DUALFUTURE: EVENT-TRIGGERED DUAL-FUTURE RESIDUAL CORRECTION FOR VISION-LANGUAGE-ACTION POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Action-chunked vision-language-action (VLA) policies execute a predicted chunk before observing its consequences, so a small prediction error at a critical action event goes uncorrected, accumulates over the executed commands, and fails the episode. We introduce **DualFuture** (Dual-Future Residual Correction), an event-triggered framework that wraps a pretrained VLA without fine-tuning its weights. At a detected event it predicts two futures—one consistent with task success, one induced by the proposed action—and decodes their discrepancy into a bounded residual added to the action before execution. Both futures and the residual are learned from realized rollout futures and known action differences, so no manual correction labels are needed. Instantiating critical events as persistent gripper transitions, we attach DualFuture to baselin VLA policy on RoboCasa365 and report three further frozen policies to situate its base success. Success rises from 73.8% to 78.7% on atomic tasks and from 45.4% to 50.9% on unseen composite tasks. On five tabletop tasks with a real Franka robot, DualFuture raises the frozen policy's pooled success from 40.0% to 52.0% under randomized object placements.

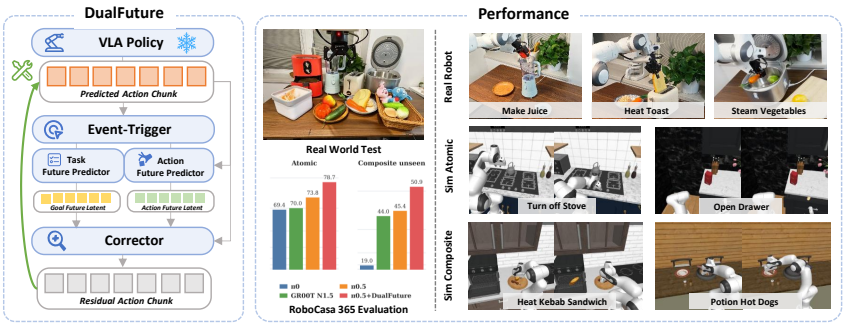

Figure 1: *Left:* Framework of DualFuture. At an imminent critical event, DualFuture decodes the gap between the futures implied by the instruction and by the proposed action into a bounded residual that corrects the raw action chunk to improve the frozen policy. *Right:* Our DualFuture consistently improves VLA policy on RoboCasa365 atomic and composite tasks and on a real Franka robot.

1 INTRODUCTION

Vision-language-action (VLA) policies map visual observations and language instructions to general-purpose robot control (Brohan et al., 2023; Kim et al., 2025; Black et al., 2025; NVIDIA et al., 2025). Many modern VLAs predict an *action chunk* and execute several commands before observing the consequences, reducing inference cost and improving temporal coherence (Zhao et al., 2023; Chi et al., 2023). This brief open-loop interval becomes brittle near actions that commit the scene to a new physical state. Small pose or timing errors at gripper closure can destabilize a grasp, while early release can miss the intended receptacle. The next policy query may then come too late to recover.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This motivates sparse correction at *critical action events*, before errors create hard-to-recover states. Existing interventions have distinct limitations. CycleVLA supports recovery through subtask backtracking, but reversing robot motion cannot restore altered object states (Ma et al., 2026). FPC-VLA corrects gripper-related actions before execution, but uses predefined directions and discrete magnitudes (Yang et al., 2026). VLA-in-the-Loop generates successful grasp videos before corrective actions, adding visual synthesis (Xu et al., 2026). VLA-Corrector responds to observed latent deviations with gradient-guided replanning, requiring a new policy query and online gradients (Pan et al., 2026). We therefore seek a corrector that intervenes before physical commitment, learns to revise the proposed chunk, and avoids intermediate video generation and gradient-based replanning.

We formulate correction as alignment between two local futures in a shared latent space. A *task-consistent future* predicts a successful local change from the observation, instruction, and event without conditioning on actions; an *action-induced future* predicts the change produced by a proposed chunk. **DualFuture** (Dual-Future Residual Correction) translates their discrepancy into a bounded residual over arm motion and event timing before execution. It wraps a frozen VLA, intervening only when a gripper opening or closing transition is imminent in the chunk's executable prefix; otherwise, the base policy runs unchanged. We learn this mapping through *rollout-driven self-supervision*: successful rollouts supply anchor chunks paired with alternative proposals from the same event snapshot, whose known action differences provide residual targets. This eliminates manual residual labels and end-to-end VLA fine-tuning.

We attach DualFuture to frozen VLA policy $\pi_{0.5}$ and evaluate it on RoboCasa365 under the pretrain-plus-target protocol, pairing corrected and base episodes from the same initialization. Success rises from 73.8% to 78.7% on atomic tasks and from 45.4% to 50.9% on unseen composite tasks. Across five tabletop tasks on a real Franka robot, the same design raises pooled success from 40.0% to 52.0% under randomized object placements and improves every task.

Our contributions are:

- We propose DualFuture, an event-triggered wrapper that corrects a frozen VLA by aligning a task-consistent future with the action-induced future of its proposed chunk.
- We introduce rollout-driven self-supervision, training the corrector from success-anchored action pairs branched from executed rollouts without manual residual labels or end-to-end VLA fine-tuning.
- We evaluate DualFuture on RoboCasa365 atomic and compositional tasks against matched correction baselines, report three additional frozen policies to contextualize base success, demonstrate 4.9- and 5.5-point gains over the base policy, and show improvements on a real Franka robot.

2 Related Work

2.1 Pre-Execution Correction without Subtask Backtracking

RT-2 and OpenVLA advance vision-language-action learning (Brohan et al., 2023; Kim et al., 2025), while ACT, Diffusion Policy, and flow-based VLAs predict coherent action sequences (Zhao et al., 2023; Chi et al., 2023; Black et al., 2025; Physical Intelligence et al., 2025). Executing a prefix without updated policy feedback can leave errors uncorrected at critical grasping or release events. Residual reinforcement learning augments existing controllers (Johannink et al., 2019), while SC-VLA, Phoenix, and FailSafe introduce reflection and recovery (Li et al., 2024; Xia et al., 2025; Lin et al., 2025). Recovery, however, cannot always restore the physical state needed for success.

CycleVLA anticipates failures and backtracks across subtasks, but reversing robot motion does not reset object states (Ma et al., 2026). Without subtask backtracking, FPC-VLA corrects gripper-related actions before execution using predefined directions and discrete magnitudes (Yang et al., 2026), while VLA-Corrector truncates and replans after detecting predicted–observed latent deviations (Pan et al., 2026). These approaches motivate revising imminent chunks according to anticipated task outcomes before disruptive actions require recovery. DualFuture decodes task-consistent and action-induced futures into a bounded residual, correcting the frozen policy's chunk before execution without subtask backtracking.

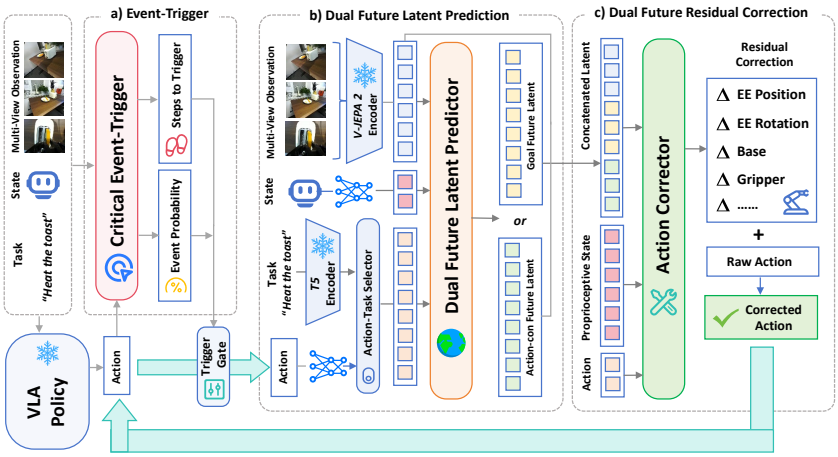

Figure 2: Overview of DualFuture. **(a)** The event-trigger opens the gate only at an imminent gripper transition; otherwise the frozen chunk runs unchanged. **(b)** One DFLP with shared weights is queried twice from the same latents, yielding a goal latent and a proposal latent. **(c)** The corrector predicts a bounded residual over end-effector pose, base, and gripper; the projected corrected chunk is executed, after which control returns to the frozen policy.

2.2 Latent Future Prediction for Efficient Correction

Predictive models support visual and learned-dynamics planning (Finn & Levine, 2017; Chua et al., 2018; Nagabandi et al., 2018), policy optimization (Janner et al., 2019), and latent imagination (Ha & Schmidhuber, 2018; Hafner et al., 2023). WorldVLA jointly trains action generation and future-image prediction (Cen et al., 2025). Goal-VLA derives targets from generated goal images (Chen et al., 2025b), while VLA-in-the-Loop generates grasp videos to guide correction (Xu et al., 2026); these latter pipelines require visual synthesis before producing action updates.

FLARE incorporates future-latent alignment into policy learning (Zheng et al., 2025), and Seer conditions inverse dynamics on predicted representations (Tian et al., 2025), integrating foresight into action generation rather than a frozen-policy residual wrapper. Latent prediction also does not eliminate online search: V-JEPA 2-AC iteratively optimizes actions against image-goal representations (Assran et al., 2025), while FOREWARN evaluates alternative latent rollouts for candidate selection (Wu et al., 2025). Both add computation between prediction and execution. DualFuture instead directly revises the proposed chunk with two latent-future predictions and one residual decoding pass per triggered event, avoiding future-image generation, candidate search, and online gradient optimization.

3 DualFuture

DualFuture wraps a frozen action-chunked VLA. Given observation $s_t = \{I_t^{(v)}, q_t\}$ and instruction l, the VLA proposes an action chunk $a_t^0 = \{u_{t+k}^0\}$. Its first H_e commands form the executable prefix that DualFuture may correct.

The wrapper has three modules (Fig. 2). First, a trigger decides whether a gripper event is imminent. Second, a shared predictor estimates (i) the goal future specified by l and (ii) the future induced by a_t^0. Third, a corrector uses their difference to produce a bounded action residual. If no event is detected, a_t^0 is returned unchanged, so the wrapper is identical to the base policy away from critical events. The VLA and visual encoder remain frozen. The three trainable modules are supervised by event labels, realized rollout futures, and known action differences, respectively, all derived from the common rollout-driven training pipeline in Sec. 4.

3.1 CRITICAL-EVENT TRIGGER

We firstly train a standalone trigger $\mathcal{T}_{\text{grip}}$ that uses the current observation I_t, q_t, and instruction l to predict the probability $\hat{p}_t$ of a gripper event in the next 50 steps and the number of steps to event $\hat{\tau}_t$, which we define as the onset of a gripper opening or closing:

$$(\hat{p}_t, \hat{\tau}_t) = \mathcal{T}_{\text{grip}}(I_t, q_t, l). \tag{1}$$

We model the trigger as a Transformer, encoding the robot state with a residual MLP, the instruction with a frozen T5 encoder (Raffel et al., 2020), and multi-view observations with a frozen Eagle2.5-VL vision encoder (Chen et al., 2025a). A two-layer Transformer fuses these modality tokens, and its `[CLS]` representation is passed to two MLP heads predicting the future-opening/closing probability and time-to-opening/closing. Its training objective is

$$\mathcal{L}_{\mathcal{T}} = 1.0\,\mathcal{L}_{\text{reg}} + 0.5\,\mathcal{L}_{\text{trig}} + 0.2\,\mathcal{L}_{\text{future}} + 0.1\,\mathcal{L}_{\text{count}}. \tag{2}$$

where the four terms are smooth L1 loss for time-to-event regression, BCE for whether there is an event, BCE for whether future-opening/closing exists, smooth L1 loss for countdown consistency, which regresses prediction_later - (prediction_earlier - delta_steps), respectively; time regression is applied only to positive samples. In Eq. 1, $\hat{p}_t$ determines whether an opening/closing onset exists, while τ_{e} requires it to be close enough to fall inside the next few executable prefixs. At inference, the trigger output is used to decide whether to activate the correction pipeline:

$$e_t = \mathbf{1}[\hat{p}_t \geq p_{\text{e}} \ \wedge \ \hat{\tau}_t \leq \tau_{\text{e}}] \tag{3}$$

When $e_t = 0$, the remaining modules are skipped and a_t^0 is executed.

3.2 DUAL-FUTURE LATENT PREDICTOR

Both the future predictor and the corrector use the same latent target. A frozen V-JEPA 2 encoder E (Assran et al., 2025) embeds the frames for 3 camera views left, right, and wrist:

$$z_t^{(v)} = E(I_t^{(v)}), \quad v = \{\text{L}, \text{R}, \text{W}\}. \tag{4}$$

The normalized view latents are concatenated as

$$z_t = \big[\, \bar{z}_t^{(\text{L})};\ \bar{z}_t^{(\text{R})};\ \bar{z}_t^{(\text{W})} \,\big], \qquad \bar{z}_t^{(v)} = \nu\big(z_t^{(v)}\big), \tag{5}$$

where ν normalizes each view and gradients do not pass through E. This common latent space allows the goal and action futures to be compared directly, without defining a task-specific geometric target.

The *Dual-Future Latent Predictor* (DFLP) is one conditional model F_ϕ with shared weights. Its inputs are the current visual latents $z_t = E(I_t)$, proprioception q_t, and a condition c; its output is a predicted future latent:

$$\hat{z}_{t+H_e} = F_\phi(z_t, q_t, c). \tag{6}$$

This follows the encoder–predictor factorization of V-JEPA 2:

$$X_t^m = [\text{Linear}(\text{LayerNorm}(z_t)), \text{MLP}(q_t), e_m, c_a^m, c_l^m] \tag{7}$$

where e_m is the mode selector, and X_t^m is the input to a 24-layered Transformer estimating an unobserved future from that context and an additional condition. In our model, the condition is either language or an action chunk:

$$c = \begin{cases} \phi_l(l), & \text{task-consistent (goal) future,} \\ \phi_a(a), & \text{action-induced (consequence) future,} \end{cases} \tag{8}$$

where ϕ_l is a frozen T5 (Raffel et al., 2020) instruction encoder and ϕ_a embeds an action chunk. Conditioning on the instruction predicts the goal future:

$$\hat{z}_t^g = F_\phi\big(z_t, q_t, \phi_l(l)\big). \tag{9}$$

It represents the local future that a successful completion of the detected event should reach, without committing to a particular action chunk.

Conditioning on an action chunk predicts its induced future:

$$\hat{z}_t^a(a) = F_\phi\big(z_t, q_t, \phi_a(a)\big). \tag{10}$$

It represents the local consequence of executing that specific chunk.

The goal branch is trained on successful trajectories; the action branch is trained on the transitions induced by its input chunks, including all trajectories. Both regress the realized future latents at $t + H_e$ conditioned on the current observation, and at $t + 2H_e$ conditioned on the predicted future at $t + H_e$:

$$\begin{aligned} \mathcal{L}_G = \|\hat{z}_t^g - z_{t+H_e}\|_1, \qquad \mathcal{L}_A = \|\hat{z}_t^a - z_{t+H_e}\|_1, \\ \mathcal{L}_{\text{rollout}} = \frac{1}{2}\left(\|\hat{z}_{t+2H_e}^g - z_{t+2H_e}\|_1 + \|\hat{z}_{t+2H_e}^a - z_{t+2H_e}\|_1\right). \end{aligned} \tag{11}$$

which are averaged before backpropagation: $\mathcal{L} = \frac{1}{2}(\mathcal{L}_G + \mathcal{L}_A)/2 + \frac{1}{2}\mathcal{L}_{\text{rollout}}$. The target z_{t+H_e} is stop-gradient, so neither loss updates E.

At inference, we predict one future per condition. This keeps the two branches directly comparable. The second branch evaluates the VLA proposal, $\hat{z}_t^{a,0} = F_\phi(z_t, q_t, \phi_a(a_t^0))$. Thus, $\hat{z}_t^g$ describes the desired goal future and $\hat{z}_t^{a,0}$ describes the proposal's induced consequence future. Both are passed to the corrector.

3.3 Dual-Future Residual Correction

With these two future predictions in hand, we now describe how the corrector converts them into an executable action update. The corrector takes the goal future $\hat{z}_t^g$, consequence future $\hat{z}_t^{a,0}$, current visual latent z_t^{cur}, normalized proprioception q_t, and action proposal a_t^0. It outputs an action residual:

$$\Delta a_t = C_\theta(\hat{z}_t^{a,0}, \hat{z}_t^g, z_t^{\text{cur}}, q_t, a_t^0), \tag{12}$$

The three latent streams are token-wise normalized and projected to encode current–consequence, current–goal, and goal–consequence relations, including the signed and absolute goal–consequence difference. The resulting features condition a three-layer Transformer decoder whose queries combine the proposed action, proprioceptive state, and learned timestep embeddings:

$$\begin{aligned} x^i = W_z\,\text{LayerNorm}(z^i), \quad i \in \{c, a, g\}, \quad d = \text{LayerNorm}(z^g) - \text{LayerNorm}(z^a), \\ M_t = \text{Concat}\Big(\big\{x^c + \text{MLP}([x^c, x^j, x^j - x^c])\big\}_{j\in\{a,g\}}, \\ x^a + \text{MLP}([x^a, x^g, W_d d, W_{|d|}|d|])\Big), \\ \Delta a_{t-cont}, a_{t-disc} = W_c, \text{sigmoid}\, W_d\big[\text{Decoder}\big(\text{MLP}(a_t^0) + \text{MLP}(q_t) + e_{1:H_e},\, M_t\big)\big]. \end{aligned} \tag{13}$$

We train the corrector from pairs of chunks described in Sec. 4.3 that start at the same snapshot z_{shared}: $\big((z_{\text{source}}, z_{\text{goal}}), z_{\text{shared}}, q, a_{\text{source}}, \Delta a^\star\big)$. If a_{source} and a_{goal} reach latent futures z_{source} and z_{goal}, the future pair specifies the source and goal outcomes. Then, the shared snapshot, source action, and current state provide the context needed to turn that latent difference into an action residual. The target residual is their known action difference:

$$\Delta a^\star = a_{\text{goal}} - a_{\text{source}}. \tag{14}$$

Thus, the corrector acts as a latent inverse-dynamics model. In implementation, continuous targets are computed in execution space and normalized by action group, while discrete dimensions are supervised separately. The corrector minimizes

$$\mathcal{L}_C = \mathcal{L}_{\text{cont}}^{\text{SmoothL1}} + \lambda_{\text{disc}}\mathcal{L}_{\text{disc}}^{\text{BCE}} + \lambda_{\text{size}}\|\Delta a\|_2^2 + \lambda_{\text{noop}}\mathcal{L}_{\text{noop}}, \tag{15}$$

where $\mathcal{L}_{\text{cont}}$ regresses the group-normalized continuous residual using the SmoothL1 loss, $\mathcal{L}_{\text{disc}}$ supervises the two discrete switch heads using binary cross-entropy, the size term regularizes correction magnitude, and $\mathcal{L}_{\text{noop}}$ suppresses corrections to 0. Since $\Delta a^\star$ is observed, this loss does not backpropagate through F_ϕ.

At inference, the realized training futures $(z_{\text{source}}, z_{\text{goal}})$ are replaced by the predicted proposal and goal futures $(\hat{z}_t^{a,0}, \hat{z}_t^g)$. The residual is added to the executable prefix from the frozen VLA and forms

$$a_t^1 = \Pi(a_t^0 + \Delta a_t). \tag{16}$$

The projection Π maps the output to the controller execution space by clipping continuous commands and thresholding discrete dimensions. The continuous residual changes end-effector pose and base commands over the executable prefix. When the trigger fires, the projected corrected chunk is executed. Otherwise, a_t^0 is executed unchanged.

4 Event-Aligned Branched-Rollout Supervision

We construct data by rolling out the frozen VLA across training tasks and environment variations. Then, at a reset point before a critical event on the successful trajectories, we sample multiple action chunks from the same pre-event state and record or predict the future caused by each chunk. For a pre–event snapshot t, this produces the branch set

$$\mathcal{B}_t = \{(a_t^+, z_t^+)\} \cup \{(a_t^k, z_t^k)\}_{k=1}^K, \tag{17}$$

where $+$ denotes the successful anchor and every z_t^k is the encoded consequence of executing alternative a_t^k for the common horizon H_e.

Each branch set supplies two forms of supervision. Event windows train the trigger; and every alternative– anchor pair provides an observed action difference for the corrector. The learned future predictor therefore describes the discrepancy used at deployment but never generates the residual label used for training.

4.1 Event-Aligned Branch Collection

On successful trajectories, an event detection rule $\mathcal{T}$ maps timestep t to either no event or an event descriptor. The correction pipeline does not depend on a specific $\mathcal{T}$; in our experiments, we detect a gripper-transition(open/close) onset. Let $q_{0:T}$ denote the observed gripper-state trace and $\mathcal{S}(q_{0:T})$ the retained opening/closing onsets produced by $\mathcal{T}$. The rule uses an adaptive motion threshold and temporal filtering to suppress short, noisy, and duplicate transitions. The event time after frame t is

$$\tau_t^\star = \min\left\{k \geq 0 :\ t + k \in \mathcal{S}(q_{0:T})\right\} \tag{18}$$

Within a lookahead window $\tau_{\max}$, this gives an event label $y_t^{\mathrm{e}} = \mathbf{1}[\tau_t^\star \leq \tau_{\max}]$ and, for positive samples, a time-to-event target $\tau_t^\star$.

From each successful trajectory, we retain multiple reset states before onsets, their executed chunk a_t^+, and the realized latent z_t^+. In simulation, we restore each reset state, draw $K = 4$ alternative chunks a_t^k from the frozen VLA with independent sampling seeds, and execute each to obtain z_t^k. We retain reset groups that pass reset-consistency and horizon-validity checks. In real-world experiments, we predict z_t^k conditioned on the sampled a_t^k using the train set DFLP action mode.

4.2 Asymmetric Dual-Future Supervision

To train the two modes of the shared DFLP, we use different subsets of the full rollout trajectories. The task-consistent mode sees only successful ones because it should predict the local future consistent with the instruction. The action-induced mode sees every trajectory and also the alternative branches $\bigcup_t \{(a_t^k, s_t, z_t^k)\}_{k=1}^K$, where t are the pre-event reset points, because consequence prediction must cover both successful and failed actions.

4.3 Grounded Future-to-Action Supervision

We train the corrector using the branch sets $\mathcal{B}_t$. For each alternative branch, the successful anchor defines a destination future and a model-free residual target:

$$\mathcal{D}_C = \left\{ \left(z_t^+, z_t^k, s_t, a_t^k, \Delta a_t^\star\right) : \Delta a_t^\star = a_t^+ - a_t^k \right\}. \tag{19}$$

Here z_t^k is the source future and z_t^+ is the goal future for the corrector, matching Eq. 12. Their paired executed actions yield an exact action-space difference for that pair.

The construction of the branched sets holds the local state, instruction, and horizon fixed, isolating the change in realized future due to the action chunk.

Table 1: Primary paired evaluation on the RoboCasa365 target splits (single-skill *atomic*, multi-step *composite*). The upper block of frozen policies establishes the range of base success; the lower block attaches DualFuture and two correction baselines to $\pi_{0.5}$ at the same events, sharing observations, initializations, and seeds. Rescue and harm are paired against the frozen base, hence undefined for the base policies (–).

Method	Atomic			Composite		
	Success (%) ↑	Rescue ↑	Harm ↓	Success (%) ↑	Rescue ↑	Harm ↓
Frozen base policies						
π_0	69.4	–	–	19.0	–	–
RLDX	70.8	–	–	29.4	–	–
GR00T N1.5	70.0	–	–	44.0	–	–
$\pi_{0.5}$	73.8	–	–	45.4	–	–
Correction on the primary policy ($\pi_{0.5}$)						
Triggered Replan	74.0	15.7	15.4	47.9	18.3	15.6
Sample-and-Rerank	74.7	14.3	13.4	49.5	17.6	13.5
DualFuture (ours)	78.7	18.2	13.3	50.9	19.1	13.6

5 Experiments

5.1 Simulation Setup

Evaluation benchmark. RoboCasa365 (Nasiriany et al., 2026) is a large-scale simulated household manipulation benchmark centered on kitchen and home environments. The target splits used here include single-skill *atomic* tasks and harder multi-step *composite* tasks. We use RoboCasa365's pretrain-plus-target protocol, in which policies are first pre-trained on 300 tasks with 100 human demonstrations per task, and then post-trained using 500 demonstrations per task. The evaluation is designed to test under held-out layouts, objects, and random seeds.

Baselines. We report four frozen policies—π_0 (Black et al., 2025), $\pi_{0.5}$ (Physical Intelligence et al., 2025), RLDX (Kim et al., 2026), and GR00T N1.5 (NVIDIA, 2025) that are pretrain-plus-posttrained. The full trajectories and branched pairs for DualFuture pipeline training are sampled from $\pi_{0.5}$. For comparisons at matched event points, we also contrast DualFuture with Triggered Replan, which re-queries the frozen VLA, and Sample-and-Rerank, which executes the best chunk among K candidates scored by goal-action agreement measured by the distance between the two modes of the latent predictor. To ensure a fair comparison, observations, snapshots, initializations, and sampling seeds are shared across methods.

Metrics. Our primary metric is task *success*. Since DualFuture only edits a frozen policy, we run every corrected episode against a base episode from the same initialization and report the two outcomes that the pair separates: a *rescue* rate R, the base policy fails and DualFuture solves the task, and a *harm* rate η, the base policy solves it and DualFuture breaks it.

Implementation details. Details of the models, optimization, limits, and hardware are included in the appendix.

5.2 Main Simulation Results

Table 1 situates $\pi_{0.5}$ among frozen policies on atomic and composite tasks and corrects only $\pi_{0.5}$. Attaching DualFuture to $\pi_{0.5}$ raises atomic success from 73.8% to 78.7%. The improvement is larger on composite tasks, where success increases from 45.4% to 50.9%, or 5.5 percentage points. Here, the corrector achieves 19.1% rescue with 13.6% harm, yielding a positive rescue–harm margin of 5.5 points. The larger net margin on composite tasks supports the intended division of labor. The frozen VLA continues to determine the long-horizon task structure, while DualFuture intervenes locally when the proposed action chunk is inconsistent with the desired near-term outcome. The correction

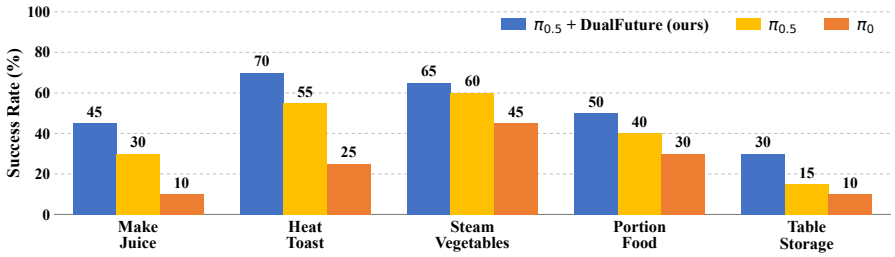

Figure 3: Real-world success rate on a Franka platform, five contact-rich kitchen tasks $\times$ 20 trials with randomized object placements. Two frozen base policies establish the range of base behavior; DualFuture is attached to $\pi_{0.5}$ without fine-tuning any VLA parameter, and shares each trial's initialization with it so that rescue and harm are counted per pair.



Figure 4: Representative real-robot execution sequences for four contact-rich tabletop tasks: Make Juice, Heat Toast, Steam Vegetables, and Portion Food. The examples illustrate the varied object configurations and manipulation stages used in our physical-robot evaluation.

baselines provide a controlled comparison because they operate on the same primary policy and at the same correction events. Relative to Sample-and-Rerank, DualFuture improves success by 4.0 and 1.4 points on atomic and composite tasks, respectively; relative to Triggered Replan, the gains are 4.7 and 3.0 points. Moreover, DualFuture obtains the highest rescue on both splits and the lowest harm on atomic tasks. This combination is important: a useful corrector must generate actions that can repair failures while preserving the frozen policy outside triggered events.

5.3 Event-Trigger Quality

Table 2: Event-trigger quality on held-out RoboCasa365 rollouts. Ground-truth events come from applying the persistence rule of Eq. 18 offline to the executed rollout; a fired trigger counts as a true positive when it matches a ground-truth gripper transition within ± 50 steps, and the cooldown admits at most one trigger per event. For atomic and composite tasks, we train separate $\mathcal{T}_{\text{grip}}$ instances and evaluate each on held-out rollouts from the corresponding task category.

Unseen split	**Recall** $\uparrow$	**Precision** $\uparrow$	**F1** $\uparrow$
Atomic	76.83	68.62	72.49
Composite	74.46	62.37	67.88

Table 2 evaluates $\mathcal{T}_{\text{grip}}$ against the event rule in Eq. 18 on held-out rollouts. Misses forgo correction opportunities, whereas false positives invoke unnecessary corrections. We train separate triggers on the corresponding atomic and composite training rollouts and evaluate each on held-out rollouts from the same task family. Recall decreases from 76.83% on atomic tasks to 74.46% on composite tasks, indicating that event localization is more challenging in longer multi-skill rollouts and partly limits the composite-task improvement.

Table 3: Corrector input ablations on paired RoboCasa365 *composite* unseen episodes. Check marks indicate which inputs of Eq. 12 the corrector receives: the task-consistent *Goal Future*, the *Induced Future* of the frozen proposal, the normalized *State* and the direct *Current Latent*. The first row is full DualFuture; each remaining row discards one input.

Corrector Inputs				**Composite**
Goal Future	Induced Future	State	Current Latent	Success ↑
✓	✓	✓	✓	50.9%
✗	✓	✓	✓	45.5%
✓	✗	✓	✓	45.4%
✓	✓	✗	✓	46.3%
✓	✓	✓	✗	47.3%

5.4 Ablations

Table 3 evaluates the value of one Eq. 12 input at a time on the harder composite unseen split by not using it as an input to the corrector. The ablations cover the goal future, induced future, normalized state, and direct current latent. In the current-latent mask, WM-derived future latents are still computed from the same observation. Masking either future causes the largest degradation, reducing success from 50.9% to 45.5% without the goal future and 45.4% without the induced future. Removing the normalized state or direct current latent has a smaller effect, yielding 46.3% and 47.3%, respectively. This suggests that the difference between the two predicted futures provide the primary correction signals, while state and current-scene features offer complementary context.

5.5 Real-World Evaluation

We finally evaluate transfer on a Franka platform following the DROID hardware design (Khazatsky et al., 2024), with two Zed 2 stereo Cameras, a Zed Mini Wrist Stereo Camera, a Franka Panda 7DoF Robot Arm and Robotiq 2F-85 Gripper, and the same controller abstraction used in simulation. Oculus Quest 2 provides 100 demonstrations for each of five contact-rich tabletop tasks: Make Juice, Heat Toast, Steam Vegetables, Portion Food, and Table Storage. Representative execution sequences are shown in Figure 4. We post-train and then freeze the base policy. We then roll out the frozen $\pi_{0.5}$ on hardware and collect 50 successful trajectories per task, detect events, sample action chunks from the reset states, predict the resulting futures using the trained DFLP and train DualFuture on these rollouts to correct it (Sec. 4.1). We evaluate each task using 20 trials with randomized object placements, correcting the policy at event triggers. Figure 3 reports per-task success.

6 Limitations and Future Work

DualFuture currently uses gripper opening and closing transitions as critical events because they provide discrete, task-relevant intervention points for many contact-rich manipulation tasks. This design does not cover failures that arise away from a gripper transition, such as inaccurate reaching, poor object alignment, collisions, or errors during sustained transport. Consequently, the corrector cannot intervene in every failure mode of the underlying policy.

Future work could extend the trigger and correction mechanism to a broader set of manipulation events. This would require modeling events as a broader range of possible failure scenarios and a more comprehansive detecting mechanism, together with safeguards that prevent frequent or unnecessary interventions. Evaluating such extensions across additional robots, tasks, and interaction modes would clarify how broadly the dual-future correction principle applies.

AI USE STATEMENT

We used AI tools primarily to support manuscript preparation and literature exploration. Writing assistance included refining wording, improving clarity and readability, and polishing the organization and presentation of the text. We also used AI to aid literature retrieval and discovery by identifying potentially relevant studies and locating sources for further examination. We did not use AI assistance to formulate the research ideas or conduct the experiments. We reviewed all AI-assisted work and take responsibility for the final content, including text, claims, and artifacts produced with AI assistance.

REFERENCES

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, François Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183, 2023.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, Deli Zhao, and Hao Chen. WorldVLA: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

Guo Chen, Zhiqi Li, Shihao Wang, Jindong Jiang, Yicheng Liu, Lidong Lu, De-An Huang, Wonmin Byeon, Matthieu Le, Max Ehrlich, Tong Lu, Limin Wang, Bryan Catanzaro, Jan Kautz, Andrew Tao, Zhiding Yu, and Guilin Liu. Eagle 2.5: Boosting long-context post-training for frontier vision-language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 101161–101184, 2025a. doi: 10.52202/085713-3047.

Haonan Chen, Jingxiang Guo, Bangjun Wang, Tianrui Zhang, Xuchuan Huang, Boren Zheng, Yiwen Hou, Chenrui Tie, Jiajun Deng, and Lin Shao. Goal-VLA: Image-generative VLMs as object-centric world models empowering zero-shot robot manipulation. *arXiv preprint arXiv:2506.23919*, 2025b.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation*, pp. 2786–2793, 2017. doi: 10.1109/ICRA.2017.7989324.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12622–12635, 2023.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation*, pp. 6023–6029, 2019.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Srirama, Lawrence Chen, Kirsty Ellis, Peter Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Ma, Patrick Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Walke, Blake Wulfe, Ted Xiao, Jonathan Yang, Arefeh Yavary, Tony Zhao, Christopher Agia, Rohan Baijal, Mateo Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Foster, Jensen Gao, David Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems XX*, 2024. doi: 10.15607/RSS.2024.XX.120.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, Beomjun Kim, Byungjun Yoon, Changsung Jang, Daewon Choi, Dongsu Han, Donguk Lee, Heeseung Kwon, Hojin Jeon, Jaehyun Kang, Jaekyoung Bae, Jihyuk Lee, Jimin Lee, John Won, Joonwoo Ahn, Junhyeong Park, Junyoung Sung, Kyungmin Lee, Minseong Han, Minsung Yoon, Sejune Joo, Seonil Son, Seungcheol Park, Seunggeun Cho, Seungjun Moon, Seungku Kim, Yonghoon Dong, Yongjin Cho, Youngchan Kim, Chang Hwan Kim, Dohyeon Kim, Heecheol Kim, Heewon Lee, Hensen Ahn, Hyungkyu Ryu, Hyunsoo Choi, Hyunsoo Shin, Jaeheon Jung, Jaewoo Kim, Jinwook Kim, Joochul Chang, Joonsoo Kim, Junghun Park, Jungwoo Park, Junho Cho, Junhyeok Park, Junwon Lee, Kangwook Lee, Kwanghoon Kim, Kyoungwhan Choe, Manoj Bhadu, Nayoung Oh, Sangjun Kim, Sangwoo Kim, Seunghoon Shim, Seunghyun Kim, Seungjun Lee, Seungyup Ka, Sungryol Yang, Wook Jung, Yashu Shukla, Yeonjae Lee, Yeonwoo Bae, and Jinwoo Shin. RLDX-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713, 2025.

Chenxuan Li, Jiaming Liu, Guanqun Wang, Xiaoqi Li, Sixiang Chen, Liang Heng, Chuyan Xiong, Jiaxin Ge, Renrui Zhang, Kaichen Zhou, and Shanghang Zhang. A self-correcting vision-language-action model for fast and slow system manipulation. *arXiv preprint arXiv:2405.17418*, 2024.

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

Chenyang Ma, Kai Lu, Guangyu Yang, Jiuming Liu, Shitong Xu, Bill Byrne, Ioannis Havoutis, Niki Trigoni, and Andrew Markham. CycleVLA: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding. *arXiv preprint arXiv:2601.02295*, 2026.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation*, pp. 7559–7566, 2018.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

NVIDIA. NVIDIA Isaac GR00T N1.5. NVIDIA Isaac GR00T release documentation, 2025. URL https://github.com/NVIDIA/Isaac-GR00T/tree/n1d5.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Jim Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive inverse dynamics models are scalable learners for robotic manipulation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=meRCKuUpmc.

Yilin Wu, Thomas Tian, Gokul Swamy, and Andrea Bajcsy. From foresight to forethought: VLM-in-the-loop policy steering via latent alignment. In *Robotics: Science and Systems XXI*, 2025. doi: 10.15607/RSS.2025.XXI.076.

Wenke Xia, Ruoxuan Feng, Dong Wang, and Di Hu. Phoenix: A motion-based self-reflection framework for fine-grained robotic action correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6981–6990, 2025. doi: 10.1109/CVPR52734.2025.00655.

Shaoqing Xu, Fang Li, Zhixiang Duan, Yifan Yang, Tianshi Xie, and Zhi-Xin Yang. VLA-in-the-Loop: Online policy correction with world models for robust robotic grasping. OpenReview, 2026. URL https://openreview.net/forum?id=aT4LG8c6DE.

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. doi: 10.1016/j.eswa.2026.131742.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, Avnish Narayan, You Liang Tan, Guanzhi Wang, Qi Wang, Jiannan Xiang, Yinzhen Xu, Seonghyeon Ye, Jan Kautz, Furong Huang, Yuke Zhu, and Linxi Fan. FLARE: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

A Implementation Details

Before introduction, we first define the following quantities:

What the wrapper observes

- s_t observation at time t, the pair $\{I_t^{(v)}, q_t\}$: one RGB frame per camera view $v \in \{\mathrm{L}, \mathrm{R}, \mathrm{W}\}$ (two third-person, one wrist) and proprioception $q_t \in \mathbb{R}^{d_q}$
- l natural-language instruction, fixed within an episode

Actions

- u_{t+k} one command, in $\mathbb{R}^{d_a}$: end-effector pose target, base command, and gripper target concatenated
- H length of a chunk emitted by the VLA
- H_e length of the executable prefix, $H_e \leq H$; the only part of a chunk DualFuture may correct
- a_t^0 frozen VLA proposal $\{u_{t+k}^0\}_{k=0}^{H-1}$; restricted to the prefix it is a matrix in $\mathbb{R}^{H_e \times d_a}$, the layout shared by a_t^1 and Δa_t

Latent futures

- z latent future, in $\mathbb{R}^{3\times 256\times d_z}$: 256 V-JEPA 2 tokens of width d_z per view for the frame reached after H_e steps, normalized token-wise over the feature dimension by ν and stacked across views, $z_t = \mathrm{stack}\left(\bar{z}_t^{(\mathrm{L})}, \bar{z}_t^{(\mathrm{R})}, \bar{z}_t^{(\mathrm{W})}\right), \quad \bar{z}_t^{(v)} = \nu\big(E(I_{t+H_e}^{(v)})\big)$

A hat marks a predicted quantity ($\hat{z}$ against a realized z), superscript 0 the VLA proposal and 1 the corrected candidate, and t indexes a decision step rather than a single control step.

Table 4: Inference-time interface of the three trainable modules. The frozen VLA, the visual encoder E, and the instruction encoder ϕ_l are not listed. All action quantities are restricted to the executable prefix of length H_e; d_a is the per-step action dimension and d_z the per-view latent dimension. *In* and *Out* denote inputs and outputs.

Critical-event trigger $\mathcal{T}_{\mathrm{grip}}$	
In	s_t; l
Out	$\hat{p}_t \in [0,1]$; $\hat{\tau}_t \in \mathbb{R}_{\geq 0}$, then $e_t \in \{0,1\}$
DFLP F_ϕ, goal mode	
In	s_t; $c = \phi_l(l)$
Out	$\hat{z}_t^g \in \mathbb{R}^{3\times 256\times d_z}$
DFLP F_ϕ, consequence mode	
In	s_t; $c = \phi_a(a)$
Out	$\hat{z}_t^a(a) \in \mathbb{R}^{3\times 256\times d_z}$
Residual corrector C_θ	
In	$\hat{z}_t^g$; $\hat{z}_t^{a,0}$; z_t^{cur}; normalized q_t; a_t^0
Out	$\Delta a_t \in \mathbb{R}^{H_e \times d_a}$, then $a_t^1 = \Pi(a_t^0 + \Delta a_t)$

The settings below are shared by all three modules and omitted from Tables 5–7, which list only what differs. All three train with AdamW in BF16 at gradient clipping 1.0, on an $80/10/10$ episode-level split over the same three views. The future predictor and corrector share both frozen encoders: E is a V-JEPA 2 ViT-G/16 emitting a $[256, 1{,}408]$ latent per view, and ϕ_l is a T5-Base. And the predictor uses a Eagle2.5-VL vision encoder and the same T5 encoder. Throughout, $H_e = 8$, $d_a = 12$, proprioception is 16-dimensional, and instructions are truncated to 48 T5 tokens. The trigger trains on four H200s, the DFLP on six, and the corrector on two.

Counting trainable weights only, and so excluding the frozen E and ϕ_l, the DFLP holds 315M parameters against 3.38M for the trigger and 6.30M for the corrector. Almost all of the wrapper's capacity therefore sits in the future predictor.

Table 5: Training hyperparameters of the critical-event trigger $\mathcal{T}_{\text{grip}}$. Under the frozen encoders only the fusion trunk is trained, and the time term is masked to positive samples.

Optimization	
Global batch size	512
Maximum epochs	50
Learning rate	10^{-4}, constant
LR scheduler / warmup	none / none
Weight decay	0.01
Data	
Distance-bucket sampling weights	$0.4/0.2/0.3/0.1$
History length	8
Architecture	
Action transformer layers	2
Fusion transformer layers	2
Fusion hidden dimension	256
Fusion attention heads	8
Dropout	0.1
Trainable parameters	3.38M
Objective and selection	
Regression / trigger / future-opening/closing / countdown-consistency weights	$1.0/0.5/0.2/0.1$
Best-checkpoint metric	val. event F1

A.1 Critical-Event Trigger

Inputs and outputs. The trigger shares no weights with the frozen VLA and so re-encodes its own visual input. Its two output scalars are thresholded at $(p_{\text{e}}, \tau_{\text{e}})$ into the gate e_t. The thresholds are set after training, so the operating point moves without retraining.

Training. Both targets, the event label and the time to event, are read off the same rollouts used everywhere else, so the trigger needs no extra annotation.

Three entries of Table 5 decide whether the module trains at all. Event steps are a small minority, so batches are sampled from four distance-to-event buckets with probabilities $0.4/0.2/0.3/0.1$, and the checkpoint is selected on validation event F1: under uniform sampling the detection term is nearly minimized by $\hat{p}_t \to 0$, the time head then sees too little gradient to calibrate, and validation loss on a negative-dominated split rewards silence. The regression, trigger, future-opening/closing, and countdown-consistency terms are weighted $1.0/0.5/0.2/0.1$, respectively. And only a small trunk trains under the frozen encoders, since the branch data is far smaller than their pretraining and a wider trunk memorizes the anchors instead of the event structure. Nothing else was tuned.

A.2 Dual-Future Latent Predictor

Inputs and outputs. The DFLP is one conditional network with a single input signature $(s_t, c) \mapsto \hat{z}_t$ (Table 4), queried under different conditions. Its state-side input is the *current* visual latents and q_t, whereas the regression target is the frame reached after the executable horizon, $I^{(v)}_{t+H_e}$. The condition slot takes either a frozen T5 instruction embedding $c = \phi_l(l)$ or an action-chunk embedding $c = \phi_a(a)$, both of the same width so one set of weights serves both modes. Since the condition is all that differs between the two queries, the goal and consequence futures land in one comparable latent space by construction.

The output is a single predicted latent, not a distribution. At inference the module runs twice, on $\phi_l(l)$ for the goal future and on $\phi_a(a^0_t)$ for the proposal future; both predictions are inputs to the corrector.

Table 6: Training hyperparameters of the DFLP F_ϕ. One set of weights serves both the goal and the consequence mode; what differs between them is the condition and the dataset, not any entry of this table.

Optimization	
Global batch size	1,536 (Atomic); 1,152 (Composite H8)
Maximum training steps	10,000
Learning rate	7.5×10^{-5} start, 4.25×10^{-4} peak, 0 final
LR scheduler	WSD
Warmup	4,500 steps
Weight decay	0.04
Data	
Training records	full-rollout ACTION / GOAL
Architecture	
Predictor initialization	random
Predictor layers	24
Hidden dimension	1,024
Attention heads	16
Position encoding	3D RoPE
Dropout	0.0
Output latent shape	$[B, 3, 256, 1{,}408]$
Trainable parameters	315M
Conditioning and objective	
Action conditioning	per-step action tokens with state context
Condition protocol	learned null slots + mode token
Rollout steps	2
Direct : rollout loss weight	0.5 : 0.5
Latent loss	L1
Per-view loss weights	$[1.0, 1.0, 1.0]$
Best-checkpoint metric	joint latent L1

Training. Both modes regress the realized latent z_t under the same ℓ_1 loss, differing in the dataset rather than the objective (Sec. 4.2). Both update the shared trunk; of the two condition encoders only ϕ_l is frozen, ϕ_a training jointly with F_ϕ. The module is trained to convergence and frozen before the corrector, so the corrector never sees a moving future model, and the targets are stop-gradient, so no gradient reaches E.

Because the trunk is randomly initialized rather than inherited, the schedule in Table 6 matters more here than for the other two modules: warmup runs close to half of training, and shortening it destabilizes the module. Early on the two conditioning modes disagree about how to move the shared trunk, and reaching the peak rate before that settles collapses the predictor onto whichever mode the batch favored.

Three further entries bear on whether the two modes stay comparable. The network retains fixed action and prompt slots in both modes: the action query uses learned null prompt slots, the goal query uses learned null action slots, and a mode token identifies the query. Rolling the predictor forward on its own output counters error accumulated over H_e, which spans several control steps, as a bias shared by both futures. Both modes are trained from full-rollout ACTION and GOAL records.

A.3 Dual-Future Residual Correction

Inputs and outputs. The two futures carry the discrepancy to be closed, while the current visual latent z_t^{cur}, normalized proprioception q_t, and the proposal a_t^0 supply the context that turns a latent difference into an executable action difference (Table 4).

Table 7: Training hyperparameters of the residual corrector C_θ. The continuous loss weights the three normalized action groups equally and averages uniformly over the executable horizon.

Optimization	
Global batch size	512
Maximum epochs	32
Learning rate	10^{-4}
LR scheduler	constant
Warmup	none
Weight decay	0.01
Architecture	
Decoder layers	3
Hidden dimension	256
Attention heads	4
Dropout	0.1
Action–goal residual stream	enabled
Trainable parameters	6.30M
Objective and selection	
Continuous loss	group-normalized Smooth-L1, $\beta = 0.05$
Base / EE-position / EE-rotation scales	0.01424 / 0.28499 / 0.07863
Switch BCE weight	0.2
Correction-size weight	0.001
Success no-op weight	0.1
Best-checkpoint metric	val. H8 continuous MAE

Training. The corrector regresses the target residual $\Delta a_t^\star = a_t^+ - a_t^k$ on the branch pairs of Sec. 4.3, an observed difference between the anchor and an alternative chunk executed from the same snapshot. We retain reset groups that pass reset-consistency, reference-completeness, and horizon-validity checks. Both F_ϕ and E stay frozen, so no corrector gradient reaches either.

Training uses the realized goal and consequence latents of each branch pair directly.

The best-checkpoint run uses a constant learning rate of 10^{-4} without warmup. It is trained from scratch with a 32-epoch budget and selected by the lowest validation H8 continuous-action MAE.

Continuous residuals are divided by train-audited scales for the base, end-effector position, and end-effector rotation groups, whose losses are then weighted equally. Two separate discrete heads predict the per-timestep `control_mode` and `gripper_close` logits. The decoder stays small, since it only turns a latent discrepancy into a bounded residual rather than models the scene.

B Simulation Experiments

B.1 Case Study

We inspect individual rollouts to see what the aggregate success numbers of Table 1 look like frame by frame.

Success Case Figure 5 shows ten successful rollouts, the five atomic target tasks first and then five *unseen* compositions. Each row is one episode, read left to right as eight uniformly sampled frames of the `robot0_agentview_left` camera, with the task name and the goal instruction the policy was given printed below the row. The atomic rows are dominated by a single contact event, and the correction is visible as the gripper closing on the handle at the pose the base policy missed. The composite rows instead chain several such events, and what the frames show is that a local edit at one boundary does not disturb the subtask structure around it: the arm finishes the corrected subtask and proceeds to the next target object without regressing on the part already completed.

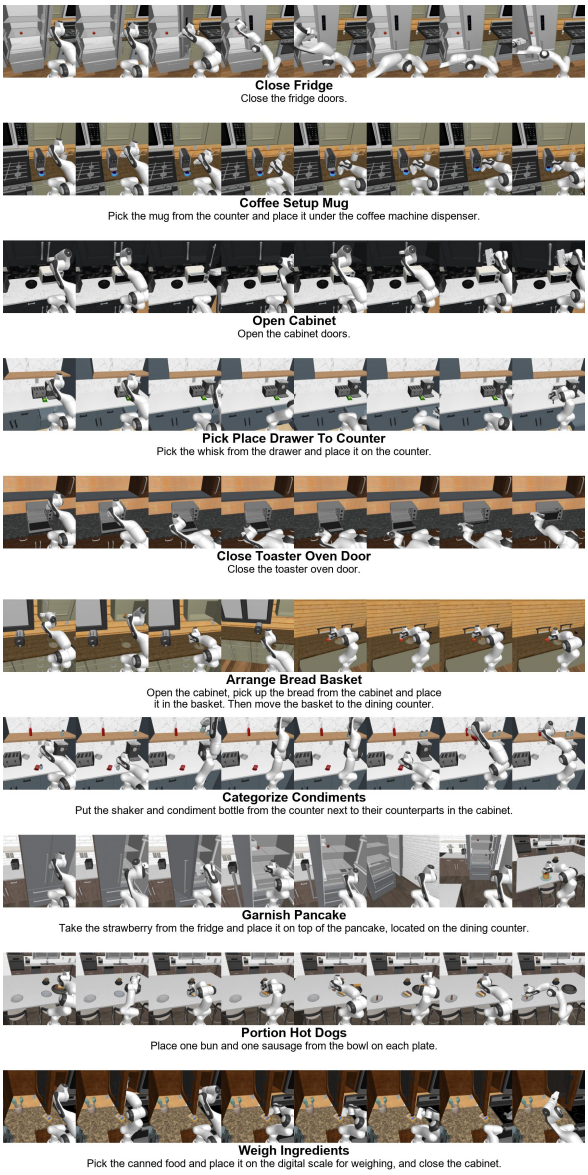

Figure 5: Successful rollouts on the RoboCasa 365 target split with DualFuture attached to a frozen $\pi_{0.5}$. Every row is a single episode shown as eight uniformly spaced frames in temporal order; beneath each row are the task name and the verbatim goal instruction supplied to the policy. Rows one to five are atomic tasks (CLOSEFRIDGE, COFFEESETUPMUG, OPENCABINET, PICKPLACEDRAWERTOCOUNTER, CLOSETOASTEROVENDOOR) and rows six to ten are *unseen* compositions (ARRANGEBREADBASKET, CATEGORIZECONDIMENTS, GARNISHPANCAKE, PORTIONHOTDOGS, WEIGHINGREDIENTS).

Failure Case Figure 6 shows four failure cases, one episode per mode.

- *Unmodelled control mapping.* In TURNOFFSTOVE the arm turns a knob, but not the one wired to the named burner. That mapping is a property of the appliance rather than of the frames, so the manipulation succeeds while the task fails. No gripper transition is imminent anywhere in the rollout, so the trigger stays shut.
- *Missing affordance prior.* TURNONMICROWAVE localizes the control panel but presses the wrong region of it, and which region is the start button is not recoverable from the instruction or a third-person frame. This is again outside DualFuture's reach: the task ends in a press rather than a grasp, so the gripper-event trigger never fires.
- *Wrong referent.* BREADSELECTION transports a pastry cleanly onto the cutting board, but not the croissant the instruction names. Because the grasp and the placement are both well formed, the discrepancy DualFuture closes is the wrong one: goal and consequence futures agree that an object is being moved to the board, and neither encodes which pastry it should have been.
- *Execution timeout.* GARNISHPANCAKE completes the transfer but re-approaches the pancake instead of committing to the placement, exhausting its step budget.

C REAL ROBOT EXPERIMENTS

C.1 EVALUATION SETUP

Table 8 lists the five contact-rich tabletop tasks with the instruction given to the policy and the reference subtask sequence an episode is scored against, spanning one to twelve subtasks. Object placements are randomized between episodes throughout this section: in the demonstrations, in the collected rollouts, and in the evaluation trials.

On the DROID platform of Sec. 5.5, the two third-person cameras and the wrist camera supply the three views $v \in \{\mathrm{L}, \mathrm{R}, \mathrm{W}\}$ that all three modules read. Nothing about the wrapper changes between simulation and hardware apart from where the rollouts come from, and how the corrector's pairs are built without state resets.

C.2 BASE POLICY POST-TRAINING

The base policy is a $\pi_{0.5}$ checkpoint mid-trained on DROID, which we post-train on our own demonstrations before freezing it. Demonstrations are collected by VR teleoperation on the same platform the policy is later deployed on, 100 per task and so 500 in total. Because teleoperation writes into the controller abstraction the policy also emits into, no action relabelling is needed between collection and training.

Post-training runs at a batch size of 64 for $40{,}000$ steps and predicts a chunk of length $H = 8$. Prediction and execution horizons therefore coincide here, $H = H_e = 8$, so the executable prefix is the whole chunk and every command the frozen policy emits is one DualFuture may edit; a fresh chunk is predicted at each decision boundary rather than any part of the previous one being carried over. The resulting policy is frozen at this point and receives no gradient from anything that follows, which is what makes the real-robot numbers a statement about the wrapper rather than about additional policy training.

C.3 ROLLOUT COLLECTION AND MODULE SUPERVISION

We then deploy the post-trained $\pi_{0.5}$ on hardware and collect 50 successful trajectories per task, 250 in total. Each rollout is scored against the reference subtask sequence while it is being collected, and only those completing the whole sequence are retained. These 250 are the only supervision the three modules receive here; no corrective teleoperation is recorded, and the demonstrations of the previous subsection are not reused as module training data.

All 250 go to the DFLP. The asymmetry of Sec. 4.2 is narrower on hardware than in simulation: without state resets there are no executed alternative branches, so the consequence mode sees the

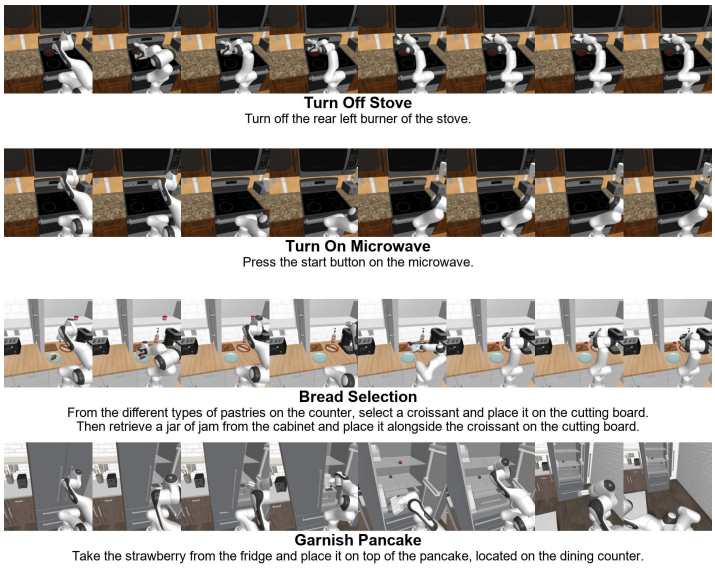

Figure 6: One unsuccessful rollout per residual failure mode, drawn as eight uniformly spaced frames of the `robot0_agentview_left` camera with the task name and the verbatim goal instruction beneath each row. Rows one and two (TURNOFFSTOVE, TURNONMICROWAVE) are the uncorrected atomic failures and rows three and four (BREADSELECTION, GARNISHPANCAKE) the composite harm cases, in the order discussed in the text.

Table 8: Goal instructions and reference subtask sequences for the five real-robot tasks; # is the number of subtasks. The policy receives only the goal instruction, which is fixed for the episode; the numbered lines are a reference decomposition used solely for scoring, not sub-goals supplied to the policy by a planner. An episode counts as successful only when the entire sequence is completed.

Task	#	Goal instruction	Reference subtask sequence
Make juice	4	"*Place the pumpkin in the blender and close the lid.*"	1. Pick the pumpkin from the table 2. Place the pumpkin into the blender 3. Pick the blender lid 4. Put the lid on the blender
Heat toast	1	"*Put the toast into the toaster.*"	1. Place the toast into the toaster
Steam vegetables	5	"*Put the carrot and the vegetable into the electric pot and close the lid.*"	1. Pick the vegetable from the cutting board 2. Place the vegetable into the electric pot 3. Pick the carrot from the cutting board 4. Place the carrot into the electric pot 5. Close the lid
Portion food	8	"*Place one piece of vegetable and one piece of bread from the bamboo basket on each plate.*"	1. Pick the bread roll from the bamboo basket 2. Place the bread roll on the right plate 3. Pick the carrot from the bamboo basket 4. Place the carrot on the left plate 5. Pick another piece of bread from the bamboo basket 6. Place the bread on the left plate 7. Pick the cucumber from the bamboo basket 8. Place the cucumber on the right plate
Table storage	12	"*Put the toys in the bamboo basket, put the snacks and bread in the white box, and put the vegetables on the plate.*"	1. Pick the plush toy from the table 2. Place the plush toy in the bamboo basket 3. Pick the toy car from the table 4. Place the toy car in the bamboo basket 5. Pick the packaged food from the table 6. Place the packaged food in the white box 7. Pick the cucumber from the table 8. Place the cucumber on the plate 9. Pick the bitter gourd from the table 10. Place the bitter gourd on the plate 11. Pick the bread roll from the table 12. Place the bread roll in the white box

recorded chunks and their realized futures rather than a neighbourhood around them, and the two modes differ in their condition more than in their coverage. The trigger reads the same trajectories, and one detail of its negatives matters: they are windows inside those successful rollouts that contain no imminent persistent gripper transition. This keeps the negative label meaning what the labelling rule makes it mean, the absence of an event within the lookahead window rather than the presence of a bad outcome.

Corrector pairs without snapshot resets. The branch construction of Sec. 4.1 assumes a restorable state, which hardware does not give us, so the corrector's pairs are built by offline replay instead. We replay each retained rollout, and at the event snapshots found in it we query the frozen VLA on the recorded observation to obtain a proposal chunk a_t^k, drawing 4 samples per snapshot with independent seeds. The recorded chunk is the anchor a_t^+, so the target residual $\Delta a_t^\star = a_t^+ - a_t^k$ is still an exact difference between two action chunks rather than anything inferred from F_ϕ, and the feasibility filter on Π's envelope is applied unchanged.

What replay cannot supply is the realized consequence of a_t^k, since that chunk is never executed. The source future is therefore the action-conditioned prediction $\hat{z}_t^a(a_t^k)$ from the DFLP, whereas the destination future is encoded directly from the observation the successful rollout actually reached after H_e steps. This reverses which side of the pair carries the substitution gap. The consequence input is already a prediction at training time and so already matches deployment, and it is the goal input that is a realized latent here; accordingly only that side is perturbed with errors drawn from the held-out ϵ_g distribution. Both E and the DFLP stay frozen while the corrector trains, as in simulation, so the predicted source future is a fixed input and not a path for corrector gradients to reach the future model.