates execution on the start-to-goal axis and a *relative* commitment Δp_{cur} that predicts how much the next chunk should advance, both evaluated against geometric ground truth from frozen DINOv2 projections on LIBERO-Long rollouts (Table 3). Without the foresight anchor $\hat{z}_g$, the model must regress progress from a generic hidden state that lacks a task-specific reference, inflating p_{cur} MAE from 0.041 to 0.173,

Table 3: **Progress and commitment accuracy** on LIBERO-Long rollouts.

Variant	p_{cur}	Δp_{cur}
PACE (full)	**0.041**	**0.021**
w/o foresight anchor	0.173	0.068
w/o Stage 1 pretrain	0.092	0.044
w/o $\mathcal{L}_{\text{cons}}$	0.063	0.029
w/o history Δp_{prev}	0.041	0.038

confirming that the anchor is central to stabilizing absolute progress estimation. Skipping Stage 1 pretraining or removing $\mathcal{L}_{\text{cons}}$ produces intermediate degradation, indicating that both the anchor geometry and its alignment with the scalar readout contribute. Removing the previous realized increment Δp_{prev} does not affect p_{cur} but nearly doubles Δp_{cur} MAE from 0.021 to 0.038, reflecting that predicting how far the next chunk should go naturally relies on knowing how far the last one did. These results confirm that the progress interface not only improves task success but produces well-calibrated signals that the action decoder can reliably condition on.

4.5 The Consistency of Committed and Realized Progress

We further evaluate whether the predicted absolute progress p_{cur} and the forward commitment Δp_{cur} can serve as failure detectors at different granularities. At the trajectory level, a final p_{cur} that falls well short of 1 indicates task incompletion. At the chunk level, we accumulate the deviation between committed and realized increments over a sliding window of $W=3$ chunks, so that a sustained gap signals not only *that* execution has derailed but *when*. We compare against three heuristic baselines on human-annotated rollouts from LIBERO and real-world tasks (Table 4). These heuristics read only the realized progress curve: the best progress reached so far, the share of chunks that move backwards, and the largest drop from the running peak (Appendix D.2). The accumulated audit residual achieves 0.92 trajectory-level AUROC and 0.85 chunk-level AUROC, outperforming all baselines at both granularities. The two learned reward models, RoboReward [24] and Robometer [27], reach 0.85 and 0.87 trajectory-level AUROC but each needs a separate VLM pass per frame, adding 65 to 70 ms per chunk against 2 ms for the audit (Appendix D.2). The advantage traces to how the commitment handles non-monotonic progress: monotonicity-based detectors flag legitimate recovery motions such as regrasping as failures, while the audit residual stays small whenever the policy *expected* the temporary retreat and committed accordingly. Figure 3 illustrates this on a real-world cup-stacking rollout, where committed and realized curves track tightly, separate after an unexpected grasp slip, and re-align once the regrasp succeeds. We scope this signal as a detection and interpretability tool rather than a closed-loop correction trigger. The diagnostic comes at zero additional cost, requiring no extra model and no weight update at inference. Learned reward models can score trajectories post hoc from any VLA, whereas the audit is available only to policies that commit to their own progress, in exchange for real-time monitoring calibrated to the policy's own progress scale (Appendix D.2).

Table 4: **Failure detection.** Trajectory-level and chunk-level AUROC ↑ on human-annotated rollouts.

Detector	Latency	Traj.	Chunk
Low p_{cur}	–	0.72	0.65
Neg. realized Δp	–	0.78	0.71
Monotonicity violation	–	0.80	0.74
RoboReward [24]	+65 ms	0.85	0.78
Robometer [27]	+70 ms	0.87	0.81
Accum. $\lvert\Delta p_{\text{audit}}\rvert$ (ours)	+2 ms	**0.92**	**0.85**

4.6 The Robustness of Progress Internalization

We evaluate whether the progress interface generalizes beyond the training configuration by testing under three categories of distribution shift on the real-world platform (Figure 6): *Visual* perturbation with randomized lighting and backgrounds, *Object* perturbation with unseen instances not present in training, and *Task* perturbation with new compositional instructions that alter goal structure. PACE averages 54% success across all three shift types, while π_0, GR00T N1, and *w/o Progress* cluster between 21% and 29%. The gap is most pronounced on Task perturbation, where PACE retains 46% while GR00T N1 drops to 8%, suggesting that a calibrated awareness of explicit progress helps the policy coordinate across novel subtask sequences that it has never rehearsed. On Object perturbation PACE keeps 54% against 42% for *w/o Commitment* and at most 28% for the other baselines. Comparing PACE against *w/o Commitment* further reveals that the forward commitment grows more valuable out of distribution. The two differ by 6.5% in-distribution (Figure 5) but by 11% on average under shift, reaching 10% on Visual perturbation alone. This widening reflects a structural advantage of the full interface: when visual details change, a single absolute progress reading can become ambiguous, but the combination of foresight-anchored estimation, historical context, and forward commitment produces a richer temporal scaffold that remains

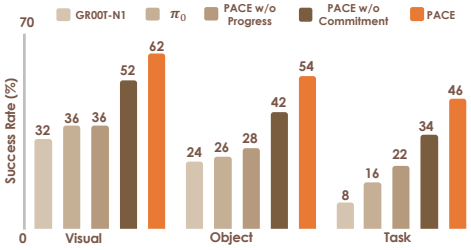

Figure 6: **Real-world generalization comparisons.** The gap between PACE and alternatives widens under perturbation with the benefit of progress internalization.

interpretable even when surface-level observations drift from training. Each shift category is evaluated with 50 rollouts per method on both the single-arm and bimanual platforms (Appendix C.2).

4.7 Progress Source Compatibility

The default progress target comes from frozen DINOv2-B geometry, and we test whether the interface depends on this choice. Table 5 swaps the progress source while keeping the policy architecture, training data, and action decoder fixed. Every source produces a working policy, with LIBERO-Long success between 92.0% and 98.0%. Normalized time t/T, which treats every demonstration as uniformly paced, sets the lower end. The two learned reward models tighten calibration (MAE 0.048 to 0.052) but reach only 93.0% to 93.4% success, while DINOv2-B geometry attains the highest rank correlation, the lowest MAE, and the highest success without dense labels or any additionally trained model. Higher rank correlation generally coincides with higher success, so the quality of the source modulates the gain while the policy-side mechanism of predicting and conditioning on progress stays unchanged. Within the DINOv2 family (lower block), all four encoder scales from ViT-S to ViT-g exceed 0.91 rank correlation and 94% success, with ViT-B the strongest (Appendix D.3).

Table 5: **Progress source compatibility.** Rank correlation and p_{cur} MAE on expert demonstrations, SR on LIBERO-Long.

Source	Labels	Rank	SR	MAE
Normalized time t/T	no	0.72	92.0	0.121
CLIP geom.	no	0.78	92.4	0.089
Qwen3-VL hidden geom.	no	0.82	92.8	0.074
Robometer [27]	yes	0.87	93.0	0.052
RoboReward [24]	yes	0.88	93.4	0.048
DINOv2-S geom.	no	0.91	94.2	0.058
DINOv2-B geom. (ours)	no	**0.97**	**98.0**	**0.041**
DINOv2-L geom.	no	0.93	95.2	0.050
DINOv2-G geom.	no	0.94	96.0	0.047

5 Conclusion

Current VLA policies generate actions with no internal model of how their own execution unfolds over time. PACE addresses this fundamental gap by internalizing task progress as a predictive interface bridging perception and action. A *foresight anchor* predicted from observation and instruction grounds progress as a geometric projection, and a compact past–present–future readout conditions the action decoder so that progress actively shapes execution rather than running alongside it. The same forward commitment that drives action generation doubles as a built-in consistency check at inference, detecting execution failures without any external model. Experiments across LIBERO, SimplerEnv, and real-world single-arm and bimanual platforms demonstrate that PACE effectively achieves 98.7% on LIBERO and 74% on real-world manipulation, with the advantage most pronounced on long-horizon tasks where temporal grounding is essential. The progress signals are well-calibrated, the internalization advantage remains robust under distribution shift, and the commit-realize progress gap provides reliable failure detection at zero additional cost. More broadly, the *perceive–commit–condition–verify* loop establishes a promising design principle for equipping VLA policies with temporal self-awareness. By requiring the model to predict its own advancement before acting and verify the outcome afterward, PACE moves VLA design beyond purely reactive control toward policies that track their own advancement progress. The commit-then-verify principle naturally extends to other temporal quantities such as subgoal completion or contact timing, pointing toward richer forms of self-monitoring in future robot policies. We believe that progress internalization, equipping the policy with a calibrated sense of its own execution *pace*, is an intuitive yet effective paradigm for enabling robust control over increasingly complex long-horizon tasks.

Limitations and future work. The foresight anchor predicts a single terminal state, so the progress axis requires distinct start and goal embeddings. Tasks that return to their initial configuration, or whose goal changes during execution, call for intermediate anchors that reset the progress scale over each segment. The default visual source is most informative when advancement produces visible state change, and tactile or force-based progress is a natural extension for contact-rich manipulation. The commitment audit detects failures but does not yet trigger recovery, and conditioning a replanning module [52] on the audit residual would close this loop (Appendix F).

Acknowledgments and Disclosure of Funding

Fei Ni was supported by a Marie Skłodowska-Curie Actions Postdoctoral Fellowship. Jiankang Deng was supported by the NVIDIA Academic Grant. The authors acknowledge the computing resources provided by Isambard-AI, part of the National AI Research Resource (AIRR).

References

[1] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[2] Zechen Bai, Chen Gao, and Mike Zheng Shou. EVOLVE-VLA: Test-time training from environment feedback for vision-language-action models. *arXiv preprint arXiv:2512.14666*, 2025.

[3] 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.

[4] Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pre-trained image-editing diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

[5] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems (RSS)*, 2023.

[7] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. UniVLA: Learning to act anywhere with task-centric latent actions. In *Robotics: Science and Systems (RSS)*, 2025.

[8] Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Fred Shentu, and Philipp Wu. SARM: Stage-aware reward modeling for long horizon robot manipulation. In *International Conference on Learning Representations (ICLR)*, 2026.

[9] Shirui Chen, Cole Harrison, Ying-Chun Lee, Angela Jin Yang, Zhongzheng Ren, Lillian J Ratliff, Jiafei Duan, Dieter Fox, and Ranjay Krishna. TOPReward: Token probabilities as hidden zero-shot rewards for robotics. *arXiv preprint arXiv:2602.19313*, 2026.

[10] Xinyi Chen, Yilun Chen, Yanwei Fu, Ning Gao, Jiaya Jia, Weiyang Jin, Hao Li, Yao Mu, Jiangmiao Pang, Yu Qiao, Yang Tian, Bin Wang, Bolun Wang, Fangjing Wang, Hanqing Wang, Tai Wang, Ziqin Wang, Xueyuan Wei, Chao Wu, Shuai Yang, Jinhui Ye, Junqiu Yu, Jia Zeng, Jingjing Zhang, Jinyu Zhang, Shi Zhang, Feng Zheng, Bowen Zhou, and Yangkun Zhu. InternVLA-M1: A spatially guided vision-language-action framework for generalist robot policy. *arXiv preprint arXiv:2510.13778*, 2025.

[11] Xiong-Hui Chen, Junyin Ye, Hang Zhao, Yi-Chen Li, Xu-Hui Liu, Haoran Shi, Yu-Yan Xu, Zhihao Ye, Si-Hang Yang, Yang Yu, Anqi Huang, Kai Xu, and Zongzhang Zhang. Deep demonstration tracing: Learning generalizable imitator policy for runtime imitation from a single demonstration. In *International Conference on Machine Learning (ICML)*, volume 235, pages 7586–7620. PMLR, 2024.

[12] Mingtong Dai, Lingbo Liu, Yongjie Bai, Yang Liu, Zhouxia Wang, Rui Su, Chunjie Chen, Liang Lin, and Xinyu Wu. RoVer: Robot reward model as test-time verifier for vision-language-action model. *arXiv preprint arXiv:2510.10975*, 2025.

[13] 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.

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

[15] Yanjiang Guo, Tony Lee, Lucy Xiaoyang Shi, Jianyu Chen, Percy Liang, and Chelsea Finn. VLAW: Iterative co-improvement of vision-language-action policy and world model. In *International Conference on Machine Learning (ICML)*, volume 306, pages 37998–38011. PMLR, 2026.

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[17] Kuo-Han Hung, Pang-Chi Lo, Jia-Fong Yeh, Han-Yuan Hsu, Yi-Ting Chen, and Winston H. Hsu. VICtoR: Learning hierarchical vision-instruction correlation rewards for long-horizon manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

[18] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-Z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning (CoRL)*, pages 991–1002. PMLR, 2022.

[19] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024.

[20] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, pages 2679–2713. PMLR, 2024.

[21] Longxin Kou, Fei Ni, Yan Zheng, Jinyi Liu, Yifu Yuan, Zibin Dong, and Jianye Hao. KISA: A unified keyframe identifier and skill annotator for long-horizon robotics demonstrations. In *International Conference on Machine Learning (ICML)*, volume 235, pages 25441–25474. PMLR, 2024.

[22] Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. In *Conference on Robot Learning (CoRL)*, pages 3200–3217. PMLR, 2025.

[23] Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli Vanderbilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct: Action reasoning models that can reason in space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2585–2592, 2026.

[24] 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.

[25] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. CogACT: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[26] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning (CoRL)*, pages 3705–3728. PMLR, 2024.

[27] 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. In *Robotics: Science and Systems (RSS)*, 2026.

[28] 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 (NeurIPS), Datasets and Benchmarks Track*, 2023.

[29] Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, Fangqiang Ding, and Ismini Lourentzou. PALM: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 28096–28110, 2026.

[30] Yanzuo Lu, Ronglai Zuo, and Jiankang Deng. RAVEN: Real-time autoregressive video extrapolation with consistency-model GRPO. *arXiv preprint arXiv:2605.15190*, 2026.

[31] 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 (ICML)*, pages 23301–23320. PMLR, 2023.

[32] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations (ICLR)*, 2023.

[33] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3M: A universal visual representation for robot manipulation. In *Conference on Robot Learning (CoRL)*, pages 892–909. PMLR, 2022.

[34] Fei Ni, Jianye Hao, Yao Mu, Yifu Yuan, Yan Zheng, Bin Wang, and Zhixuan Liang. MetaDiffuser: Diffusion model as conditional planner for offline meta-RL. In *International Conference on Machine Learning (ICML)*, volume 202, pages 26087–26105. PMLR, 2023.

[35] Fei Ni, Jianye Hao, Shiguang Wu, Longxin Kou, Jiashun Liu, Yan Zheng, Bin Wang, and Yuzheng Zhuang. Generate subgoal images before act: Unlocking the chain-of-thought reasoning in diffusion model for robot manipulation with multimodal prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13991–14000, 2024.

[36] Fei Ni, Jianye Hao, Shiguang Wu, Longxin Kou, Yifu Yuan, Zibin Dong, Jinyi Liu, Mingzhi Li, Yuzheng Zhuang, and Yan Zheng. PERIA: Perceive, reason, imagine, act via holistic language and vision planning for manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 17541–17571, 2024.

[37] Fei Ni, Zhuo Chen, Yifu Yuan, Zibin Dong, Xianze Yao, Shan Luo, Jianye Hao, Jiankang Deng, and Stefanos Zafeiriou. SemanticVLA: Towards semantic reasoning over action memorization via synergistic explicit trace and latent action planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12237–12247, 2026.

[38] Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903, 2024.

[39] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024.

[40] Guanren Qiao, Sixu Lin, Ronglai Zuo, Zhizheng Wu, Kui Jia, and Guiliang Liu. SignBot: Learning human-to-humanoid sign language interaction. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 20471–20478, 2026.

[41] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Jiayuan Gu, Zhigang Wang, Yan Ding, Bin Zhao, Dong Wang, and Xuelong Li. SpatialVLA: Exploring spatial representations for visual-language-action models. In *Robotics: Science and Systems (RSS)*, 2025.

[42] 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 *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021.

[43] Mohit Shridhar, Yat Long Lo, and Stephen James. Generative image as action models. In *Conference on Robot Learning (CoRL)*, pages 2429–2455. PMLR, 2024.

[44] 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.

[45] Homer Rich Walke, Kevin Black, Tony Z. Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, Abraham Lee, Kuan Fang, Chelsea Finn, and Sergey Levine. BridgeData V2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, pages 1723–1736. PMLR, 2023.

[46] 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.

[47] 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.

[48] Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Chenyang Zhao, 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. *Expert Systems with Applications*, 316: 131742, 2026.

[49] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[50] Jia-Fong Yeh, Chi-Ming Chung, Hung-Ting Su, Yi-Ting Chen, and Winston H. Hsu. Stage conscious attention network (SCAN): A demonstration-conditioned policy for few-shot imitation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, pages 8866–8873, 2022.

[51] Jia-Fong Yeh, Kuo-Han Hung, Pang-Chi Lo, Chi-Ming Chung, Tsung-Han Wu, Hung-Ting Su, Yi-Ting Chen, and Winston H. Hsu. AED: Adaptable error detection for few-shot imitation policy. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 136805–136836, 2024.

[52] Yifu Yuan, Yaoting Huang, Xianze Yao, Yutong Li, Shuoheng Zhang, Linqi Han, Pengyi Li, Jiangeng Sun, Wenting Jia, Zhao Zhang, et al. Embodied-R1.5: Evolving physical intelligence via embodied foundation models. *arXiv preprint arXiv:2606.11324*, 2026.

[53] 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.

[54] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11975–11986, 2023.

[55] Yuelin Zhang, Sijie Cheng, Chen Li, Zongzhao Li, Yuxin Huang, Yang Liu, and Wenbing Huang. Recurrent reasoning with vision-language models for estimating long-horizon embodied task progress. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 41150–41159, 2026.

[56] Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. 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 (CVPR)*, pages 1702–1713, 2025.

[57] Ronglai Zuo, Rolandos Alexandros Potamias, Qi Sun, Evangelos Ververas, Jiankang Deng, and Stefanos Zafeiriou. MaDiS: Taming masked diffusion language models for sign language generation. *arXiv preprint arXiv:2601.19577*, 2026.

A Method Details

A.1 Progress Target Construction

A growing body of work has shown that diverse visual representations, from self-supervised encoders such as DINOv2 [39], CLIP [42], and SigLIP [54] to VLM-based reward models such as RoboReward [24] and Robometer [27], encode task-advancement structure in their embedding geometry: frames closer to task completion tend to cluster along a recognizable direction in the latent space. The progress targets used throughout PACE are designed to read out this latent structure as a single scalar, through a geometric projection that is agnostic to which encoder provides the embeddings. Any representation whose geometry correlates with semantic task advancement can serve as the underlying space, and the same formula applies without modification. Our default implementation uses frozen DINOv2-Base (ViT-B/14) CLS embeddings for their strong spatial-semantic correspondence and the practical advantage of requiring no additional labels or model training. Section D.3 verifies that the interface accommodates alternative sources spanning self-supervised encoders of varying scale, VLM hidden states, and trained reward models (Table 9).

Geometric progress. Given a demonstration trajectory $\tau = (o_0, o_1, \ldots, o_T)$ with language instruction ℓ, we extract frame-level embeddings $z_t = f_\psi(o_t) \in \mathbb{R}^d$ from a frozen encoder f_ψ. The progress of any intermediate frame o_t relative to a start–goal pair (o_s, o_g) is defined as (reproducing the main-paper definition)

$$p(z_t;\, z_s, z_g) \;=\; \frac{\langle\, z_t - z_s,\; z_g - z_s \,\rangle}{\|z_g - z_s\|_2^2}\,, \quad z_i = f_\psi(o_i)\,, \tag{7}$$

yielding $p = 0$ at the start and $p = 1$ at the goal by construction. Only the two endpoints are pinned: an intermediate frame that overshoots the goal or retreats behind the start projects outside $[0, 1]$, so the training target is $p_t^{\mathrm{gt}} = \mathrm{clamp}\big(p(z_t;\, z_s, z_g),\, 0,\, 1\big)$. The formula measures what fraction of the total start-to-goal displacement in representation space has been covered, reducing a high-dimensional similarity landscape to a single scalar that tracks task advancement regardless of trajectory duration or execution pace. Increment targets follow directly: $\Delta p^{\mathrm{gt}}_{\mathrm{prev}}(k) = p^{\mathrm{gt}}_k - p^{\mathrm{gt}}_{k-1}$ and $\Delta p^{\mathrm{gt}}_{\mathrm{next}}(k) = p^{\mathrm{gt}}_{k+1} - p^{\mathrm{gt}}_k$, where k indexes chunk boundaries.

Why proportional projection. The choice of Eq. (7) over two natural alternatives reflects a deliberate geometric trade-off. Any displacement $z_t - z_s$ in representation space decomposes into an *on-axis* component along the start-to-goal direction, reflecting task advancement, and an *off-axis* residual orthogonal to it, capturing progress-irrelevant variation such as lighting change, camera jitter, or background shift. Euclidean distance to the goal, $\|z_t - z_g\|$, conflates these two components: a frame that drifts far off-axis but has barely advanced would appear distant from the goal, producing a spuriously low progress reading. Cosine similarity between $z_t - z_s$ and $z_g - z_s$ retains only the angular relationship and discards how far along the axis the trajectory has actually traveled, collapsing frames at 10% and 90% advancement to similar scores when both point in roughly the same direction. The proportional projection isolates the on-axis fraction, making it sensitive to both direction and magnitude while remaining invariant to off-axis displacement. Because this decomposition operates on the vector geometry rather than on encoder-specific properties, the same formula transfers across representation spaces with minimal degradation (Section D.3). Across 12 frozen encoders and two real-robot tasks, the projection tracks task progress better than native goal similarity in 22 of 24 encoder–task comparisons, so the gain comes from the read-out as well as from the representation.

What the projection assumes. Eq. (7) pins only the two endpoints, so it fixes the scale at 0 and 1 without constraining the path between them. Increments along the single axis add up, and any route that reaches the goal reads 1 whatever order its stages follow. On 19 rollouts of order-interchangeable two-object tasks, the two stage displacements sum to 0.944 to 1.000 of the axis whichever object is handled first. The definition does not require monotonic advancement. On the 379 LIBERO-Long expert demonstrations, 39.9% of frame-to-frame increments (stride 4) are negative, while the target still correlates with time (Spearman $\rho(t, p) = 0.72$). Normalized time t/T is the only strictly monotone source in Table 9 and also the weakest (92.0% vs. 98.0% LIBERO-Long success). PACE does not rely on monotonicity, since realized increments enter the decoder signed and the forward commitment is a signed target. The one working assumption is that the start–goal pair identifies

Algorithm 1 Foresight-anchored progress-conditioned training.

Require: Demonstration trajectory τ, instruction ℓ, frozen encoder f_ψ

Sample chunk boundary k and effective-start frame o_s ($s \le k$, Stage 1 only).
Extract z_s, z_k, z_T from f_ψ; compute progress targets via Eq. (7).
Compute $p_k^{\mathrm{gt}}, \Delta p_{\mathrm{prev}}^{\mathrm{gt}}(k), \Delta p_{\mathrm{next}}^{\mathrm{gt}}(k)$.
Run VLA with observation history $(o_{k-H}, \ldots, o_k)$, instruction ℓ, scalar token $\Delta p_{\mathrm{prev}}(k)$, and query tokens $q_{\mathrm{goal}}, q_p, q_{\Delta p}$.
Predict terminal anchor $\hat{z}_g$, current progress $p_{\mathrm{cur}}^{\mathrm{pred}}$, and commitment $\Delta p_{\mathrm{cur}}^{\mathrm{commit}}$.
Stage 1: $\mathcal{L}_{\mathrm{pretrain}} = \mathcal{L}_{\mathrm{goal}} + \lambda_p \mathcal{L}_{\mathrm{progress}} + \lambda_c \mathcal{L}_{\mathrm{cons}}$. Update query heads only.
Stage 2: Condition action DiT on $\phi_g(\hat{z}_g), \phi_p(p_{\mathrm{cur}}), \phi_\Delta(\Delta p_{\mathrm{cur}})$ (all detached).
$\mathcal{L}_{\mathrm{stage2}} = \mathcal{L}_{\mathrm{flow}} + \lambda_\Delta \mathcal{L}_{\mathrm{commit}} + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{pretrain}}$. Update DiT and backbone.

Algorithm 2 Progress-conditioned inference with commitment audit.

Require: Trained VLA π_θ, frozen encoder f_ψ, audit threshold τ, window size W=3

Initialize audit buffer $\mathcal{B} \leftarrow \emptyset$; observe o_0; set $p_{\mathrm{cur}}(0) \leftarrow 0$, $\Delta p_{\mathrm{prev}}(0) \leftarrow 0$.
for chunk $k = 0, 1, 2, \ldots$ **do**
  Run VLA forward pass $\rightarrow p_{\mathrm{cur}}(k)$, $\Delta p_{\mathrm{cur}}^{\mathrm{commit}}(k)$.
  Generate action chunk $A_k \sim \pi_\theta(\cdot \mid z_k, \phi_g, \phi_p, \phi_\Delta)$.
  Execute A_k; observe o_{k+1}.
  Compute $p_{\mathrm{cur}}(k{+}1)$ from $f_\psi(o_{k+1})$ via geometric projection.
  $\Delta p^{\mathrm{actual}}(k) \leftarrow p_{\mathrm{cur}}(k{+}1) - p_{\mathrm{cur}}(k)$.
  $\Delta p_{\mathrm{audit}}(k) \leftarrow \Delta p^{\mathrm{actual}}(k) - \Delta p_{\mathrm{cur}}^{\mathrm{commit}}(k)$.
  Append $|\Delta p_{\mathrm{audit}}(k)|$ to $\mathcal{B}$; compute windowed mean over last W entries.
  if windowed mean $> \tau$ **then**
    Flag potential execution failure; optionally trigger replanning.
  end if
  $\Delta p_{\mathrm{prev}}(k{+}1) \leftarrow \Delta p^{\mathrm{actual}}(k)$.
end for

the task, with $\|z_g - z_s\|$ bounded away from zero. On the same demonstrations, the task-mean axis length on unit-normalized DINOv2-B embeddings ranges from 0.610 to 0.895, and the smallest single-episode length is 0.510. Tasks whose initial and terminal configurations coincide fall outside this assumption, as noted in the limitations (Section 5). On Cup Stack-and-Restore, a diagnostic task built for this case in which three cups are stacked into a pyramid, dismantled and returned to their marked initial poses, every endpoint-conditioned source (our geometric read-out, LIV [31], RoboReward-8B [24]) stays at 12 to 16% success, the level of normalized time; ordered keyframes supplied by hand raise PACE to 34%, and on LIBERO the same re-anchoring lifts per-task correlation by 0.25 to 0.48 for every visual source we tested.

A.2 Training Recipe

Training proceeds in two stages that mirror the structure of the progress interface itself. Stage 1 trains the foresight anchor and progress readout heads on large-scale demonstration videos *without any action labels*, building the model's ability to perceive where a task stands and where it is headed. Stage 2 introduces the action decoder and teaches the model to condition its motor output on the progress signals learned in Stage 1. The two stages share the same progress target construction (Section A.1) but differ in data, objectives, and which parameters are updated. Algorithm 1 and Algorithm 2 specify the two core procedures in pseudocode.

Stage 1: Progress perception pretraining. The goal of Stage 1 is to teach the VLA where a task is headed and how far execution has come, before it ever sees an action label. The model learns to predict a foresight anchor from observation and language, providing a stable reference for what task completion looks like, and to localize the current observation along the resulting start-to-goal axis, producing a calibrated progress reading. The anchor is not an end in itself but the geometric scaffold that makes progress well-defined: without knowing where the task is going, the model cannot judge how far it has come. Demonstration-conditioned policies such as SCAN [50] and DDT [11] share

the view that a long-horizon policy should know where it stands, but they locate execution against an expert trajectory supplied at test time; the foresight anchor is predicted by the policy itself, so no demonstration is needed at inference and progress is a continuous scalar rather than an index into a trajectory.

We pretrain on a curated mixture of approximately 260K demonstration episodes spanning Franka, Google Robot, and WidowX platforms, drawn from DROID, Fractal, Bridge V2, BC-Z, and MolmoAct. Since no action decoding occurs at this stage, each episode needs only observation frames and a language instruction. All episodes are automatically annotated with geometric progress targets via frozen DINOv2-Base projection before training begins, providing simultaneous supervision for both the foresight anchor and the progress readout without any human labels. The full data composition and annotation pipeline are described in Section A.3.

The backbone processes observation and language through the standard VLA forward pass, with $K{+}2$ appended query tokens (q_{goal}, q_p, $q_{\Delta p}$) producing the foresight anchor $\hat{z}_g$, the progress readout $p_{\text{cur}}^{\text{pred}}$, and the forward commitment $\Delta p_{\text{cur}}^{\text{commit}}$. The Stage 1 objective combines three losses (Eq. 3 in the main paper): $\mathcal{L}_{\text{goal}}$ supervises the anchor against the frozen embedding of the actual terminal frame, $\mathcal{L}_{\text{progress}}$ supervises the progress readout against the geometric target from Eq. (7), and $\mathcal{L}_{\text{cons}}$ keeps the scalar readout calibrated to the anchor-implied projection rather than drifting into an unstructured regression target. The backbone is kept frozen throughout Stage 1; only the $K{+}2$ query token embeddings and their projection heads are updated.

Effective-start augmentation. Stage 1 must train the foresight anchor to generalize across starting configurations it has never seen. When the start frame o_s is always o_0, every training sample anchors progress to a single initial state per trajectory, and the model overfits to that viewpoint and object layout. We augment each demonstration by sampling effective start frames o_s with $s \leq t$ drawn uniformly from the first 30% of the trajectory. Geometrically, replacing o_0 with o_s changes both the origin and the direction of the projection axis in Eq. (7): the reference vector $z_g - z_s$ rotates and rescales in representation space, so the model must learn to predict and project against a family of related axes rather than a single fixed direction. Each trajectory produces 4 augmented tuples per epoch, and we enforce a minimum separation of 5 frames between the effective start and the query frame to prevent degenerate near-identity projections. This augmentation is applied only during Stage 1; in Stage 2, the start frame is always the first frame of the fine-tuning demonstration.

Stage 2: Progress-conditioned action finetuning. Stage 2 introduces the GR00T FlowMatching action DiT and trains it to generate action chunks conditioned on the progress signals learned in Stage 1. In simulation all parameters are finetuned; for the real-world tasks, LoRA [16] adapters (rank 16, targeting q and v attention projections) are applied to the backbone to allow task-specific adaptation while preserving the pretrained foresight geometry. The three progress signals ($\hat{z}_g$, p_{cur}, Δp_{cur}) are detached before entering zero-initialized conditioning MLPs prepended to the DiT, so the flow-matching loss does not back-propagate into the readout heads and distort their calibration. The Stage 2 objective combines the flow-matching action loss with continued progress supervision:

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{flow}} + \lambda_{\Delta} \left\| \Delta p_{\text{cur}}^{\text{commit}} - \Delta p_{\text{next}}^{\text{gt}} \right\|_2^2 + \lambda_{\text{aux}} \, \mathcal{L}_{\text{pretrain}} \,, \tag{8}$$

where the auxiliary $\mathcal{L}_{\text{pretrain}}$ term maintains readout calibration as the backbone adapts.

A.3 Pretraining Dataset

Stage 1 requires only demonstration videos paired with language instructions, with no action labels needed. We curate **Progress-Mixture-260K**, a multi-embodiment pretraining mixture totaling approximately 260K demonstration episodes from a subset of Open X-Embodiment [38], DROID [19], and MolmoAct [23], spanning three robot platforms (Figure 7). DROID contributes 55K episodes (21%) on Franka with dense 15 FPS coverage across 564 in-the-wild scenes. Fractal (RT-1) [6] supplies 90K episodes (35%) on Google Robot at 3 FPS, offering extensive task variations for compositional generalization. Bridge V2 [45] provides 55K episodes (21%) from WidowX at 5 FPS, covering broad object and scene diversity. BC-Z [18] contributes 45K episodes (17%) on Google Robot at 10 FPS with high-quality teleoperated pick-and-place demonstrations. MolmoAct adds 15K episodes (6%) of Franka household manipulation, providing longer-horizon tasks that complement the shorter-horizon OXE data. The resulting platform distribution is approximately 27% Franka, 52% Google Robot, and 21% WidowX.

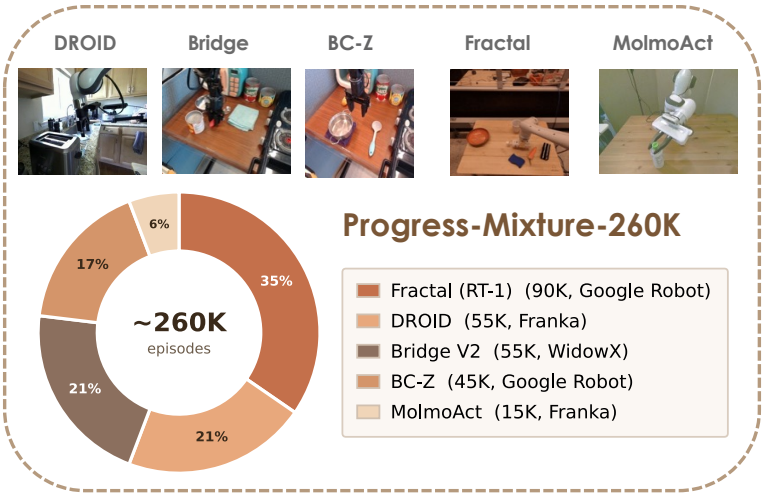

Figure 7: **Stage 1 pretraining data composition.** Approximately 260K episodes from five sources across three robot platforms, automatically annotated with geometric progress targets via frozen DINOv2-Base projection.

Automatic progress annotation. Before training, we annotate the entire Progress-Mixture-260K with geometric progress targets using the frozen DINOv2-Base encoder and the proportional projection from Eq. (7). For each episode the encoder extracts per-frame CLS embeddings, and the start-to-goal projection yields an absolute progress scalar p_t^{gt} and chunk-level increments $\Delta p_{\text{prev}}^{\text{gt}}$, $\Delta p_{\text{next}}^{\text{gt}}$ at every frame boundary. The terminal embedding z_T additionally serves as the foresight anchor target via $\mathcal{L}_{\text{goal}}$. This pipeline is fully automatic and the same set of targets simultaneously supervises both *where the task is going* (foresight anchor) and *how far execution has come* (progress readout), eliminating the need for separate annotation or training stages.

B Implementation Details

B.1 Model Architecture

PACE follows the widely adopted two-module VLA design in which a pretrained vision-language model produces a contextual representation that an action decoder converts into continuous motor commands [5, 3, 7]. The VLM backbone is Qwen3-VL-4B [1], a 4-billion-parameter vision-language model that fuses multi-resolution visual tokens with language embeddings through interleaved attention. The action decoder is a flow-matching DiT that generates action chunks via 4-step denoising, following the formulation demonstrated in π_0 [5] and GR00T N1 [3].

The progress interface appends $K{+}2$ learnable query tokens to the backbone input sequence, initialized from the embedding layer. K tokens $\mathbf{q}_{\text{goal}}$ are dedicated to foresight anchor prediction: their layer-18 hidden states are mean-pooled and projected through a two-layer MLP head to $\mathbb{R}^d$ (matching the visual encoder's CLS dimension, $d{=}768$ for the default DINOv2-Base). The remaining two tokens q_p and $q_{\Delta p}$ each project to a single progress value through a dedicated linear head, producing the absolute progress reading and forward commitment respectively. The realized increment Δp_{prev} enters the backbone as a scalar input token, so it shapes the states from which p_{cur} and Δp_{cur} are read out and, through them, the action decoder.

Progress conditioning tokens are prepended to the DiT input sequence rather than injected via AdaLN, following our ablation in Section D.1 showing that token prepend yields 0.9% higher success on LIBERO. The conditioning MLPs (2-layer, hidden dimension 768) are zero-initialized so that early Stage 2 training receives no progress signal, allowing the DiT to first converge under its standard flow-matching objective before gradually incorporating progress information. The commitment audit operates at inference time as a sliding-window residual over $W{=}3$ chunks with threshold $\tau{=}0.04$ (Section B.3).

Table 6: Training hyperparameters. Stage 1 pretrains progress perception on Progress-Mixture-260K without action labels. Stage 2 finetunes the full model for action generation on downstream task demonstrations.

Hyperparameter	**Stage 1: Progress Pretrain**	**Stage 2: Action Finetuning**
Data	Progress-Mixture-260K ($\sim$260K episodes)	Task-specific demos
Backbone	Qwen3-VL-4B (frozen)	Qwen3-VL-4B (full; LoRA for real)
Trainable parameters	Query tokens + projection heads	All (sim) / DiT + LoRA + heads (real)
Action decoder	—	Flow-matching DiT
LoRA rank	—	— (sim) / 16 (real)
LoRA target modules	—	— (sim) / q, v projections (real)
Optimizer	AdamW	AdamW
Learning rate	1×10^{-4}	$4{\times}10^{-5}$ / $1{\times}10^{-5}$ / $1{\times}10^{-4}$ (base / VLM / DiT)
LR schedule	Cosine decay	Cosine decay
Warmup steps	500	5000
Per-GPU batch size	16	16
Gradient accumulation	2	1
Number of GPUs	8$\times$ H200	4$\times$ GH200
Effective batch size	$16 \times 2 \times 8 = 256$	$16 \times 1 \times 4 = 64$ (sim) / 128 (real)
Training steps	30K	30K (LIBERO) / 80K (Bridge) / 20K (real)
λ_p (progress)	0.5	—
λ_c (consistency)	0.1	—
λ_Δ (commitment)	—	0.1
λ_{aux} (auxiliary pretrain)	—	0.1
Denoising steps (DiT)	—	4
Action chunk size H	—	8 (LIBERO) / 16 (Bridge) / 10 (real)
Training time	$\sim$25h	$\sim$7h (sim) / $\sim$4h (real)

B.2 Training and Hyperparameters

Table 6 lists all hyperparameters for both stages. Stage 1 pretrains on Progress-Mixture-260K (Section A.3), a curated collection of approximately 260K demonstration episodes across Franka, WidowX, and Google Robot platforms. With effective-start augmentation producing 4 tuples per trajectory, the effective training set contains roughly 1.04M samples per epoch. Since the backbone is kept frozen and only the lightweight query tokens and projection heads are updated, Stage 1 exposes the progress interface to diverse trajectories across three embodiments while completing in roughly 25 hours on 8 H200 GPUs.

Stage 2 introduces the flow-matching action DiT and finetunes the full model in simulation (LoRA for the real-world tasks). For LIBERO [28], we finetune jointly across all four suites, using the official 500 demonstrations per suite for 30K steps. For SimplerEnv, we finetune on Bridge dataset demonstrations and evaluate zero-shot on WidowX. For real-world tasks, we collect 60 teleoperated demonstrations per task and finetune jointly for 20K steps. The full loss specification is provided in Section A.2.

B.3 Inference Protocol

Algorithm 2 specifies the full inference loop in pseudocode. At each chunk boundary k, the frozen visual encoder extracts an embedding from the current observation o_k, and its geometric projection onto the foresight anchor gives the realized progress, while the VLM readout gives the $p_{\text{cur}}(k)$ that conditions the action decoder (Algorithm 2). The realized increment $\Delta p_{\text{prev}}(k) = p_{\text{cur}}(k) - p_{\text{cur}}(k{-}1)$ is computed from consecutive projections on the same scale as the training targets, where p_{cur} here denotes the projected value, requiring no additional supervision or learned component. Note that the realized increment $\Delta p_{\text{prev}}(k)$ reflects the actual number of actions executed between consecutive forward passes, which may be fewer than the predicted chunk size H when early re-inference is triggered. The progress interface is thus anchored to the *executed* control interval rather than the predicted action chunk, ensuring that the commitment and audit signals remain calibrated to what the policy actually delivered.

Commitment audit. After each chunk executes, we compare the realized increment $\Delta p_{\text{actual}}(k)$ against the forward commitment $\Delta p_{\text{cur}}^{\text{commit}}(k)$ that was produced *before* execution. The absolute residual $|\Delta p_{\text{audit}}(k)|$ is accumulated over a sliding window of $W{=}3$ chunks; when the windowed mean exceeds a threshold $\tau{=}0.04$, the system flags a potential execution failure. We fix $W{=}3$ and $\tau{=}0.04$ across all experiments (LIBERO, SimplerEnv, and real-world platforms) without per-task

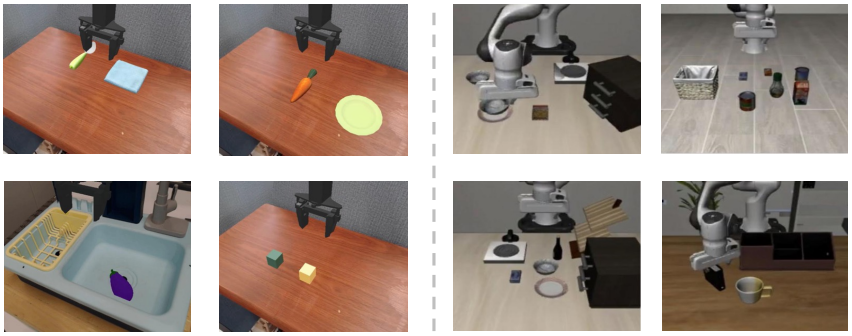

Figure 8: Simulation benchmarks. LIBERO (right) provides language-conditioned manipulation across four suites on Franka, with LIBERO-Long serving as the primary stress test for temporal grounding. SimplerEnv (left) evaluates cross-domain transfer on WidowX under visual appearance shifts.

tuning. In the current implementation the audit signal is used purely for execution monitoring and offline diagnostics rather than as a closed-loop correction trigger, consistent with the scope discussed in Section 4.5 of the main paper. Detection performance is evaluated via trajectory-level and chunk-level AUROC on human-annotated rollouts (Table 8).

C Experiment Settings

C.1 Simulation Environments

We evaluate on two complementary simulation benchmarks that probe different aspects of progress-aware control: LIBERO for language-conditioned task diversity across multiple progress regimes, and SimplerEnv for cross-domain robustness under visual appearance shifts.

LIBERO. We assess manipulation capabilities across four task suites on the Franka Panda arm [28]: *Spatial* (spatial reasoning with position references), *Object* (object attribute variations), *Goal* (goal specification diversity), and *Long* (compositional long-horizon tasks requiring multiple sequential subtask completions). We finetune **a single checkpoint jointly across all four suites** for 30K steps with effective batch size 64 and action chunk size $H{=}8$. Each suite contains 10 tasks, and we evaluate over 50 rollouts per task with randomized initial states. Success is determined by task-specific programmatic checks, including object placement within tolerance thresholds and sequential subtask completion for Long. LIBERO-Long is the primary stress test for our method: its 10 tasks each require 3–5 sequential subtasks where errors at any boundary can cascade, making temporal grounding essential. We report all four suites rather than Long alone, since a useful progress interface should preserve short- and medium-horizon competence without introducing unnecessary overhead.

SimplerEnv. We evaluate cross-domain transfer on the WidowX manipulation suite in SimplerEnv [26], which tests robustness to visual appearance shifts through controlled lighting and background perturbations. We train on Bridge dataset demonstrations and evaluate across four tasks: Put Spoon, Put Carrot, Stack Block, and Put Eggplant. Each task is assessed over 24 episodes with systematic visual variations. Training on real Bridge data and evaluating in simulation, on a platform whose morphology and camera viewpoint differ from the Franka setup of LIBERO, stress-tests whether the progress interface transfers across embodiments and visual domains. We finetune for 80K steps with batch size 64 and action chunk size $H{=}16$.

C.2 Real-World Environment

We deploy PACE on two physical platforms covering four tasks that each stress a different aspect of progress-aware control (Figure 9).



Figure 9: **Real-world robot platforms.** *Top:* Single-arm Franka Research 3 (7-DOF) with wrist and exterior cameras for Table Sorting and Cup Stacking. *Bottom:* Bimanual ARX Lift2S (14-DOF) with front-view and wrist cameras for Towel Folding and Block Disassembly. Both platforms run inference on a single RTX 5090 GPU for real-time control.

Single-arm platform. The single-arm setup uses a Franka Research 3 (7-DOF, maximum reach 855mm, repeatability $\pm$0.1mm) equipped with a Franka Hand parallel-jaw gripper. Visual observations come from two Intel RealSense D435 cameras: a *wrist camera* mounted near the end-effector for close-up manipulation feedback during grasping and placement, and an *exterior camera* on a fixed adjustable stand providing a third-person overview of the workspace. Both cameras capture 640$\times$480 RGB at 30 FPS, downsampled to 224$\times$224 for policy input. The workspace is a 120cm $\times$ 80cm tabletop with controlled overhead lighting. The policy runs on a single NVIDIA RTX 5090 GPU with action chunk size H=10, yielding one forward pass per second of real-time execution.

Bimanual platform. The bimanual setup uses a pair of ARX Lift2S arms mounted on a shared mobile base, providing a 14-dimensional joint action space (7-DOF per arm). Each arm is equipped with a parallel-jaw gripper for coordinated grasping and manipulation. Visual observations follow the same wrist-plus-global configuration: a *front-view camera* provides a global perspective of the shared workspace, and a *wrist camera* captures close-up views for fine contact feedback. Both cameras capture 640$\times$480 RGB at 30 FPS, downsampled to 224$\times$224. The policy runs on the same NVIDIA RTX 5090 GPU. Bimanual coordination introduces additional progress-awareness challenges: the two arms must synchronize across temporally offset subtasks, and temporary regression (e.g., releasing one side to regrasp) is more frequent than in single-arm manipulation.

Task design. *Table Sorting* (single-arm) requires the robot to categorize and sort scattered food items on a cluttered tabletop by semantic category. Given a language instruction such as "sort up the meat on the table" or "sort up vegetables on the table," the robot must identify items belonging to the target category (e.g., fish, lobster, chicken drumstick for meats), grasp each item, and place it into a designated container. Object identities, quantities, and initial positions are randomized across trials. A trial is successful when all items of the specified category have been correctly placed. This task tests whether the progress interface enables sustained multi-step execution with correct category discrimination: without temporal grounding, the policy tends to pick incorrect items or revisit already-sorted ones.

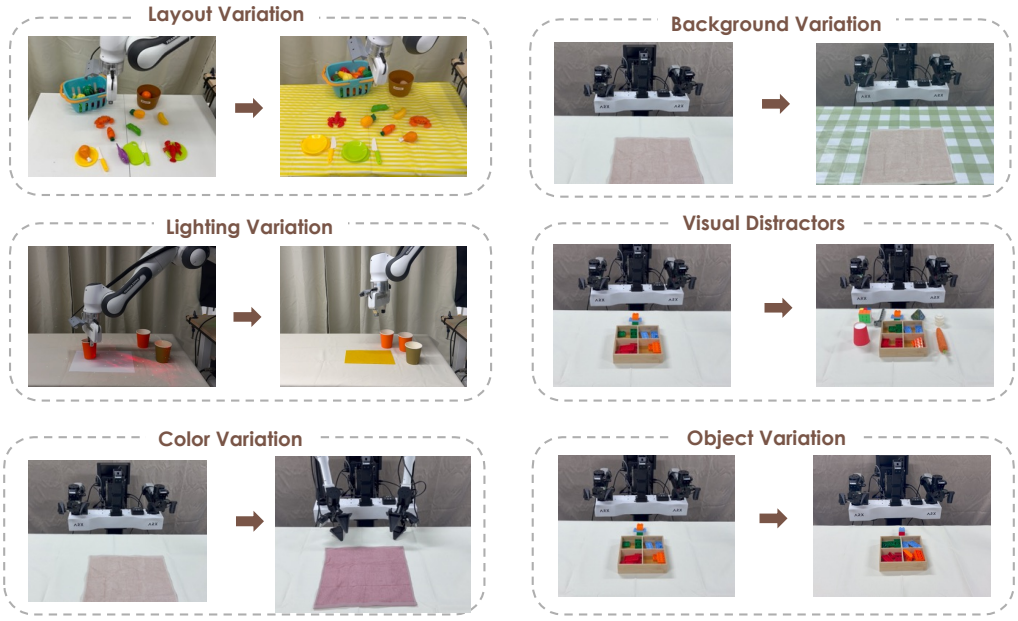

Figure 10: **Real-world generalization perturbations.** We evaluate robustness under several categories of distribution shift: *Layout Variation* (randomized object positions), *Background Variation* (different surroundings), *Lighting Variation* (altered illumination), *Visual Distractors* (irrelevant objects added to the workspace), *Color Variation* (modified object appearances), and *Object Variation* (unseen instances not present in training). These perturbations are grouped into Visual, Object, and Task shift categories for the generalization analysis in Section 4.6.

Cup Stacking (single-arm) requires building a two-layer pyramid from three cups. The robot must first place two base cups into stable proximity at a target location, then grasp the third cup and stack it on top. A trial is successful when the three-cup structure remains stable for 3 seconds after the robot releases. The key challenge is a sustained alignment phase during which the robot makes correct micro-adjustments to the base cups but visible progress is minimal, followed by an abrupt stacking transition. A policy without progress awareness often initiates the stacking motion prematurely, collapsing the structure.

Towel Folding (bimanual) spans multiple folding stages where deformable contact can plateau or locally regress across folds. Given the instruction "fold the towel," the robot must grasp two edges of the cloth, pin down one side before regrasping, execute one or more folding motions, and produce a neatly folded result. Success requires the final configuration to match a target folded shape within visual tolerance. The progress-awareness challenge is distinguishing genuine stalling from slow but steady alignment of deformable material, where visual change is minimal but task advancement continues.

Block Disassembly (bimanual) begins with interlocking colored bricks assembled in a fixed configuration. The robot must pick up the assembled structure using coordinated dual-arm grasping, then separate individual bricks and place each into its correspondingly colored location. The disassembly order is not prescribed; the robot must determine a feasible sequence from the current configuration. A trial is successful when all bricks have been separated and placed at their designated positions. The initial pick-up and separation phases often produce temporary negative progress on the foresight-anchored scale, since lifting and pulling apart the structure temporarily moves the scene away from the final sorted configuration. This makes the task a natural test bed for the commitment mechanism's ability to accommodate non-monotonic advancement.

Generalization evaluation. To assess whether the progress interface transfers beyond the training configuration, we evaluate under three categories of distribution shift on the real-world platform (Figures 10–11). *Visual shift* alters the scene appearance through randomized lighting, background replacement, and visual distractors while keeping objects and instructions unchanged. *Object shift*

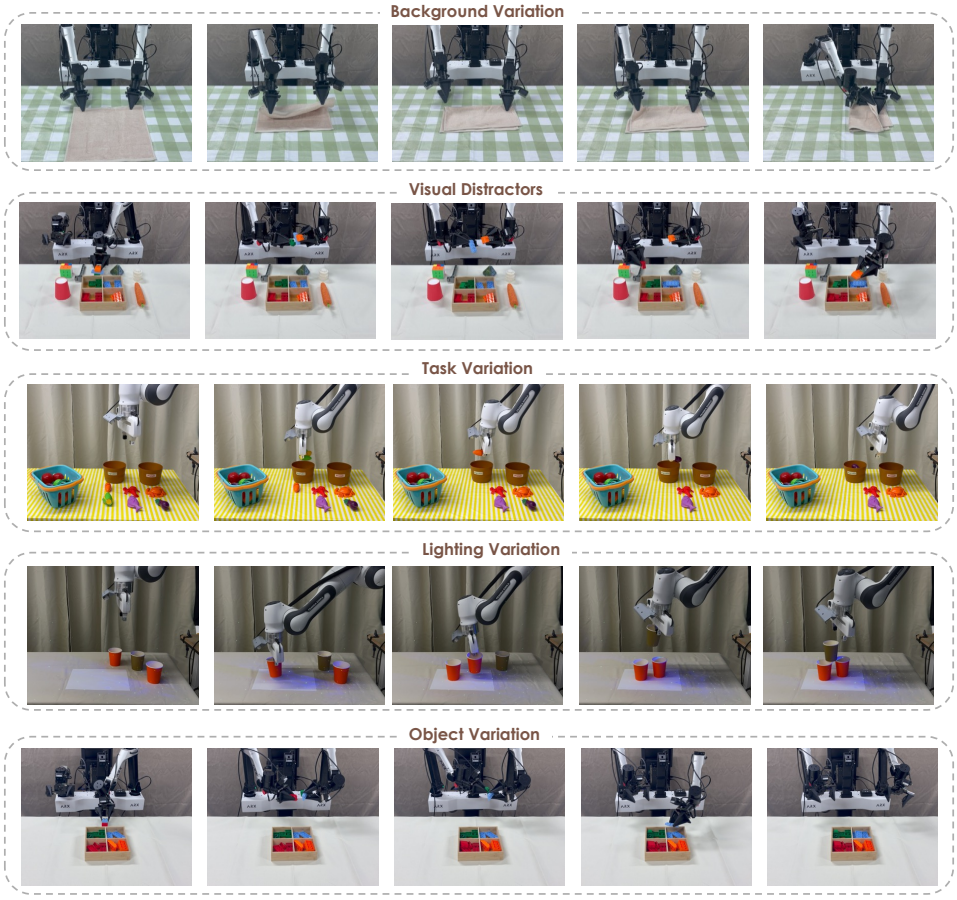

Figure 11: **Real-world generalization perturbations on both platforms.** We evaluate robustness under background variation, visual distractors, task variation, lighting changes, and object variation, corresponding to the three shift categories.

replaces training objects with unseen instances of similar semantic category (e.g., different cup colors or food items not seen during demonstration collection). *Task shift* introduces novel compositional instructions that alter the goal structure, requiring the policy to coordinate across subtask sequences it has never rehearsed. For each shift category, we run 50 rollouts per method on the single-arm and bimanual platforms and report success rate in Section 4.6.

Training protocol. For each real-world task, we collect 60 teleoperated demonstrations via kinesthetic teaching by two operators, recorded at 10 Hz. We finetune the Stage-2 model for 20K steps with batch size 128, action chunk size $H{=}10$, and LoRA rank 16. Data augmentation includes random cropping ($\pm 10\%$) and color jittering to improve robustness to lighting variations. We set a maximum episode length of 300 timesteps (30 seconds at 10 Hz), marking trials as failures if tasks remain incomplete. All evaluation uses 50 rollouts per method with randomized object positions and scene layouts, and success is verified manually. All baselines are finetuned with identical demonstrations, training steps, batch size, and augmentation pipeline, differing only in their respective model architectures and progress interfaces. Baselines without progress interfaces (OpenVLA, SpatialVLA, UniVLA) use the same LoRA configuration and learning rate schedule.

Table 7: Fine-grained progress interface ablations. LIBERO: average success rate across all four suites (%). Real: average over four real-world tasks (%). Upper block varies *what* enters the action decoder; middle block varies *training procedure*; lower block varies *how* progress enters the decoder. Bold marks the full model.

Variant	LIBERO Avg (%) ↑	Real Avg (%) ↑
Base VLA (no progress)	94.0	45.0
Foresight anchor only (backbone shaping, no DiT input)	95.2	48.0
Foresight conditioned ($\hat{z}_g \rightarrow$ DiT, no progress scalars)	96.0	56.0
Co-decode (progress heads active, not fed back to the decoder)	95.4	56.5
Condition on p_{cur} only	97.2	64.0
w/o Commitment ($p_{\text{cur}} + \Delta p_{\text{prev}}$, no Δp_{cur})	97.6	67.5
w/o History ($p_{\text{cur}} + \Delta p_{\text{cur}}$, no Δp_{prev})	97.4	66.0
w/o Stage 1 pretrain (all heads trained from scratch in Stage 2)	96.4	58.5
AdaLN conditioning	97.8	69.0
PACE (token prepend)	**98.7**	**74.0**

D Additional Results and Analysis

D.1 Progress Interface Ablations

The main paper ablates three high-level components of the progress interface (Section 4.3). Here we provide finer-grained variants that isolate additional design questions not addressed in the main paper, including whether the foresight anchor alone suffices as a conditioning signal, whether Stage 1 pretraining on Progress-Mixture-260K contributes beyond the architecture, and how historical pace and forward commitment each shape action generation. All variants share the same Qwen3-VL-4B backbone and action decoder; only the progress pathway and training procedure differ. Table 7 reveals a clear progression from no progress awareness to the full interface.

Structured progress adds value beyond goal conditioning alone. A potential concern is that the foresight anchor $\hat{z}_g$ already captures the task endpoint, and feeding it directly into the action decoder might provide sufficient temporal guidance without additional progress scalars. The *Foresight conditioned* variant tests this: $\hat{z}_g$ enters the DiT as a conditioning token, but no progress reading (p_{cur}, Δp_{cur}) is produced or conditioned on. This variant reaches 96.0% on LIBERO and 56.0% on real tasks, a meaningful improvement over the base VLA (94.0% / 45.0%) that confirms the value of goal-aware action generation. Adding structured progress on top further improves to 98.7% / 74%, a gain of 2.7% in simulation and 18% on real robots. The gap reflects that a single goal embedding tells the decoder *where* the task is going but not *how far* execution has come or *how fast* it should advance, information that the compact progress scalars provide in a form that is both more explicit and more directly actionable than a high-dimensional latent vector. The anchor itself need not be exact: grouping LIBERO-Long rollouts by their median anchor error $\|\hat{z}_g - z_g\|$ gives 99.0, 98.5, 97.9 and 96.6% success at errors of 0.05, 0.09, 0.15 and 0.24, so even an error of about a third of the task axis costs only 2.4%.

Historical pace and forward commitment play complementary temporal roles. Removing the forward commitment Δp_{cur} while retaining perception and history costs 1.1% on LIBERO and 6.5% on real tasks, and removing the historical pace Δp_{prev} produces a slightly larger drop (1.3% / 8.0%). The two quantities provide temporal context from opposite directions: Δp_{prev} disambiguates the static p_{cur} reading by revealing whether the task has just transitioned, stalled, or recovered, while Δp_{cur} declares the intended near-future advancement and feeds it forward into the action decoder. In a counterfactual test that fixes the frame and the progress value and varies only the recent pace, the predicted action shifts by 7–8$\times$ the same-input sampling noise. The real-world gap is consistently larger than the simulation gap, suggesting that temporal context becomes more valuable when the policy encounters the variability and noise of physical execution. Together with absolute progress, these two relative quantities form the minimal past–present–future interface that equips the policy with a complete sense of its own execution pace.

Stage 1 pretraining on diverse data strengthens the progress geometry. Skipping Stage 1 and training all progress heads from scratch during Stage 2 action finetuning drops performance to 96.4% / 58.5%, a 2.3% / 15.5% gap relative to the full model. The architecture is identical in both cases, so the difference isolates the contribution of large-scale progress pretraining. Importantly, the backbone is kept entirely frozen during Stage 1; only the lightweight query tokens and their projection heads receive gradient updates. If the improvement arose from the model absorbing

Table 8: Failure detection comparison on human-annotated rollouts from LIBERO and real-world tasks. The commitment audit achieves the strongest detection accuracy as a built-in byproduct of the progress interface, without requiring any additional model or inference overhead.

Detector	Type	Extra latency	Traj. AUROC ↑	Chunk AUROC ↑
Low p_{cur}	heuristic	—	0.72	0.65
Neg. realized Δp	heuristic	—	0.78	0.71
Monotonicity violation	heuristic	—	0.80	0.74
RoboReward [24]	learned reward	+65 ms	0.85	0.78
Robometer [27]	learned reward	+70 ms	0.87	0.81
Commitment audit (ours)	**built-in**	**+2 ms**	**0.92**	**0.85**

richer visual representations through exposure to 260K additional episodes, the frozen backbone—which houses the vast majority of learned representations—could not benefit. The observed gain is therefore attributable to the progress-specific perceptual objective rather than general representation enrichment from additional data. Stage 1 leverages a key structural advantage of the progress interface: perceiving absolute progress requires only observation frames and language instructions, not action labels, allowing the foresight anchor and progress readout to train on 260K diverse trajectories across three embodiments that would be unavailable to an action-supervised stage. This broad exposure builds a robust progress geometry that the downstream history and commitment signals depend on, since both relative quantities are meaningful only when the absolute reading they derive from is well-calibrated. Without this foundation, the progress heads must simultaneously discover the geometric structure of progress and calibrate it to task-specific demonstrations, a dual burden that 500 demonstrations per suite cannot sustain without overfitting to the training distribution.

Token prepend outperforms AdaLN as the conditioning mechanism. The final two rows compare how progress enters the action decoder, a design choice that any conditional diffusion planner faces [34]. AdaLN modulates layer normalization statistics throughout the DiT, while token prepend adds progress as input tokens that participate in self-attention alongside the action sequence. Token prepend yields 0.9% higher success on LIBERO and 5.0% on real tasks. We attribute this to the prepend design preserving the DiT's original attention structure, whereas AdaLN modifies normalization globally and can interact unpredictably with the flow-matching denoising dynamics.

D.2 Failure Detection Comparison

Section 4.5 shows that the commitment audit outperforms heuristic baselines for execution failure detection. To place this result in broader context, we extend the comparison to include RoboReward [24] and Robometer [27], two recent general-purpose reward models that represent the current state of learned trajectory evaluation. For each detector, we run inference on the same set of human-annotated rollouts from LIBERO and real-world tasks. External reward models score every frame and accumulate per-chunk score deltas over the same sliding window ($W{=}3$) used by the commitment audit, ensuring that all methods operate on identical temporal granularity.

Heuristic baselines. Writing p_k for the progress at chunk k, the three heuristic detectors score a rollout from the realized progress curve alone. *Low* p_{cur} measures how far the best progress reached so far falls short of completion, $s = 1 - \max_{j \le k} p_j$. *Negative realized* Δp is the fraction of chunks that move backwards, $s = \frac{1}{k} \sum_{j \le k} \mathbf{1}[\Delta p_j < 0]$. *Monotonicity violation* is the largest drawdown from the running peak, $s = \max_{j \le k} \left(\max_{i \le j} p_i - p_j \right)$. The commitment audit instead averages the residual of Eq. (6) over the last $W{=}3$ chunks, $s = \frac{1}{W} \sum_{j=k-W+1}^{k} |\Delta p_{\text{audit}}(j)|$. Because the heuristics see only what was realized, an intended regrasp and a failure look the same to them, whereas the audit compares each realized increment with the policy's own earlier commitment.

The commitment audit matches dedicated reward models at negligible cost. Table 8 reports trajectory-level and chunk-level AUROC across all six detectors. The two learned reward models substantially outperform the heuristic baselines, reaching 0.85–0.87 trajectory AUROC, reflecting the strong semantic grounding that large-scale trajectory supervision provides. The commitment audit achieves 0.92 trajectory AUROC and 0.85 chunk AUROC, the highest on both metrics, while adding only 2 ms of latency since the residual between committed and realized increments is already computed as part of the standard inference loop. Each learned reward model, by contrast, requires a separate VLM forward pass per frame, adding 65–70 ms per chunk.

Table 9: Progress source compatibility. Upper block: cross-family comparison (ordered by rank correlation). Lower block: DINOv2 encoder-scale comparison (ordered by model size). Calibration metrics are evaluated on curated expert demonstrations spanning LIBERO and OXE; SR is reported on LIBERO-Long. All variants share identical policy architecture, training data, and action decoder.

Progress source	**Labels**	**Extra model**	**Rank corr.** $\uparrow$	**SR (%)** $\uparrow$	p_{cur} **MAE** $\downarrow$
Normalized time t/T	no	no	0.72	92.0	0.121
CLIP endpoint geom.	no	frozen	0.78	92.4	0.089
Qwen3-VL hidden geom.	no	backbone	0.82	92.8	0.074
Robometer	yes	trained	0.87	93.0	0.052
RoboReward	yes	trained	0.88	93.4	0.048
DINOv2-S geom. (ViT-S/14)	no	frozen	0.91	94.2	0.058
DINOv2-B geom. (default)	**no**	**frozen**	**0.97**	**98.0**	**0.041**
DINOv2-L geom. (ViT-L/14)	no	frozen	0.93	95.2	0.050
DINOv2-G geom. (ViT-g/14)	no	frozen	0.94	96.0	0.047

Task-specific calibration as a built-in advantage. Learned reward models produce general-purpose progress scores trained across diverse embodiments and task families, which makes them effective trajectory evaluators in aggregate. At the chunk level, however, the scoring function may not perfectly align with the task-specific progress geometry that the policy operates on. The commitment audit compares the policy's own forward prediction against what actually happened, using the same foresight-anchored scale that conditions action generation. This coupling between the conditioning signal and the monitoring signal means that detected deviations are directly relevant to the policy's own decision-making context, which may explain the stronger chunk-level detection. A learned error probe in the spirit of PrObe [51], trained on our backbone, reaches 0.86 trajectory AUROC on the same rollouts. It needs error-labelled supervision but can pick up patterns that a scalar commitment does not carry, so the two signals are complementary.

The two paradigms offer complementary properties. Learned reward models can evaluate trajectories *post hoc* from any VLA without requiring the policy to produce a forward commitment, making them broadly applicable as offline evaluation tools. The commitment audit is available only in policies that include a commitment mechanism as part of their interface, but in return provides real-time monitoring that is inherently calibrated to the policy's own progress representation. In settings where both signals are available, fusing them could yield more robust detection, a direction we leave for future exploration.

D.3 Progress Source Compatibility

The experiments above use DINOv2-Base geometry as the default progress source. This section examines how the interface behaves when the underlying source changes, first across *encoder scales* within the DINOv2 family and then across entirely different *source families*, while keeping the policy architecture, training data, and action decoder identical in every case. All calibration metrics (rank correlation, p_{cur} MAE) are computed against geometric ground truth on curated expert demonstrations spanning LIBERO and OXE (covering Franka, WidowX, and DROID platforms), which exhibit near-monotonic forward advancement and thus provide a stable reference for assessing signal quality. Success rate is reported on LIBERO-Long.

All DINOv2 scales exhibit progress geometry, with ViT-B as the practical sweet spot. The lower block of Table 9 evaluates four DINOv2 variants spanning a 40$\times$ parameter range (ViT-S/14 through ViT-g/14). All four achieve rank correlation above 0.91 and SR above 94%, indicating that the start-to-goal projection captures a property broadly shared across the DINOv2 representation family. ViT-B/14 achieves the strongest overall calibration (rank correlation 0.97, SR 98.0%) and stands notably above the other three scales, which cluster between 94% and 96% SR. One plausible explanation is that the foresight anchor must be *predicted* from the VLM hidden state, and a moderate target dimension (d=768 for ViT-B) may offer a favorable balance between learnability of this prediction and the expressiveness needed to distinguish progress-relevant variation. ViT-S compresses the target to d=384, possibly limiting fine-grained discrimination, while ViT-L (d=1024) and ViT-g (d=1536) expand it without yielding commensurate gains in the downstream conditioning loop. We adopt ViT-B/14 as the default, noting that the other variants remain viable and may close the gap with additional pretraining data or longer Stage 1 schedules.

The property extends across encoder families. Replacing DINOv2 entirely with five alternative sources (upper block of Table 9) shows that progress-like structure exists in a variety of representation

spaces, though at varying quality levels. Normalized time t/T provides a lower bound: treating all demonstrations as uniformly paced still yields 92.0% SR, above the 90.6% of the base VLA without any progress signal on LIBERO-Long (Figure 5), but offers minimal benefit. CLIP endpoint geometry and Qwen3-VL hidden-state geometry improve incrementally, the latter being noteworthy in that it derives progress from the VLM's *own* intermediate-layer CLS embeddings onto the start-to-goal axis, requiring no external encoder. Its reasonable rank correlation (0.82) suggests that the VLM backbone encodes some degree of task-advancement structure, though the joint language-modeling objective may dilute the spatial geometry that a dedicated self-supervised encoder preserves more cleanly. The two learned reward models (Robometer and RoboReward) achieve 93.0–93.4% SR with tighter absolute calibration (MAE 0.048–0.052), benefiting from dense semantic supervision. The gap to DINOv2-B (98.0%) suggests that temporal ordering consistency, which self-supervised spatial correspondence training tends to produce, may be particularly well-suited for the progress internalization pipeline.

Signal quality and downstream performance. Across the nine sources in Table 9, higher rank correlation generally coincides with stronger task success, and better absolute calibration tends to accompany higher SR. The association is broadly positive but not strictly monotonic across all individual entries, which is expected given that closed-loop performance depends on the interaction of several factors beyond a single calibration metric, including embedding dimensionality, anchor learnability, and how well the source's geometry aligns with the denoising dynamics of the action decoder. The practical takeaway is that DINOv2-B geometry provides the strongest downstream performance among all sources tested while requiring no dense labels and no additional model training. More broadly, the observation that all nine sources produce functional policies through the same interface, with SR ranging from 92% to 98%, supports the view that **the value of PACE lies primarily in the policy-side mechanism that predicts and conditions on progress**, with the specific source acting as an interchangeable input whose quality modulates but does not gate the benefit.

E Qualitative Analysis

E.1 Progress Curves in Practice

Figures 12–14 visualize the predicted progress trajectory alongside the committed and realized curves on representative rollouts from simulation and real-world tasks. Each plot overlays three signals: the *actual progress* p_{cur} computed from frozen DINOv2 projections, the *committed progress* accumulated from the model's forward declarations Δp_{cur}, and an ablated variant that removes the historical pace input Δp_{prev} to reveal the stabilizing role of temporal context.

Accumulated commitment tracks task structure faithfully. A single-chunk commitment Δp_{cur} is a local quantity that can fluctuate with observation noise, partial occlusion, or minor pose variation. When accumulated over the trajectory, however, these per-chunk fluctuations average out, and the resulting committed curve traces the same macro-level staircase as the realized progress. This smoothing effect is visible across all three tasks: the committed and actual curves may diverge momentarily at individual chunks, yet their accumulated trajectories remain tightly coupled throughout successful execution. The accumulated form is also what makes the audit residual practical. A per-chunk deviation can spike due to a single noisy observation and would produce frequent false alarms if thresholded directly. The windowed accumulation absorbs transient noise while preserving genuine execution failures, which manifest as a sustained and growing gap rather than an isolated spike.

Historical pace sharpens commitment prediction. The ablated variant (dotted gray curve) removes Δp_{prev} from the model input, forcing the commitment head to predict the next increment without knowing how much the previous step actually advanced. The effect is striking: without history, the committed curve oscillates erratically, frequently overshooting or undershooting the actual progress by large margins. This is most pronounced during transitional phases, such as the lingering period before placement in Figure 12 or the fold transition in Figure 13, where the pace of advancement changes abruptly. With historical pace available, the commitment head can detect whether the task has just accelerated, stalled, or is in a steady phase, and calibrate its forward declaration accordingly. The gap between the two committed curves confirms that Δp_{prev} is not merely a redundant input but provides the temporal derivative information that transforms a noisy per-chunk guess into a calibrated forward commitment.

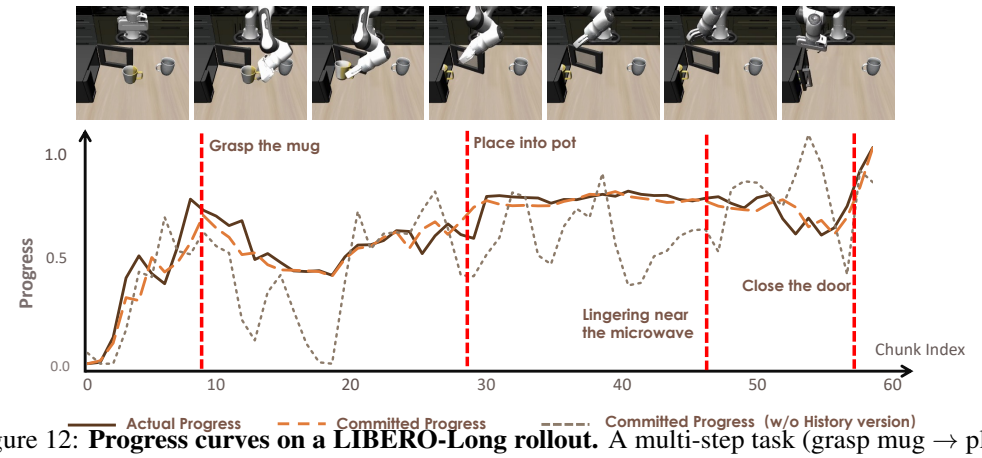

Figure 12: **Progress curves on a LIBERO-Long rollout.** A multi-step task (grasp mug → place into pot → move near microwave → close door) with subtask boundaries marked by red dashed lines. The accumulated committed progress closely tracks actual progress, while the history-free ablation (dotted) exhibits large oscillations, particularly during the lingering phase near the microwave.

Subtask boundaries produce characteristic progress signatures. Red dashed lines in each figure mark subtask transitions identified from the task instruction. These boundaries tend to coincide with inflection points in the progress curve, where a rising segment during active execution gives way to a brief plateau or local regression as the robot transitions to the next target. In Figure 12, the plateau between chunks 30–46 corresponds to the robot lingering near the microwave after placing the mug into the pot, a phase where visual change is minimal but the policy maintains near-zero commitment rather than hallucinating forward progress. In Figure 13, the progress dip around chunk 30 coincides with the transition between folding stages, where repositioning the grasp temporarily moves the scene away from the final folded configuration. These qualitative patterns are consistent with the quantitative findings in Section 4.4, suggesting that the progress interface captures the temporal rhythm of multi-step manipulation at a resolution that aligns with interpretable task structure.

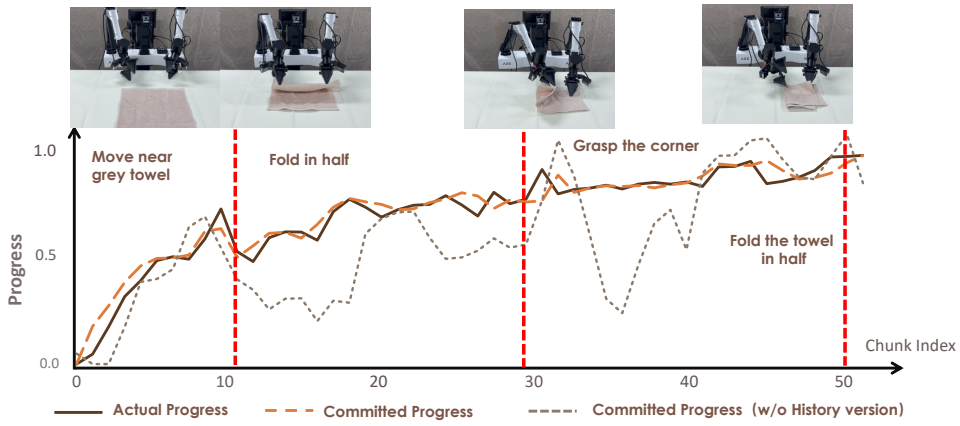

Figure 13: **Progress curves on a real-world towel folding rollout.** Four subtask stages (approach towel → fold in half → grasp corner → second fold) are marked by dashed lines. The progress dip near chunk 30 coincides with the transition between folding stages, where repositioning the grasp temporarily reduces measured progress. Removing historical pace (dotted) destabilizes the commitment prediction across all phases.

E.2 Single-Arm Tasks

Table Sorting (Figure 15) requires the robot to identify, grasp, and place multiple food items into the correct container according to a semantic category instruction. Across successful trials, the policy maintains coherent sequencing through three to four pick-and-place cycles, correctly discriminating between visually similar items such as fish and lobster under the "meat" instruction and switching

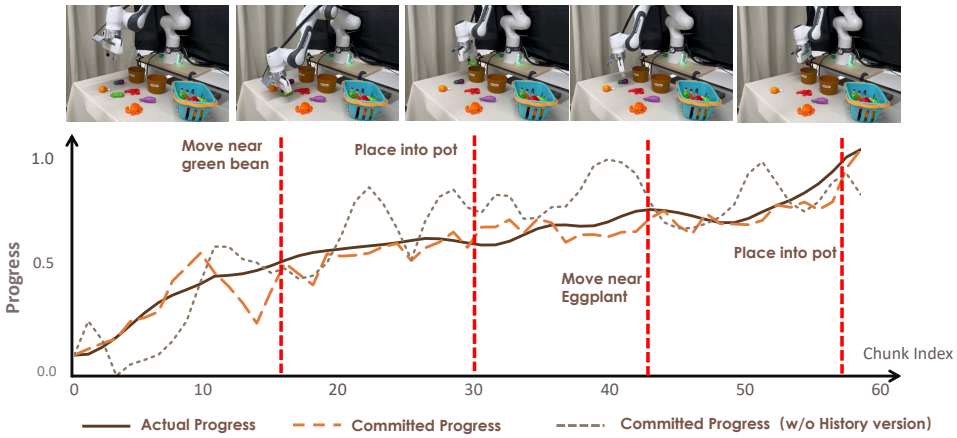

Figure 14: **Progress curves on a real-world table sorting rollout.** The task involves sequential pick-and-place of multiple items (green bean → eggplant → second placement). Committed and actual progress co-vary through each cycle, with the history-free variant (dotted) producing substantially noisier predictions that drift from the realized trajectory.

targets between categories across trials. The second row shows a notable recovery: the robot initially loses grip on the target object mid-transport but regrasps and completes the placement without disrupting subsequent subtasks, suggesting that the historical pace signal helps the policy recognize a stall and re-engage rather than proceed to the next target prematurely. The failure case in the top row traces to a spatial misalignment between the gripper and the container rim, causing the object to land outside the target zone. This type of contact-level error falls outside what the progress interface can prevent, though the resulting commitment–realization divergence makes it readily detectable after the fact.

Cup Stacking (Figure 16) demands a prolonged alignment phase followed by a single decisive stacking motion. Successful trials show the policy patiently adjusting the base cups through a series of micro-corrections where visible scene change is minimal, then executing a confident vertical placement once the base configuration stabilizes. This patience is consistent with the forward commitment signaling near-zero expected advancement during alignment and a larger positive increment at the transition to stacking. Failures occur when the top cup slips during the final placement. Because the policy has already committed to a substantial forward increment at this point, the mismatch between intention and outcome surfaces clearly in the audit residual, distinguishing the failure from the low-activity alignment phase that precedes it.

E.3 Bimanual Tasks

Towel Folding (Figure 17) involves coordinated grasping and multi-stage folding of a deformable cloth. Successful trials demonstrate stable bimanual coordination across the grasp–flatten–fold sequence, with the two arms alternating between pinning and pulling roles as the cloth transitions through intermediate configurations. The progress trajectory in these trials is notably non-monotonic: releasing one edge to regrasp at a better angle temporarily moves the scene away from the folded goal state, yet the policy treats this regression as a planned corrective motion rather than an error. The failure case stems from a missed initial grasp on one edge of the towel, which propagates into a malformed fold that the policy cannot recover from. This highlights that while the progress interface helps coordinate multi-phase manipulation, it cannot compensate for upstream contact failures that corrupt the cloth configuration early in the sequence.

Block Disassembly (Figure 18) requires the robot to lift an interlocking structure, separate individual bricks, and sort them by color into designated compartments. Successful trials show the policy executing a consistent disassembly order: a coordinated dual-arm lift followed by sequential single-brick removals, with each separation producing a brief phase of negative progress as the workspace temporarily diverges from the final sorted configuration before snapping forward once the brick reaches its target slot. The failure case occurs when bricks slip from the gripper during the initial coordinated lift. The policy continues to expect forward advancement while the structure remains

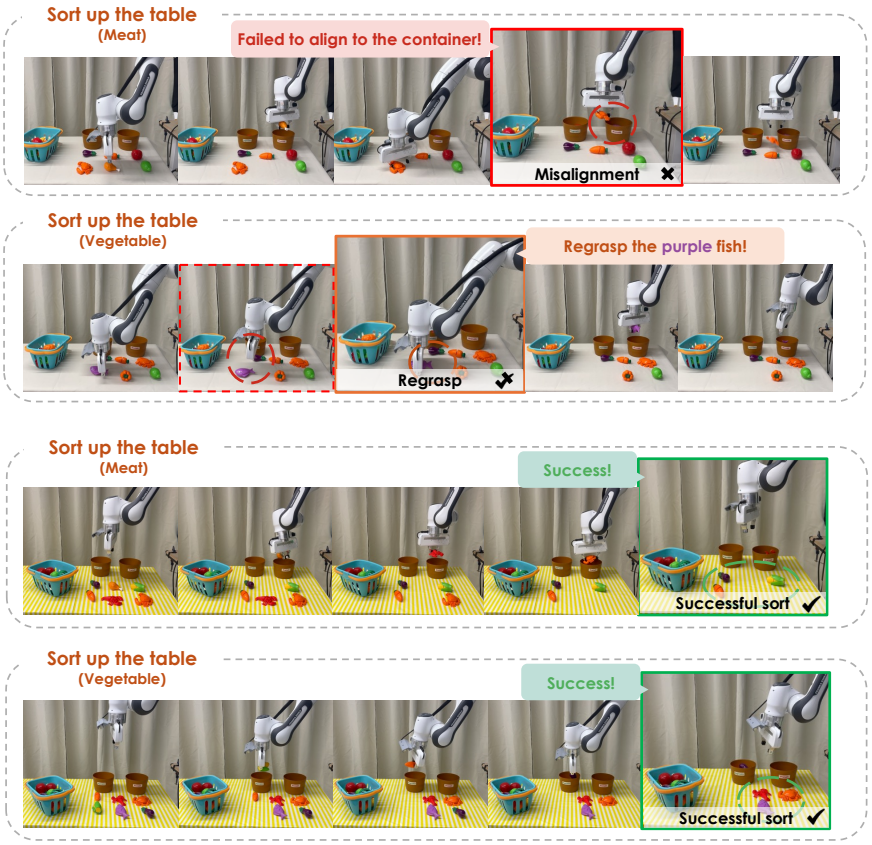

Figure 15: **Table sorting case study.** Successful cases (green) demonstrate sustained multi-step categorization under different instructions. The failure case (red, top) exposes a container misalignment during placement, while the second row illustrates recovery through regrasping.

unseparated, producing an accumulating gap between committed and realized progress that grows over several chunks. This sustained divergence pattern is characteristic of contact failures in bimanual tasks, where the error compounds across both arms rather than resolving through single-arm retry.

E.4 Failure Patterns and Future Directions

Across the four real-world tasks, the majority of failures share a common upstream cause rather than reflecting a shortcoming of the progress interface itself. Understanding where and how failures arise helps clarify which aspects of manipulation the current interface already covers and where complementary signals could extend its reach.

Contact-level failures. The most frequently observed pattern is a missed or unstable grasp that prevents the intended manipulation from taking effect. In table sorting this appears as a slip during transport, in cup stacking as an insufficiently secure grip before the stacking motion, and in block disassembly as a loss of hold during the coordinated lift. These are fundamentally low-level contact events that occur before the progress interface has an opportunity to shape the subsequent trajectory. What the interface does provide is early visibility: when the expected advancement fails to materialize, the commit–realize gap widens within one or two chunks, making the event straightforward to detect even without force or tactile sensing. Incorporating such sensing as a complementary input channel is a natural next step that could enable the policy to anticipate contact instability before it disrupts the progress trajectory.

Appearance-driven anchor shift. Under substantial visual domain shift, particularly novel lighting conditions or unseen object appearances, the foresight anchor can gradually drift from the true goal configuration. Because the entire progress scale is derived from this anchor, a shifted reference



Figure 16: **Cup stacking case study.** Successful cases (green) show stable pyramid construction through sustained alignment. Failure cases (red) occur when the top cup slips during the final stacking transition, a scenario where the commitment audit detects the abrupt progress–commitment divergence.

manifests as a slow, systematic bias in the progress reading rather than an abrupt divergence. The effect is most visible on the generalization evaluation (Section 4.6), where the gap between in-distribution and out-of-distribution performance reflects, in part, anchor quality. Enriching the visual diversity during Stage 1 pretraining is a direct mitigation, and the source-agnostic design of the interface (Section D.3) opens the door to anchoring progress on representations that are inherently more robust to appearance variation, such as multimodal reward embeddings or depth-aware encoders.

Bimanual desynchronization. Towel folding and block disassembly occasionally exhibit a failure mode specific to dual-arm coordination: one arm completes its subtask while the other stalls or regresses, producing conflicting progress signals that the current single-scalar interface averages over. Extending the interface to maintain per-arm progress channels could capture this asymmetry more faithfully, enabling the policy to re-synchronize rather than proceeding with a partially completed bimanual action. This direction connects naturally to the broader idea of multi-axis temporal self-awareness discussed in Section F.

Slow stalling. A small fraction of failures involve the policy entering a low-activity loop where neither the committed nor the realized increment is large, so the audit residual stays inconspicuous even though the task is not advancing. These cases tend to arise when the robot has partially completed a subtask and the remaining motion is ambiguous from the current viewpoint. Rather than exposing a gap in the progress signal, they suggest that progress alone is a necessary but not always sufficient indicator of healthy execution. Complementing the progress audit with lightweight activity-level cues, such as end-effector displacement or action entropy, could flag these quiescent stalls and prompt re-engagement without introducing a separate reward model.

Taken together, these observations reinforce that the progress interface reliably captures the temporal structure of long-horizon manipulation and surfaces most execution anomalies as a built-in byproduct. The remaining failure patterns point less to limitations of the commit–verify principle than to

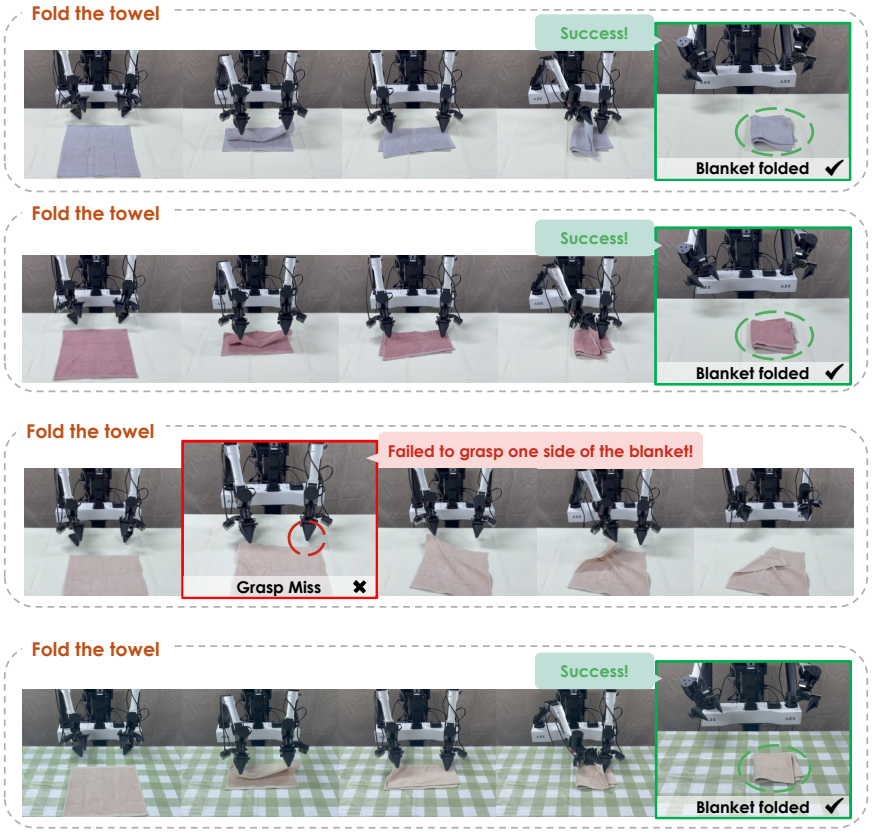

Figure 17: **Towel folding case study.** Successful cases (green) complete multi-stage folding with coordinated bimanual grasps. The failure case (red) stems from a missed grasp on one edge, causing the subsequent fold to collapse.

opportunities for enriching the sensory and structural inputs that the interface operates on, directions we discuss further in Section F.

F Limitations and Future Work

Building and evaluating the progress interface across simulation and real-world platforms has revealed both the reach and the natural boundaries of the current design. The observations below outline where these boundaries point to concrete extensions, each building directly on the progress interface that PACE establishes.

Hierarchical anchoring for compositional tasks. The foresight anchor predicts a single terminal-state embedding, which provides a stable and effective reference for manipulation tasks with a well-defined endpoint. Compositional tasks with conditional branching or goals that shift mid-execution could benefit from a richer representation, and the commit-then-verify loop is not inherently tied to a single anchor. A natural extension is a hierarchy of intermediate anchors, each grounding progress over a local task segment with its own commitment and audit axis. Subtask-level commitments would guide immediate actions while a global commitment tracks coarse advancement across segments, enabling the policy to reset its local progress scale after each boundary. Discovering these intermediate anchors automatically from unlabeled demonstrations, for instance through keyframe identification [21] or changepoint detection in the foresight embedding trajectory, is a tractable problem that could unlock substantially longer task compositions than the current interface supports. Future frames forecast by a streaming video model [30] are another possible source of such anchors. Visual planners that generate subgoal images [4, 35] and stage-based reward models such as R^2VLM [55] and VICtoR [17] are natural sources as well.



Figure 18: **Block disassembly case study.** Successful cases (green) demonstrate coordinated dual-arm separation and color-sorted placement. The failure case (red) occurs when bricks slip from the gripper during the initial lift, producing the sustained negative audit residual described in Section E.4.

From detection to active recovery. The commitment audit surfaces execution failures within a few chunks at zero additional cost, providing a temporally localized, signed error signal that is well suited to drive a recovery module. A promising direction is to condition a lightweight replanning policy on the audit residual, using its magnitude and sign to select among strategies such as reattempting the current subtask, retreating to a previous stable configuration, or requesting human assistance. The residual could also modulate action-generation temperature, encouraging more exploratory behavior when coherence breaks down and more conservative execution when the gap is small. Combining this with the hierarchical anchoring above, where the policy can re-enter any subtask boundary after a detected deviation, would close the loop between detection and correction. Choosing among $K{=}4$ sampled chunks with a lightweight head that predicts each one's commit-realize deviation already improves LIBERO-Long by 0.8%.

The commit-realize gap as an intrinsic reward. Beyond failure detection, the audit residual has a deeper structural property: it is a self-generated prediction error that correlates with task advancement. Recent work has shown that progress-based dense rewards substantially accelerate online RL fine-tuning of VLA policies [2, 53], but these methods rely on an *external* progress estimator. PACE already generates an analogous signal internally: a policy that consistently under-delivers on its commitments accumulates a persistent negative residual, while one that advances as planned sees the residual vanish. Using the commit-realize gap as an intrinsic dense reward for test-time RL could enable the policy to refine itself through environment interaction without any additional learned model. The key advantage is that the reward is automatically calibrated to the policy's own progress geometry, ensuring the optimization target and the conditioning signal share the same representational frame. Whether this self-supervised loop converges reliably across diverse tasks and embodiments is a question we find particularly exciting.

Multi-modal and scalable progress perception. The default progress source, frozen DINOv2 geometry, is most effective when task advancement produces visible state change. The source-

agnostic interface validated in Section D.3 already shows that alternative representations, from VLM hidden states to trained reward models, slot into the same loop with minimal modification, and this modularity naturally accommodates non-visual modalities. Extending the interface to tactile or force-based progress predictors could substantially strengthen performance on contact-rich manipulation such as insertion or deformable material handling, where visual appearance saturates well before the task is complete. On the data side, Stage 1 requires only videos with language instructions and no action labels, making it directly compatible with internet-scale video corpora. Scaling to such data could broaden the foresight anchor's generalization to novel tasks and unfamiliar categories, and exploring whether the geometric projection transfers across substantially different embodiment morphologies, from tabletop arms to humanoid platforms [3, 40], is a compelling direction.

Toward broader temporal self-awareness. The commit-then-verify principle at the heart of PACE is not specific to task progress. Any quantity that can be predicted before an action, realized after execution, and compared against the prediction defines a potential conditioning signal and its own audit axis. Subgoal completion, contact timing, energy expenditure, or safety margins could each serve this role, equipping the policy with a richer temporal self-model that captures not only how far the task has advanced but also how efficiently, how safely, and how reliably it is being executed. Composing multiple such signals into a unified interface would move VLA design closer to the kind of temporal self-awareness that underlies skilled human manipulation. The same predict-then-verify loop could also extend beyond manipulation to other sequential motion generation, such as sign language production [57].

More broadly, PACE takes a step toward a simple but underexplored idea: that robot policies benefit from being asked to predict more than they strictly need to act. Current VLA models excel at mapping observations to motor commands, yet this purely reactive loop leaves the policy without any internal model of how its own execution unfolds over time. By asking the policy to perceive where a task stands, commit to where it should go, and verify whether execution delivered on that commitment, we equip it with a lightweight temporal self-model that improves action quality, surfaces failures, and remains interpretable, all within a single forward pass. The directions above suggest that this commit and then verify loop is far from exhausted: richer anchoring, multi-modal sensing, self-supervised adaptation, and multi-axis temporal awareness each offer concrete paths to deepen the interface without altering its core design. We believe PACE represents an early point on this trajectory rather than a final design, and we hope it encourages the community to rethink what VLA policies should internalize, moving beyond predicting what to do next toward understanding what doing it should accomplish.

G Broader Impacts

PACE is foundational research on how a vision-language-action policy organizes its own execution over time, and we expect its societal impact to mirror that of the broader VLA literature it builds on. On the positive side, equipping policies with a calibrated sense of execution *pace* can make long-horizon manipulation more reliable, which directly benefits applications such as assistive home robotics, eldercare, warehouse and laboratory automation, and any setting where graceful behavior under partial failures matters more than peak performance under ideal conditions. The built-in commit–realize audit also lowers the barrier to deploying long-horizon policies in safety-sensitive environments, since it surfaces emerging failures from inside the policy itself rather than requiring a separate monitoring stack. On the negative side, more capable manipulation policies may accelerate displacement in routine physical-labor jobs, and the same temporal self-model that improves benign execution could in principle be reused in dual-use settings such as autonomous physical surveillance or unsanctioned automation. The interface itself does not introduce new high-risk capabilities beyond what existing VLAs already provide, and we view the most plausible mitigations as community-level rather than method-level: open release with documented limitations, transparent reporting of failure modes (Appendix E), and continued attention to deployment context. We do not anticipate immediate misuse risks specific to PACE, and we encourage downstream users to pair the audit signal with appropriate human oversight rather than treating it as a standalone correctness guarantee.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 3.1 through Section 3.4 describe the foresight-anchored progress interface and the commit-realize audit residual. The reported headline numbers (98.7% on LIBERO, 74% on real-world manipulation) and the qualitative claims about generalization and failure detection are supported by Section 4.1–4.7.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 and Appendix F (Limitations and Future Work) discuss the single-endpoint assumption of the foresight anchor, the gap between failure detection and active recovery, the dependence on visual progress signals for contact-rich manipulation, and the bimanual desynchronization failure mode observed in real-world rollouts.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not state formal theorems or lemmas; the equations introduced in Section 3.1–3.4 are loss-function and projection definitions rather than results requiring proof.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Appendix A specifies the foresight-anchor and progress-target construction, the two-stage training algorithm (Algorithm 1), and the inference-time audit procedure (Algorithm 2). Appendix B describes the model architecture (Section B.1) and full training hyperparameters (Table 6). Appendix C documents the simulation and real-world evaluation protocols.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code and trained checkpoints are open-sourced and available from the project page. The pretraining mixture is composed entirely of public datasets cited in Appendix A.3, and the architecture, training algorithm, and hyperparameters in Appendix B are detailed enough to support an independent reimplementation.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix B.2 (Table 6) lists the optimizer, learning rate, schedule, batch size, training steps, loss weights, LoRA configuration, and hardware for both stages. Appendix C describes the simulation and real-world evaluation protocols, including trial counts and demonstration counts.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: We follow the standard evaluation protocol in the VLA literature and report mean success rates over large trial counts (500 per LIBERO suite, 50 per real-world task).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix B.2 (Table 6) reports the GPU type, GPU count and approximate training time, and Appendix C.2 the hardware used for real-world deployment.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics and confirm that the work conforms to it. The paper studies a policy interface for robotic manipulation; no human-subjects data, sensitive personal data, or scraped web content was collected.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix G discusses both positive impacts (more reliable assistive, eldercare, warehouse, and laboratory robotics; built-in failure auditing for safety-sensitive deployment) and negative impacts (labor displacement in routine manipulation work, dual-use concerns shared with the broader VLA literature), together with the mitigation considerations we view as appropriate.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a high-risk model or scraped dataset. The pretrained backbones used (Qwen3-VL, DINOv2) are publicly released by their original authors under their respective terms of use.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets and pretrained models used are credited in-text with formal citations. The pretraining mixture (DROID, Bridge V2, Open X-Embodiment / RT-1, BC-Z, MolmoAct) is documented in Appendix A.3, and the model backbones (Qwen3-VL-4B, DINOv2-Base) and benchmarks (LIBERO, SimplerEnv) are cited in Appendix B.1 and Appendix C. We use each asset within its publicly released terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released code and checkpoints come with documentation, linked from the project page. The Progress-Mixture-260K used for Stage 1 pretraining is a curation of existing public datasets cited in Appendix A.3; the curation protocol is documented there.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: All data collection (real-world teleoperation) and annotation (failure-detection AUROC labels) were performed by the authors; no crowdsourcing or third-party human subjects were involved.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human-subjects research; no participant risk assessment or IRB review is required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The proposed VLA uses Qwen3-VL-4B as its vision-language backbone, in line with standard practice for recent VLA architectures. The role and configuration of the backbone are documented in Appendix B.1 (model architecture). LLMs were not used as authoring or editing tools for the methodology itself.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.