DREAMAVOID: CRITICAL-PHASE TEST-TIME DREAMING TO AVOID FAILURES IN VLA POLICIES

Anonymous authors
Paper under double-blind review

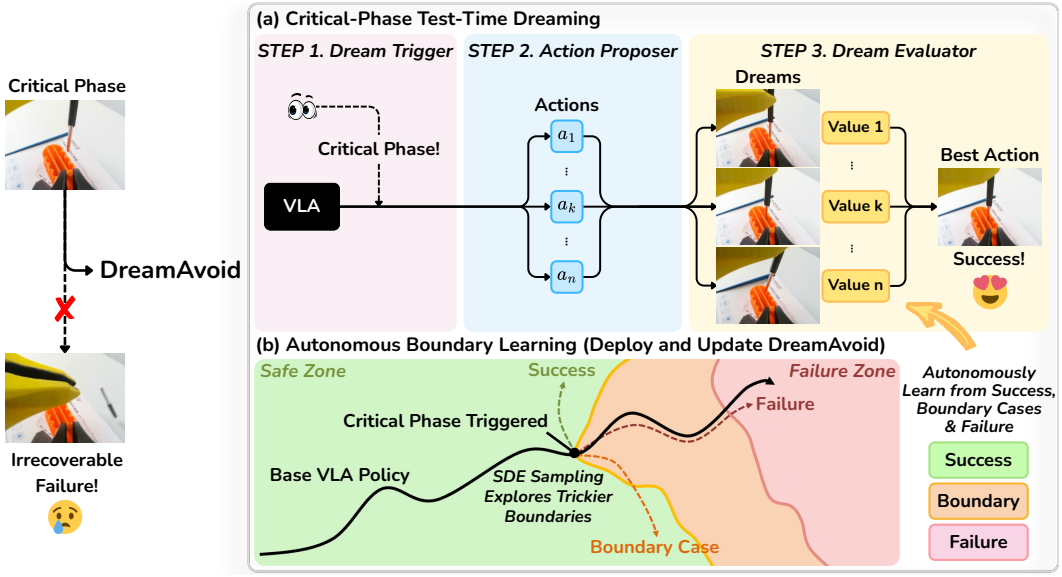


Figure 1: We propose DREAMAVOID, (a) a *critical-phase test-time dreaming* framework that enables the VLA to anticipate and avoid failure. This framework consists of a Dream Trigger, an Action Proposer, and a Dream Evaluator. (b) We also introduce an *autonomous boundary learning* paradigm to equip the world-model-based Dream Evaluator with failure awareness.

ABSTRACT

Vision-Language-Action (VLA) models are often brittle in fine-grained manipulation, where minor action errors during the critical phases can rapidly escalate into irrecoverable failures. Since existing VLA models rely predominantly on successful demonstrations for training, they lack an explicit awareness of failure during these critical phases. To address this, we propose DREAMAVOID, a critical-phase test-time dreaming framework that enables VLA models to anticipate and avoid failures. We also introduce an autonomous boundary learning paradigm to refine the system's understanding of the subtle boundary between success and failure. Specifically, we (1) utilize a Dream Trigger to determine whether the execution has entered a critical phase, (2) sample multiple candidate action chunks from the VLA via an Action Proposer, and (3) employ a Dream Evaluator, jointly trained on mixed data (success, failure, and boundary cases), to "dream" the short-horizon futures corresponding to the candidate actions, evaluate their values, and select the optimal action. We conduct extensive evaluations on real-world manipulation tasks and simulation benchmarks. The results demonstrate that DREAMAVOID can effectively avoid failures, thereby improving the overall task success rate. Across four real-world tasks, DREAMAVOID achieves 72.5% success, compared with 48.8% for the base policy and 54.4% for GPC-RANK. Our code is available at https://anonymous.4open.science/r/DreamAvoid.

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.

1 INTRODUCTION

Vision-Language-Action (VLA) models provide a promising framework for general-purpose robotic manipulation by directly mapping multimodal observations and language instructions into the action space (Kim et al., 2025; Black et al., 2025a;b). However, they remain fragile in fine-grained manipulation scenarios. In tasks such as insertion, precise alignment, and contact-sensitive placement, a small local action error can rapidly escalate into an irrecoverable failure (Xu et al., 2026; Zhang et al., 2026; Hu et al., 2025). In many cases, the policy does not perform poorly over the entire trajectory; rather, failures are often triggered by brief yet low-tolerance *critical phases* (Onishi et al., 2026). How can we equip VLAs with stronger consequence-awareness during these critical phases? In human cognition, complex physical tasks are often balanced between fast, intuitive execution and slow, deliberate foresight. When threading a needle or plugging in a cord, we do not consciously plan every minute movement; we rely on intuitive motor priors. However, at critical moments, we briefly pause and simulate in our minds, "dreaming" of the physical consequences our actions are about to bring, to avoid making mistakes.

Inspired by this, a natural choice is to spend extra test-time computation so that the VLA can "dream" before acting. Powerful large-scale pre-trained world models (Gao et al., 2026) can help capture complex environmental dynamics and generate high-fidelity future simulations based on candidate actions. Can we leverage this capability to achieve better consequence-awareness for robots through dreaming at test-time? We conducted a pilot study on a charger plugging task, comparing two test-time compute paradigms: (1) Direct Semantic Evaluation, which uses a VLM to predict success or failure by receiving past and current visual observations and future joint-space or end-effector-space numerical action sequences; and (2) Explicit Spatiotemporal Dreaming, which uses a large-scale pre-trained action-conditioned world model to explicitly forward-render actions into future visual frames, which are then evaluated by the same VLM. The results showed that when faced with abstract numerical inputs, the VLM tends towards a conservative "almost certain to fail" prediction pattern, whereas its accuracy improves to 80.0% when evaluating the visual simulations provided by the world model. This contrast reveals an insight: for fine-grained manipulation, visual consequence prediction can help bridge the gap between abstract actions and physical consequences. Therefore, rendering actions into high-fidelity future frames for explicit test-time dreaming is a promising approach.

Following the idea of introducing extra computation at test-time, several studies have proposed their own solutions. RoboMonkey utilizes a VLM as an online verifier to score and filter candidate actions (Kwok et al., 2025). GPC and Visual MPC evaluate candidate actions through predicted consequences, while VLA-Reasoner performs online tree search (Qi et al., 2026; Finn & Levine, 2017; Guo et al., 2025). On the other hand, some research achieves dynamic scaling of test-time compute through implicit iterative reasoning within a continuous latent space (Tur et al., 2026). However, implementing test-time computation in robotic manipulation tasks still faces two challenges. (1) Always-on inference can be costly (Yang et al., 2026); executing heavy rollouts at every replan incurs additional computation even during routine execution (Tur et al., 2026; Guo et al., 2025). This motivates selective intervention that concentrates computation on consequential decisions while retaining the base policy during routine execution. (2) Existing VLAs typically rely on successful trajectories for training (Kim et al., 2025). Such training teaches policies to imitate successful behavior but provides limited supervision on unsuccessful and near-boundary outcomes (Black et al., 2025b; Wu et al., 2025). This deficiency is particularly prominent during critical phases, where the robot needs to distinguish between subtle action differences that carry significant consequences.

To address these challenges, we propose DREAMAVOID, a critical-phase test-time dreaming framework designed to enhance the ability of VLAs to avoid failures (Figure 1). We also introduce an autonomous boundary learning paradigm. Specifically, (1) DREAMAVOID proposes a selective intervention mechanism. During routine execution, the robot follows the frozen base policy. When a lightweight Dream Trigger detects an impending critical phase, the system invokes an Action Proposer to sample multiple diverse candidate action chunks. Subsequently, a world-model-based Dream Evaluator engages in short-horizon dreaming to predict their future visual evolution and values, selecting the highest-scoring candidate. (2) DREAMAVOID proposes an autonomous boundary learning paradigm, introducing failure and boundary data from which the world models can benefit. For the dreams to truly serve a preventive purpose, the world model must possess the ability to have "nightmares." If trained solely on success data, the world model may generate optimistic hallucinations due to a lack of negative boundary constraints. By deploying DREAMAVOID to explore

and collect success, failure, and boundary data, we enable the world model to autonomously learn the boundary between success and failure.

We evaluate DREAMAVOID across multiple real-world tasks that require precise alignment, stable contact, and effective failure avoidance. Additionally, we conduct simulation benchmarking on SimplerEnv (Li et al., 2025) and RoboTwin 2.0 (Chen et al., 2026b). The experimental results demonstrate that DREAMAVOID improves aggregate task success rates compared to base VLA policies. These findings suggest that test-time compute can be useful without being always-on; when it is invoked during critical phases, world-model-based test-time dreaming can help correct local errors before failures occur, thereby enhancing the reliability of VLAs.

Our contributions are threefold: (1) We emphasize the focus on critical phases and propose DREAMAVOID, a critical-phase test-time dreaming framework that helps VLAs foresee and avoid failures in advance. This framework keeps the base policy frozen and consists of a Dream Trigger, an Action Proposer, and a Dream Evaluator. (2) We propose an autonomous boundary learning paradigm that endows the world models with failure awareness by introducing failure and boundary data. (3) We provide evidence for the advantages of explicit spatiotemporal dreaming in failure prediction through a pilot study. We conduct extensive evaluations on real-world manipulation tasks and simulation benchmarks. The results show that DREAMAVOID can effectively intervene before minor execution errors deteriorate into irrecoverable failures, improving overall task success rates.

2 RELATED WORK

Test-Time Compute. Existing methods improve VLA robustness through online adaptation (Liu et al., 2026; Bai et al., 2025), recurrent reasoning (Tur et al., 2026), or candidate evaluation with world models and verifiers (Qi et al., 2026; Kwok et al., 2025). DREAMAVOID concentrates additional computation at predicted critical phases, using a large-scale pretrained world model for failure-aware short-horizon dreaming. Unlike the pixel-level goal scoring in Visual MPC (Finn & Levine, 2017), DREAMAVOID assesses potential failures in predicted futures. Autonomous boundary learning further refines predictions near success–failure transitions using deployment-collected data, supporting preventive action selection. A comprehensive discussion of related work is provided in Appendix A.

3 PILOT STUDY: CAN DREAMING IMPROVE FAILURE PREDICTION IN CRITICAL PHASES?

Table 1: Pilot study on failure prediction for fine-grained manipulation (Charger Plugging). All conditions use Gemini 3.6 Flash on the same 40 episodes. Brackets give Wilson 95% confidence intervals for accuracy.

Action representation	Accuracy (%)	Failure precision	Failure recall
Joint-space text	57.5 [42.2, 71.5]	56.4%	100.0%
EEF-space text	67.5 [52.0, 79.9]	64.5%	90.9%
DreamDojo video	80.0 [65.2, 89.5]	88.9%	72.7%

We first explore the paradigm of predicting fine-grained manipulation failures at test time. Given the strong semantic reasoning capabilities of VLMs, an intuitive baseline is to directly use them as consequence predictors during test time. We conducted a pilot study on the Charger Plugging task, which is highly sensitive to minor deviations. We constructed a test set containing 40 episodes (18 successes and 22 failures), which were collected by rolling out a base policy $\pi_{0.5}$ (Black et al., 2025a) trained on 100 human teleoperation demonstrations in the real world. To accurately evaluate the model's reasoning ability during the critical phase, we located a key interaction frame t_{crit} within each test trajectory. This moment is defined as a frame within the 1-2 s immediately preceding the end-effector's decisive physical contact with the target object (e.g., just before the plug aligns with the socket). At each t_{crit}, we extract the current frame o_t and a historical observation sequence. The action input is the downsampled future action chunk u_t.

We compare two test-time failure prediction paradigms: (1) Direct Semantic Evaluation: We input the visual observation sequence containing the historical and current frames, along with the textified

joint-space or end-effector (EEF)-space action sequence and its semantic description, into Gemini 3.6 Flash (Google DeepMind, 2026). We use few-shot prompting (Brown et al., 2020) for it to predict end-to-end whether executing u_t will result in a task failure (such as a collision or misalignment). (2) Explicit Spatiotemporal Dreaming: We use DreamDojo (Gao et al., 2026), an action-conditioned, large-scale pre-trained world model, to forward-render u_t into future video frames, which are then evaluated by Gemini 3.6 Flash acting as a visual evaluator. Details are provided in Appendix B.

The experimental results reveal the differences in prediction reliability between the two paradigms (Table 1). The accuracy of the Direct Semantic Evaluation paradigm with joint-space text is only 57.5%. Analysis shows that when faced with abstract numerical inputs, the VLM tends toward a conservative "almost always predict failure" pattern: it successfully identified all 22 failure samples, but also misclassified 17 successful samples as failures. Although the VLM can understand the semantic goal of "Plug the charger into the socket," it struggles to establish an effective connection between the raw numerical action chunk and its actual physical consequences in this setting. In fine-grained manipulation, even a tiny deviation in the action trajectory can lead to catastrophic collisions. However, these minuscule numerical differences are challenging for the VLM to interpret reliably, limiting its ability to accurately predict the physical evolution of the trajectory. Although the EEF-space text representation improves accuracy from 57.5% to 67.5%, notable limitations remain. Conversely, Explicit Spatiotemporal Dreaming demonstrated stronger predictive capabilities, achieving an accuracy of 80.0%. Although optimistic in some cases (missing 6 failures), it drastically suppressed false alarms (misclassified success cases dropped from 17 to 2). By explicitly simulating numerical actions forward into visual futures, the world model serves as a critical spatiotemporal bridge. It translates abstract numerical errors into explicit visual deviations, for example, directly dreaming of the plug hitting the edge of the socket or the tabletop. Subsequently, Gemini 3.6 Flash can assess this visual anomaly from the generated high-fidelity frames. This pilot study supports explicit spatiotemporal dreaming as a useful way to assess failure risks in critical phases.

4 DREAMAVOID

Based on the insights from the pilot study (§3), we propose DREAMAVOID, a critical-phase test-time dreaming framework for VLA manipulation. The core idea is that the robot directly executes the frozen base VLA policy during most time steps, and only triggers test-time dreaming when it predicts an imminent transition into a critical phase. DREAMAVOID consists of three modules: Dream Trigger, Action Proposer, and Dream Evaluator. DREAMAVOID allocates compute-intensive failure-aware reasoning to the truly important moments. Algorithm 1 summarizes execution.

Problem Formulation. We consider a trajectory $\tau = \{(o_t, l, a_t)\}_{t=1}^{T}$, where o_t denotes the visual observation at time t, l is the language instruction, and a_t is the robot action. Parameterized via **flow matching**, the base VLA policy π_θ predicts an action chunk of horizon H, $u_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}]$, such that $u_t \sim \pi_\theta(\cdot \mid o_t, l)$.

4.1 DREAM TRIGGER

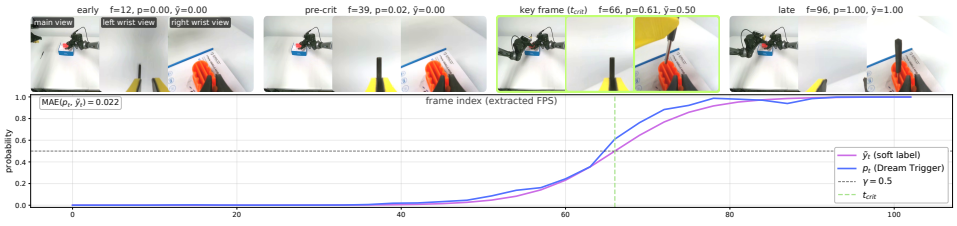


Figure 2: Qualitative example of the Dream Trigger ($\beta = 5$ frames in the soft label $\tilde{y}_t$).

The first step of DREAMAVOID is to determine whether the current state has entered a critical phase. We introduce a lightweight Dream Trigger $p_t = f_\phi(\mathcal{O}_{t-K+1:t}, s_t)$, whose inputs are (1) single- or multi-view observations from the recent K frames $\mathcal{O}_{t-K+1:t} = \{o_{t-K+1}, \ldots, o_t\}$, and (2) the current proprioceptive state s_t. Its output $p_t \in [0, 1]$ is an onset-detection score. We implement the trigger with a frozen DINOv2 encoder and a three-layer MLP. After collecting teleoperation

demonstrations, we use Gemini 3.6 Flash offline to annotate two timestamps for each designated interaction: the critical-phase onset t_{crit} and the associated decisive-contact time t_{contact}. The onset is identified within the 1–2 s preceding this task-defined contact. We define the reference pre-contact intervention window as $\mathcal{I} = [t_{\text{crit}}, t_{\text{contact}})$. The onset provides training supervision, while the interval provides an offline reference for evaluating detection timing. Trigger training uses the hard target $y_t = \mathbb{I}[t \geq t_{\text{crit}}]$. To smooth the transition around an uncertain annotated onset, we replace it with $\tilde{y}_t = \text{Sigmoid}((t - t_{\text{crit}})/\beta)$, where timestamps are expressed as video-frame indices and β controls the transition scale. This target represents whether the onset has been passed and therefore remains high after contact. The deployment rule below controls whether a new intervention is issued. The Dream Trigger is supervised using binary cross-entropy with class-balanced weights: $\mathcal{L}_{\text{trigger}} = -\frac{1}{T}\sum_{t=1}^{T}\left[w_+ \tilde{y}_t \log p_t + (1 - \tilde{y}_t)\log(1 - p_t)\right]$. The weight $w_+ = N_-/N_+$ balances pre-onset and post-onset frames, where N_- and N_+ are computed from the hard targets. For each real-world task, we use 100 training videos containing approximately 26,000–34,000 frames. Frame-level targets are constructed from the annotated onset of each supervised stage using the soft-label rule above. At each policy decision when the evaluator is idle, we combine the trigger score with a persistent stage flag $d_t \in \{0, 1\}$, initialized to zero at rollout reset. The intervention condition is $c_t^{\text{test}} = (1 - d_t)\mathbb{I}[p_t \geq \gamma]$. If $c_t^{\text{test}} = 0$, the system executes the base-policy action chunk. Otherwise, the scheduler sets $d_t = 1$ and launches one cycle of action proposal, candidate evaluation, and selection. After executing the selected chunk, control returns to the same decision rule. The flag remains set until the rollout resets, preventing repeated calls for the same stage even if p_t remains high. For multi-stage tasks, each designated stage has its own prediction head and flag, allowing later stages to trigger independently. The full scheduling rule is given in Appendix C.1. We use $\beta = 5$ frames and $\gamma = 0.5$, with asynchronous trigger polling at 1 Hz and serialized evaluator calls. Online decisions use causal visual observations, proprioception, and stage flags. Detector events and actual evaluator invocations are counted separately (Appendix C.1.1).

4.2 Action Proposer

When the system determines that it is currently in a critical phase, DreamAvoid does not discard the base VLA policy, but instead continues to use it as a generator for candidate actions. In this paper, the base policy is modeled using flow matching, progressively mapping noise into an action chunk by integrating a conditional velocity field within the action space. During standard inference, this generation process follows a probability flow ODE that is deterministic given the observation, language instruction, and initial noise; different initial noise samples can yield different action chunks. To diversify candidate action chunks during testing, DreamAvoid injects additional stochastic noise during ODE integration (Liu et al., 2025; Chen et al., 2026a). Standard flow matching inference corresponds to the following probability flow ODE: $\mathrm{d}u = v_\theta(u, \rho \mid o_t, l)\,\mathrm{d}\rho$, where $\rho \in [0, 1]$ denotes the internal flow matching time variable, and v_θ is the conditional velocity field that determines the deterministic evolution trajectory of the action chunk from the initial noise distribution to the target distribution. To support multi-candidate sampling during testing, we add noise during integration to obtain the following stochastic process: $\mathrm{d}u = v_\theta(u, \rho \mid o_t, l)\,\mathrm{d}\rho + \sigma(\rho)\,\mathrm{d}w_\rho$, where $\sigma(\rho)$ controls the sampling noise intensity, and w_ρ denotes the Wiener process. In practice, we adopt a constant noise scheduling strategy. The specific values of σ for different tasks can be found in Table 5, Appendix C.2. By repeating this stochastic sampling process, we can generate multiple candidate action chunks for the same time step to be used for the Dream Evaluator.

This design serves two primary purposes. First, all candidate actions still originate from the velocity field of the same pre-trained policy; therefore, they are not blind searches in an unconstrained action space, but are guided by the task semantics and action priors encoded in the base policy. Second, the randomness introduced by SDE allows the system to expose multiple locally plausible action hypotheses by perturbing the sampling trajectory beyond its initial noise. This is particularly crucial during critical phases: multiple seemingly reasonable action chunks might lead to drastically different downstream outcomes. For example, one candidate might accomplish precise alignment, while another might cause lateral deviation, collision, or object slipping.

4.3 Dream Evaluator

Action-conditioned world models provide a natural mechanism for evaluating the subsequent effects of candidate actions. DreamDojo (Gao et al., 2026), which is pre-trained on large-scale videos

and post-trained on robotics data, provides rich physical priors of manipulation tasks for test-time dreaming. Specifically, for each candidate action chunk $u_t^{(k)}$ and current observation o_t, we use the distilled DreamDojo (Appendix C.3.1) to predict its corresponding future observation sequence $\hat{y}_t^{(k)} = \mathcal{W}_\psi(o_t, u_t^{(k)})$. We define the value of a candidate action chunk as the estimated progress delta it brings within the short-horizon window. Built upon a VLM trained on a large-scale dataset, Robometer (Liang et al., 2026) acts as a general-purpose robotic reward model capable of predicting frame-level task progress $\eta \in [0, 1]$. This allows us to automatically compute the offline progress pseudo-labels $z_t^{(k)}$ for the corresponding executed trajectories (Appendix C.3.2). We use $z_t = \eta_{t+H} - \eta_t$, with $z_t = -1$ for irreversible failures. During the training of the value model, the model needs to learn accurate physical consequences from clear, real-world videos. However, during test time, it only receives video prediction $\hat{y}_t^{(k)}$ generated by DreamDojo, which may contain blurriness or artifacts. To balance learning efficiency and overcome this domain shift, we employ a *real-dream joint training* strategy.

When constructing the inputs for the task-specific value model, (1) we directly input the *real future sequence* y_t^{true} into the model (fitting $V_\omega(o_t, u_t, y_t^{\text{true}}) \to z_t$), enabling the network to learn precise value mappings based on observed visual features; (2) we input (o_t, u_t) into DreamDojo to generate *dreams* $\hat{y}_t$ and use them as inputs as well (fitting $V_\omega(o_t, u_t, \hat{y}_t) \to z_t$). By treating the generated sequences as a form of in-domain data augmentation, we encourage the value model to learn to extract crucial physical progress features despite the visual imperfections of DreamDojo. During this process, the supervision targets z_t required for the aforementioned fitting are derived from Robometer scores on the real future trajectories y_t^{true} sampled from the offline dataset, with the irreversible-failure override described above. Furthermore, the vast majority of routine offline teleoperation trajectories are in a state of steady progression (exhibiting small positive progress deltas). Direct random sampling would cause the model to lose its discriminative power for dangerous situations. To address this, DA-ABL uses priority-based oversampling when constructing the training batch $\mathcal{B}$: 20% of the data is sampled from the terminal state horizons of successful completions, 40% is drawn from the additionally collected boundary and failure data (see §4.4, including labeled irreversible failures penalized to $z = -1$), and the remaining 40% comes from routine steady progression phases. We jointly use a Huber robust regression loss (Huber, 1992) and a continuous margin ranking loss:

$$\mathcal{L}_{\text{VE}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} L_\delta(\hat{v}^{(i)}, z^{(i)}) + \frac{\lambda_{\text{rank}}}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} \max\left(0, -(\hat{v}^{(i)} - \hat{v}^{(j)}) + \alpha(z^{(i)} - z^{(j)})\right),$$

where $\hat{v}^{(i)}$ and $z^{(i)}$ are the predicted progress delta and the supervision target for the i-th flattened sample in batch $\mathcal{B}$, respectively. $\mathcal{P} = \{(i, j) \in \mathcal{B} \times \mathcal{B} \mid z^{(i)} > z^{(j)}\}$ is the set of valid training pairs, L_δ denotes the Huber loss, λ_{rank} balances the two loss terms, and α is a scaling factor requiring the network's predicted score difference to open up a corresponding margin. During testing, the Dream Evaluator selects the candidate action with the highest predicted progress delta $\hat{v}^{(k)}$. At deployment, the value model scores all eight candidates in approximately 40 ms; Robometer is used offline to generate training labels.

4.4 Dream the Boundaries: Autonomous Learning from Success, Failure & Boundary Cases

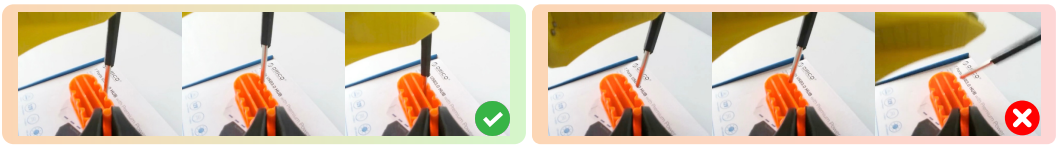


Figure 3: Boundary data in $\mathcal{D}_{online}$.

DreamAvoid introduces failure and boundary data to generate discriminative and informative "dreams." Let $\mathcal{D}^+$ and $\mathcal{D}^-$ denote the sets of successful and failed trajectories, respectively. The base policy is trained exclusively on $\mathcal{D}^+$ and remains frozen during boundary learning and deployment. Its intuitive actions while "awake" are therefore based on successful demonstrations. Conversely,

the Dream Evaluator, the engine for test-time dreaming, adapts its world and value models on $\mathcal{D}^+ \cup \mathcal{D}^-$. The motivation behind this design is that, if trained only on successful data, the world model can generate "optimistic dreams" (consistent with the optimistic errors in §3 and related work (Guo et al., 2026; Yin et al., 2026)). It can learn what success looks like, but struggles to clearly identify which locally feasible actions will actually lead to failure. Especially in fine-grained manipulation tasks, visually subtle deviations can result in drastically different outcomes. By incorporating failure trajectories, the world and value models receive supervision on the critical boundaries between success and failure. However, human teleoperation datasets $\mathcal{D}_{teleop}$ primarily consist of successful demonstrations, making it difficult to provide informative failure cases. To strengthen failure awareness in the world and value models, we propose an *autonomous boundary learning* paradigm. We deploy the initial version of DREAMAVOID (trained only on $\mathcal{D}_{teleop}$), obtaining a mixed dataset $\mathcal{D}_{online}$ that contains successful, boundary, and failed trajectories. At critical phases, the Action Proposer actively explores the action manifold and generates diverse candidate actions. Due to the optimistic hallucinations of the initial Dream Evaluator, it may misjudge locally plausible but globally dangerous actions as safe, thereby naturally exposing the system to trickier boundary cases. These boundary data can either result in narrowly completing the task or lead to failure by a single misstep (Figure 3). The collected empirical data is assigned value labels z_t (defined in §4.3). Subsequently, we combine $\mathcal{D}_{online}$ with $\mathcal{D}_{teleop}$ to fine-tune the Dream Evaluator, adapting its world model and training its value model.

5 REAL-WORLD EXPERIMENTS

We conduct real-world experiments on the AgileX PiperX robotic platform, which is equipped with one front camera (RealSense D455) and two wrist cameras (RealSense D435), as shown in Figure 4. The robot receives RGB observations and language instructions, and executes the action chunks output by the policy. We design four real-world manipulation tasks for evaluation that require accurate spatial alignment, stable contact establishment, and failure avoidance, including **Cup Sleeving**, **Charger Plugging**, **Cap Opening**, and **Screw Insertion**. Compared to typical scenarios, these tasks are much more sensitive to small action deviations. The specific task instructions and sample observations from the three viewpoints can be found in Table 7 and Appendix D.2. Each method is evaluated over 40 independent trials per task (160 in total).

Evaluated Methods. To conduct a comprehensive evaluation, we compare DREAMAVOID with baseline methods and introduce different variants of DREAMAVOID. We compare DREAMAVOID with the following baselines: (1) Base VLA policy: directly executes the action chunk output by $\pi_{0.5}$. (2) GPC-RANK (Qi et al., 2026): this method repeatedly samples multiple candidate action chunks during testing and utilizes a world model for online ranking. We use GPC-RANK as the primary baseline to evaluate the combined design of DREAMAVOID: First, regarding the trigger timing, GPC-RANK is an always-on intervention, whereas DREAMAVOID implements critical-phase intervention. Second, in terms of consequence prediction, our Dream Evaluator combines physical priors from large-scale pre-training with adaptation on real failure and boundary data. Finally, DREAMAVOID uses a learned progress value to rank candidate actions. For consistency, both are evaluated based on the same base policy $\pi_{0.5}$. To examine the contributions of our training paradigm (see §4.3 and §4.4), we define two variants of our method based on the training data distribution: (1) DREAMAVOID-Vanilla (hereafter **DA-Vanilla**): Both the base VLA policy and the Dream Evaluator are trained exclusively on the human teleoperation dataset $\mathcal{D}_{teleop}$. (2) DREAMAVOID-ABL (hereafter **DA-ABL**): This represents the full DREAMAVOID framework incorporating Autonomous Boundary Learning. The base policy remains trained solely on $\mathcal{D}_{teleop}$, while the Dream Evaluator is fine-tuned on the mixed dataset $\mathcal{D}_{teleop} \cup \mathcal{D}_{online}$. The same base policy remains frozen in both variants.

Results. Figure 4 summarizes the real-world results. DREAMAVOID consistently outperforms the base policy and the GPC-RANK method across all tasks. DA-ABL achieves the highest average success rate of 72.5%, surpassing the base policy ($\pi_{0.5}$, 48.8%) by 23.8 percentage points. Additionally, it surpasses the baseline method GPC-RANK (54.4%) in average success rate by 18.1 percentage points. This demonstrates that selective, failure-aware test-time dreaming with rich physical priors is beneficial. Success counts, Wilson 95% confidence intervals, and task-stratified CMH tests are reported in Appendix F (Table 11). Different variants of DREAMAVOID also allow us to examine the importance of our training paradigm. DA-Vanilla already yields an average improvement of

18.1 percentage points over the base policy. However, incorporating autonomous boundary learning (DA-ABL) brings an additional 5.6-point performance improvement. DA-Vanilla is trained exclusively on successful demonstration trajectories, which could limit its ability to intervene effectively during critical phases. In contrast, DA-ABL aims to enhance the Dream Evaluator's failure awareness by incorporating boundary and failure trajectories. DREAMAVOID can reduce failures caused by

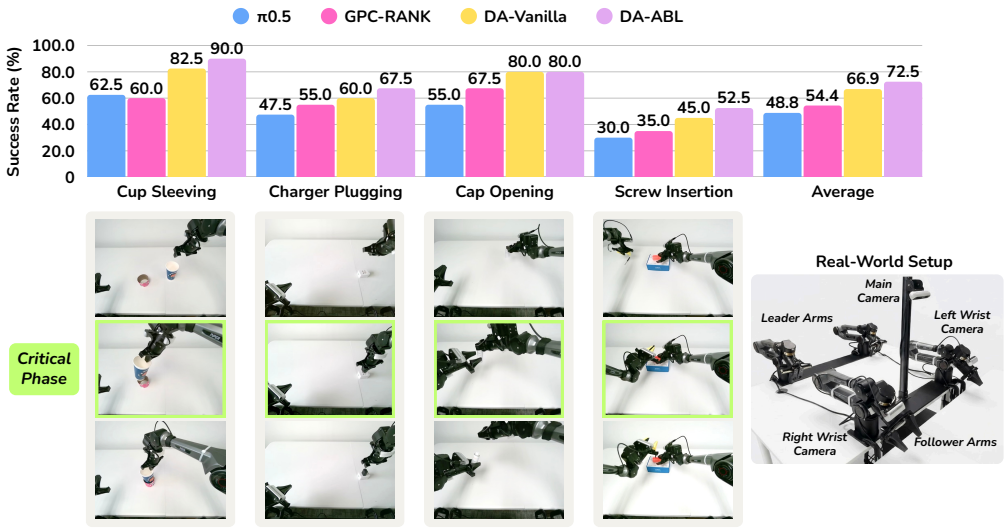


Figure 4: Comparisons in the real world. Each real-world task is evaluated over 40 independent trials.

lateral drift, premature collisions, and contact misalignment. In several experimental trajectories, the base policy produces reasonable actions at most time steps, but makes locally plausible yet globally detrimental decisions as it approaches the critical phase. By sampling multiple candidate action chunks and explicitly evaluating their short-horizon downstream consequences, DREAMAVOID is able to rank alternatives above these failure-prone proposals and instead execute actions that are more consistently aligned with task progression.

Deployment Cost. A key design of DREAMAVOID is to intervene only during the critical phase. This requires just 1–3 interventions per real-world trajectory. For the observed 300–480-frame rollouts, the estimated cumulative inference time is 2.7–7.3 s for DREAMAVOID, compared with 8.3–13.9 s for GPC-RANK under their respective deployment schedules. Detailed timing breakdowns and call-count estimation procedures are provided in Appendix D.3.

Ablation Study. We conduct ablation studies to analyze the contribution of the modules in DREAMAVOID to the performance improvement. All ablation experiments are conducted on real-world manipulation tasks. **(1) Action Proposer vs. Repeated ODE.** We analyze the impact of the candidate action generation method on system performance. To ensure a fair comparison, we construct a strong baseline (Repeated ODE + DA-ABL Dream Evaluator): it retains the Dream Evaluator of DA-ABL, but replaces the candidate action generation mechanism by inputting different random initial noises into the base policy and running deterministic flow inference multiple times to generate the same number of candidate actions. Table 2 shows that even when equipped with the same Dream Evaluator, the average success rate of Repeated ODE is only 57.5%, which is below the full method DA-ABL using SDE sampling (72.5%). This aligns with the principle of "garbage in, garbage out": if the Action Proposer cannot provide high-quality candidate solutions, leading to a severely homogenized candidate pool, then even if the Dream Evaluator can identify impending disasters, it cannot recover an action absent from the pool. This suggests that, compared to deterministic trajectories that only alter the noise at the initial timestep (ODE), the continuous random injection of SDE during the denoising process can explore the action manifold more effectively, potentially exposing richer and higher-quality feasible solutions. Such sample diversity can be important for avoiding subtle failure boundaries during critical phases. **(2) Dream Evaluator vs. Random Select.** We evaluate the contribution of world-model-based consequence evaluation by replacing the Dream Evaluator with random selection within the SDE candidate pool. Table 2 shows that the average success rate of the

random select variant is only 45.0%, which is not only much lower than DA-ABL (72.5%), but even lower than the base policy (48.8%). This demonstrates that the candidate diversity created by the Action Proposer alone is not sufficient to avoid failures. In critical phases with extremely low error tolerance, blind diversity can instead introduce harmful jitters. Because disastrous physical conflicts may be hidden within locally reasonable candidate actions, once the system randomly selects these failure-prone actions, it can lead to irreversible collisions. Therefore, the system benefits from an evaluation mechanism capable of explicitly foreseeing the physical consequences of actions to pick out more promising action paths from diverse hypotheses.

Table 2: Ablation study results.

Method	Components		Task Success Rate (%)				Average
	Action Proposer	Dream Evaluator	Cup	Charger	Cap	Screw	
$\pi_{0.5}$	×(Single ODE)	×	62.5	47.5	55.0	30.0	48.8
DREAMAVOID Variants	×(Repeated ODE)	✓	67.5	55.0	70.0	37.5	57.5
	✓(SDE)	×(Random Select)	65.0	35.0	55.0	25.0	45.0
DA-ABL	✓	✓	**90.0**	**67.5**	**80.0**	**52.5**	**72.5**

6 ROBOTWIN AND SIMPLERENV BENCHMARKS

We evaluate $\pi_{0.5}$ on RoboTwin 2.0 (Chen et al., 2026b), and GR00T N1.6 on SimplerEnv (Li et al., 2025; Bjorck et al., 2025). Following the real-world procedure, we use Gemini 3.6 Flash solely offline to annotate critical-phase onsets using training videos and task instructions. These annotations train a lightweight Dream Trigger that schedules interventions using only causal visual observations and benchmark-provided proprioception at test time. Trigger and SDE sampling parameters are selected on a separate validation split and fixed before testing. On RoboTwin 2.0, across eight bimanual tasks with 100 episodes each, DA-ABL increases success rate from 66.3% to 70.8%. On SimplerEnv, success increases from 59.9% to 63.6% on Bridge and from 76.4% to 80.7% on Fractal. These comparisons are conducted at the episode level and stratified by task. Full success rates and intervals are reported in Appendix E.

7 LIMITATIONS AND FUTURE WORK

First, adapting the failure-aware framework DREAMAVOID to other policies (world action models or traditional control policies) is a promising direction. Second, although DREAMAVOID strictly limits dense computations to sparse critical phases, generating high-fidelity future videos and scoring multiple candidate actions still introduces latency. Future work could not only employ various video generation acceleration techniques, but also opt to execute forward outcome simulations within a compact latent space. Third, beyond optimizing decision-making at test-time, future work could leverage the high-fidelity videos generated by the world model as a rich source of training data, utilizing methods like advantage-weighted regression to continuously update the base policy itself. See Appendix G for further discussion.

8 CONCLUSION

This paper explores failure avoidance in VLA models. Explicit spatiotemporal simulation can provide useful foresight for failure avoidance. We propose the DREAMAVOID framework. Through critical-phase test-time dreaming, DREAMAVOID decouples intuitive execution from foresight by simulating the short-horizon futures of candidate actions during critical phases to select the highest-scoring candidate. To alleviate the challenge of state evaluation in complex scenarios, we introduce an autonomous boundary learning paradigm, enabling the model to better grasp the boundaries between success and failure. Experiments demonstrate that DREAMAVOID can help correct local errors before they escalate. In summary, we emphasize that test-time dreaming matters; integrating intuitive execution with selective dreaming at test time is an effective pathway to enhancing the reliability of VLAs. Through critical-phase dreaming, we can achieve more robust error avoidance capabilities while concentrating additional inference on detected critical phases.

AI USE STATEMENT

Gemini 3.6 Flash and Robometer are components of the research pipeline, as described in Sections 3 and 4.3. Generative AI tools were also used for language polishing. The authors reviewed all AI-assisted content and take full responsibility for the final text, claims, and empirical results.

ETHICS STATEMENT

The reported improvements concern empirical task success and do not constitute certified collision avoidance or safe operation under distribution shift. Dreaming pauses and execution of the best available low-value candidate can still create physical risks. Real-world use therefore requires appropriate supervision and the robot's independent safety mechanisms.

REPRODUCIBILITY STATEMENT

Appendices B–E describe the observation/action setup, trigger training and sensitivity, candidate-sampling settings, evaluator targets, data sources, and evaluation counts. Appendix F specifies the confidence intervals and statistical tests. The anonymous code link is given in the abstract.

REFERENCES

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.

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

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael 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 *Proceedings of Robotics: Science and Systems (RSS)*, 2025b.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Xiang Li, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, Tiejun Huang, Yu Wang, and Chao Yu. π_{RL}: Online rl fine-tuning for flow-based vision-language-action models, 2026a. URL https://arxiv.org/abs/2510.25889.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=itonej9GIV.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

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

Yunhai Feng, Jiaming Han, Zhuoran Yang, Xiangyu Yue, Sergey Levine, and Jianlan Luo. Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2038–2062. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/feng25b.html.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, et al. Dreamdojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.

Google DeepMind. Gemini 3.6 flash. https://ai.google.dev/gemini-api/docs/models/gemini-3.6-flash, 2026.

Wenkai Guo, Guanxing Lu, Haoyuan Deng, Zhenyu Wu, Yansong Tang, and Ziwei Wang. Vla-reasoner: Empowering vision-language-action models with reasoning via online monte carlo tree search. *arXiv preprint arXiv:2509.22643*, 2025.

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. *arXiv preprint arXiv:2602.12063*, 2026.

Songqiao Hu, Zeyi Liu, Shuang Liu, Jun Cen, Zihan Meng, Shihefeng Wang, Xiang Li, and Xiao He. Vlsa: Vision-language-action models with plug-and-play safety constraint layer. *arXiv preprint arXiv:2512.11891*, 2025.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *arXiv preprint arXiv:2506.08009*, 2025.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pp. 492–518. Springer, 1992.

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*, pp. 2679–2713. PMLR, 2025.

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 Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3200–3217. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/kwok25a.html.

Huanyu Li, Kun Lei, Sheng Zang, Kaizhe Hu, Yongyuan Liang, Bo An, Xiaoli Li, and Huazhe Xu. Failure-aware rl: Reliable offline-to-online reinforcement learning with self-recovery for real-world manipulation. *arXiv preprint arXiv:2601.07821*, 2026.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning*, pp. 3705–3728. PMLR, 2025.

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

Changyu Liu, Yiyang Liu, Taowen Wang, Qiao Zhuang, James Chenhao Liang, Wenhao Yang, Renjing Xu, Qifan Wang, Dongfang Liu, and Cheng Han. On-the-fly vla adaptation via test-time reinforcement learning. *arXiv preprint arXiv:2601.06748*, 2026.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di ZHANG, and Wanli Ouyang. Flow-grpo: Training flow matching models via online rl. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3468–3484. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/liu23g.html.

Yujiro Onishi, Ryo Takizawa, Yoshiyuki Ohmura, and Yasuo Kuniyoshi. Exploration-assisted bottleneck transition toward robust and data-efficient deformable object manipulation. *arXiv preprint arXiv:2603.13756*, 2026.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Han Qi, Haocheng Yin, Aris Zhu, Yilun Du, and Heng Yang. Inference-time enhancement of generative robot policies via predictive world modeling. *IEEE Robotics and Automation Letters*, 2026.

Yalcin Tur, Jalal Naghiyev, Haoquan Fang, Wei-Chuan Tsai, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Recurrent-depth vla: Implicit test-time compute scaling of vision-language-action models via latent iterative reasoning. *arXiv preprint arXiv:2602.07845*, 2026.

Kun Wu, Ning Liu, Zhen Zhao, Di Qiu, Jinming Li, Zhengping Che, Zhiyuan Xu, and Jian Tang. Learning from imperfect demonstrations with self-supervision for robotic manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16899–16906. IEEE, 2025.

Charles Xu, Jost Tobias Springenberg, Michael Equi, Ali Amin, Adnan Esmail, Sergey Levine, and Liyiming Ke. Rl token: Bootstrapping online rl with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026.

Yanting Yang, Shenyuan Gao, Qingwen Bu, Li Chen, and Dimitris N Metaxas. Seeing farther and smarter: Value-guided multi-path reflection for vlm policy optimization. *arXiv preprint arXiv:2602.19372*, 2026.

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.

Tenny Yin, Zhiting Mei, Zhonghe Zheng, Miyu Yamane, David Wang, Jade Sceats, Samuel M Bateman, Lihan Zha, Apurva Badithela, Ola Shorinwa, et al. Playworld: Learning robot world models from autonomous play. *arXiv preprint arXiv:2603.09030*, 2026.

Heng Zhang, Rui Dai, Gokhan Solak, Pokuang Zhou, Nikos Tsagarakis, Yu She, and Arash Ajoudani. Safe learning for contact-rich robot tasks: A survey from classical learning-based methods to safe foundation models, 2026. URL https://arxiv.org/abs/2512.11908.

A Related Work

Brittle VLA Fine-Grained Manipulation. VLA models combine large-scale vision-language pretraining with robotic action generation, substantially improving robots' semantic understanding and cross-task generalization abilities (Kim et al., 2025; Black et al., 2025a;b). However, recent studies suggest that the bottleneck of VLA models often lies not in insufficient global semantic understanding, but in local decision errors at a few highly sensitive moments (Xu et al., 2026). As a result, VLA models remain brittle in tasks that require fine-grained alignment, contact control, and obstacle avoidance (Zhang et al., 2026; Hu et al., 2025). Building on this observation, this paper treats failures in fine-grained manipulation as a temporally localized critical phase problem.

Test-Time Compute. To enhance deployment robustness, existing works have begun introducing additional computation into the testing phase. The differences between our method and existing test-time compute mainly lie in two aspects. (1) In terms of trigger timing, existing works tend to favor always-on computation (Liu et al., 2026; Tur et al., 2026; Qi et al., 2026; Bai et al., 2025; Feng et al., 2025). TT-VLA (Liu et al., 2026) and EVOLVE-VLA (Bai et al., 2025) utilize dense progress rewards for constant online policy adaptation during inference. RD-VLA achieves adaptive computation depth through latent recurrent inference (Tur et al., 2026). GPC invokes its world model to evaluate candidate actions at every replanning step (Qi et al., 2026). In contrast, our method DreamAvoid directly executes the base VLA policy most of the time, triggering additional computation only when the Dream Trigger predicts entry into a temporally sparse yet consequence-sensitive critical phase. (2) In terms of evaluation methods, most Verifier or MPC works predominantly rely on current-state scoring or pixel-level goal alignment (Finn & Levine, 2017; Kwok et al., 2025). Visual MPC compares candidate futures based on action-conditioned video prediction, but its scoring criterion is the probability of designated pixels reaching predefined target positions (Finn & Levine, 2017). RoboMonkey investigates VLA test-time scaling by Gaussian perturbation sampling and selecting from candidate actions using a VLM-based verifier (Kwok et al., 2025). GPC combines Diffusion Policy (Chi et al., 2025) with a world model to evaluate candidate actions during deployment, but it does not leverage the rich physical priors embedded in large-scale pre-trained world models (Qi et al., 2026). In comparison, we utilize the high-fidelity future simulation capability of large-scale pre-trained world models. DreamAvoid couples critical-phase-triggered, failure-aware short-horizon dreaming with autonomous boundary learning. By collecting success, failure, and boundary data through deployment, we refine the world model's predictions near success–failure transitions to better support preventive action selection.

Failure Detection. Handling failures is typically viewed as a reactive process: the system must first detect the occurrence of an error and subsequently attempt to correct it. For instance, AHA (Duan et al., 2024) introduces an instruction-fine-tuned VLM specifically designed to detect and reason about manipulation failures using natural language. REFLECT (Liu et al., 2023) explains failures and proposes corrective suggestions by summarizing the robot's experiences. While these post-hoc reasoning capabilities are invaluable for offline policy improvement, reward generation, and sub-task verification, they inherently operate only after a failure has already occurred.

To achieve a safer execution process, our work shifts the reasoning focus of the system from reactive failure detection to proactive avoidance. Instead of waiting for a failure to happen before reasoning, DreamAvoid utilizes test-time dreaming to explicitly forward-simulate the consequences of candidate actions. By invoking this failure-aware evaluation at critical phases, DreamAvoid is able to intercept minor action deviations before they accumulate into irreversible failures. Consequently, our approach acts as a preventive safety layer, effectively complementing the aforementioned works.

World Models. World models provide a natural mechanism for evaluating the downstream consequences of candidate actions. DreamDojo shows that action-conditioned world models, pretrained on large-scale video data and then post-trained on robot data, can support physically meaningful future prediction (Gao et al., 2026). Prior work such as VLAW (Guo et al., 2026) and FARL (Li et al., 2026) show that failed trajectories are important for improving robot robustness, recovery ability, and feasibility judgment.

B Pilot Study Details

To evaluate the model's reasoning capabilities during the critical phase of fine-grained manipulation, we establish a standardized evaluation setup.

Common Setup. Gemini 3.6 Flash identified a critical interaction frame t_{crit} in each test trajectory offline from the recorded video and task instruction. At each t_{crit}, we extract the current frame o_t along with a historical observation sequence consisting of 5 frames with a sampling stride of 8. Each frame is constructed by concatenating three camera views (the main view and the left and right wrist views) into a 2×2 grid (with the bottom-right corner filled in black).

At this moment, the base policy $\pi_{0.5}$ predicts an action chunk of length $H = 50$ (approximately 1.67 s in a 30 fps video). The action consists of 6-dimensional absolute joint angles and the gripper state for both the left and right arms, totaling 14 dimensions. To align with the inference frequency of the world model and reduce the text comprehension burden on the VLM, we downsample the future action chunk using a stride of 4, extracting 13 actions with indices $[0, 4, 8, \ldots, 48]$. This sparse action sequence u_t spans 48 time steps, corresponding to an execution window of approximately 1.6 s into the future.

Setup for Explicit Spatiotemporal Dreaming. The historical observation stride is 8, and the future action sequence stride is 4. A sparser visual history is sufficient to capture the macroscopic motion context. However, accurately predicting the physical evolution of the critical phase, which is sensitive to minute deviations, requires denser action-conditioned inputs to ensure that the world model can perform precise spatiotemporal rendering. The world model DreamDojo is fine-tuned on 100 human teleoperation trajectories. Figure 5 illustrates the actual execution alongside the future frames predicted by the world model.

C DreamAvoid Framework Details

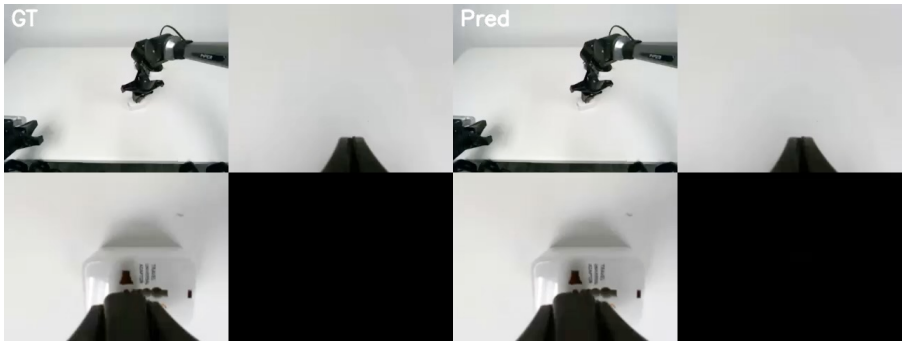
The complete execution procedure of DreamAvoid is summarized in Algorithm 1.

Algorithm 1 DreamAvoid

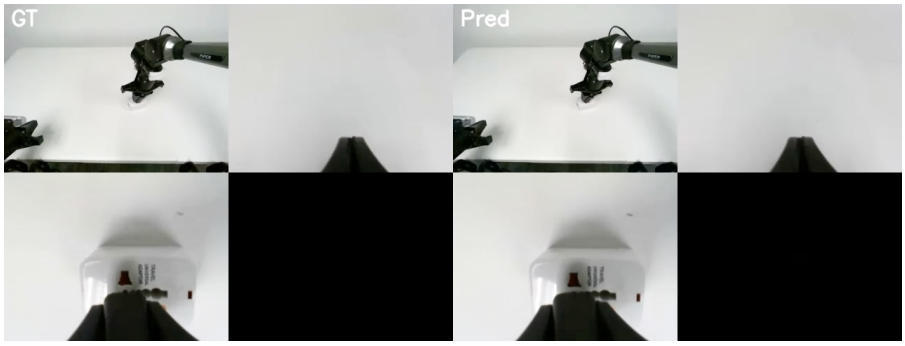
Given instruction l, initialize $d^{(m)} = 0$ for all $m = 1, \ldots, M$ at rollout reset.
while the rollout is active **do**
  At the next policy decision, after chunk execution completes and with the evaluator idle, observe o_t and s_t.
  Read the latest scores $\boldsymbol{p}_t = [p_t^{(1)}, \ldots, p_t^{(M)}]$ from the asynchronously polled Dream Trigger.
  Form $\mathcal{A}_t = \{m \in \{1, \ldots, M\} : d^{(m)} = 0,\ p_t^{(m)} \geq \gamma\}$.
  if $\mathcal{A}_t \neq \varnothing$ **then**
    Set $d^{(m)} = 1$ for all $m \in \mathcal{A}_t$.
    Sample $\{u_t^{(k)}\}_{k=1}^{N_c}$ via the stochastic flow solver (Action Proposer).
    for $k = 1$ to N_c **do**
      Predict $\hat{y}_t^{(k)} = \mathcal{W}_\psi(o_t, u_t^{(k)})$.
      Score $\hat{v}_t^{(k)} = V_\omega(o_t, u_t^{(k)}, \hat{y}_t^{(k)})$.
    end for
    Select $k^* = \arg\max_k \hat{v}_t^{(k)}$.
    Execute $u_t^{(k^*)}$.
  else
    Execute the default chunk $u_t \sim \pi_\theta(\cdot \mid o_t, l)$.
  end if
end while

C.1 Dream Trigger

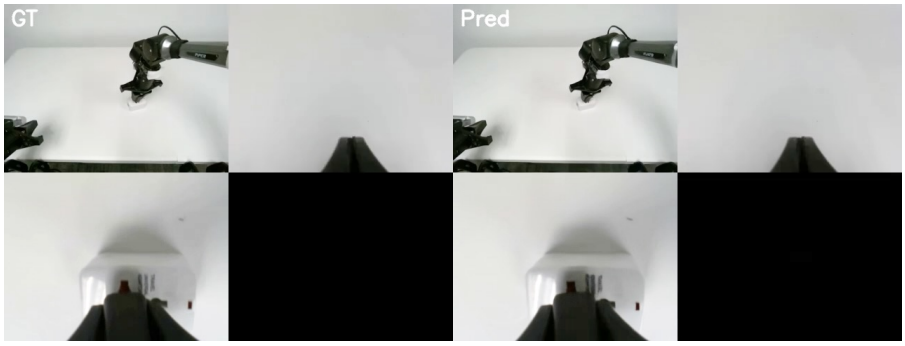
Critical-phase onsets for trigger training are annotated offline by Gemini 3.6 Flash from task videos and instructions. Approximately 26,000–34,000 frame labels were obtained from 100 training videos



(a)



(b)



(c)

Figure 5: (a) shows the current frame t_{crit}, while (b) and (c) show the actual future outcome frames (GT) and the video frames predicted by DreamDojo (Pred). Each 2x2 image block contains different camera views: the top-left is the main view, the top-right is the left wrist view, and the bottom-left is the right wrist view. This concatenation method follows the approach of DreamZero (Ye et al., 2026) to accommodate the single-image input requirement of DreamDojo.

per task. Gemini 3.6 Flash is used only for offline annotation; the learned trigger uses causal visual observations and proprioception at test time.

Considering the requirements for real-time inference speed, we select the frozen DINOv2-ViT-B/14 (Oquab et al., 2023) as the backbone for Dream Trigger. First, we perform frame-by-frame feature encoding on the images from each camera feed (which include sparsely sampled historical observations and the current frame) and execute average pooling along the temporal dimension. Subsequently, the aggregated single- or multi-camera visual features are concatenated with the current proprioceptive state s_t and jointly fed into a three-layer MLP classification head to determine the current phase state. In real-world deployment, we configure Dream Trigger to poll asynchronously at a frequency of 1 Hz, which is shorter than the 1.67 s execution cycle of the action chunk.

On a single RTX 4090, trigger inference takes approximately 50 ms per poll. It runs in a separate process alongside policy inference, with reported average GPU utilization below 5% and no observable additional control-loop delay in this setup.

For a task with $M > 1$ designated interaction stages (e.g., grasping $\rightarrow$ alignment), the trigger outputs $\boldsymbol{p}_t = [p_t^{(1)}, \ldots, p_t^{(M)}]$. Each head corresponds to a task-defined stage with its own annotated onset $t_{\text{crit}}^{(m)}$, contact time $t_{\text{contact}}^{(m)}$, and reference window $\mathcal{I}^{(m)} = [t_{\text{crit}}^{(m)}, t_{\text{contact}}^{(m)})$. Its training target is $\tilde{y}_t^{(m)} = \text{Sigmoid}((t - t_{\text{crit}}^{(m)})/\beta)$. The heads share the visual backbone and MLP, with an M-dimensional final layer, and are trained by summing their binary cross-entropy losses:

$$\mathcal{L}_{\text{trigger}}^{\text{multi}} = \sum_{m=1}^{M} \mathcal{L}_{\text{trigger}}^{(m)}. \tag{1}$$

Each stage has a persistent flag $d_t^{(m)}$, initialized to zero at rollout reset. At an available policy decision, the eligible stages are

$$\mathcal{A}_t = \left\{ m : d_t^{(m)} = 0,\ p_t^{(m)} \geq \gamma \right\}, \qquad c_t^{\text{test}} = \mathbb{I}[\mathcal{A}_t \neq \varnothing]\,. \tag{2}$$

If $\mathcal{A}_t$ is nonempty, the scheduler launches one evaluation cycle and sets $d_t^{(m)} = 1$ for all $m \in \mathcal{A}_t$. Simultaneously eligible stages share this invocation. Other stage flags remain unchanged. Thus, a previously triggered head cannot generate repeated calls even if its score remains high, while an untriggered stage can initiate a later intervention. All flags are cleared only when the rollout resets. This rule permits at most one invocation per designated stage and at most M invocations per rollout.

C.1.1 Held-Out Detection and Sensitivity

We evaluate the trigger on 80 held-out videos across the four real-world tasks, with critical-phase reference annotations generated offline by Gemini 3.6 Flash. Consecutive positive polls are merged into one intervention event. An event is correct if it occurs within the model-annotated pre-contact intervention window; recall counts whether the window contains at least one detection. Event precision uses detected events as its denominator, whereas recall uses the 80 annotated phases.

Table 3: Trigger threshold sensitivity on 80 held-out videos. Intervention counts merge consecutive positive polls.

γ	Event precision	Phase recall	Events / trajectory
0.3	84.9% (79/93)	98.8% (79/80)	1.16
0.5	92.7% (76/82)	95.0% (76/80)	1.03
0.7	97.2% (70/72)	87.5% (70/80)	0.90

Table 4: Soft-label sensitivity after retraining the Charger Plugging trigger on the same training set.

β (frames)	Event precision	Phase recall	Events / trajectory
2	94.7%	90.0%	0.95
5	95.0%	95.0%	1.00
10	87.0%	100.0%	1.15

A lower γ increases coverage and intervention count, while a higher value misses more phases. At the default $\gamma = 0.5$, event precision is 92.7% (76/82), phase recall is 95.0% (76/80), and the missed-phase rate is 5.0% (4/80). A narrower β loses recall on Charger Plugging; a broader transition increases activations.

C.2 Action Proposer

Table 5 lists the noise level σ, candidate count N_c, and action horizon H. Both DreamAvoid variants share these settings. GPC-RANK uses its own sampling mechanism, with the same N_c and H for each environment. Varying σ changes diversity and fidelity to the frozen policy; the Repeated-ODE comparison tests removal of integration noise.

Table 5: Hyperparameters for Action Proposer in real-world and simulation environments, detailing the SDE noise level σ, action candidate counts N_c, and action horizon H. Both DA-Vanilla and DA-ABL share the same parameters. GPC-RANK utilizes its own generative sampling mechanism rather than SDE sampling, hence the SDE noise level σ is not applicable (N/A).

Environment	Method	Task	σ	N_c	H
Real-World	DREAMAVOID	Cup Sleeving	0.1	8	50
		Charger Plugging	0.05		
		Cap Opening	0.08		
		Screw Insertion	0.05		
	GPC-RANK	All Tasks	N/A	8	50
Simulation	DREAMAVOID	SimplerEnv	0.3	4	16
		RoboTwin 2.0	0.3		50
	GPC-RANK	SimplerEnv	N/A	4	16

C.3 DREAM EVALUATOR

C.3.1 WORLD MODEL

The world model is initialized with DreamDojo-2B (Gao et al., 2026) and post-trained on target robot data. To support low-latency online intervention, we follow the Self Forcing paradigm (Huang et al., 2025) to further distill the bidirectional teacher world model into an autoregressive student model. This distillation process consists of two stages: (1) Warmup: under a teacher forcing mechanism, the student model's predictions are regressed onto the ODE trajectories generated by the teacher. (2) Distillation: utilizing a distribution matching method, the student model is trained autoregressively on its own generated context. This student model replaces the bidirectional attention mechanism with a causal attention mechanism and reduces the number of diffusion sampling steps, thereby achieving efficient online rollout generation while maintaining temporal consistency and action controllability. A single 13-frame video inference takes approximately 2 s on a single RTX 4090 GPU.

C.3.2 VALUE MODEL

We regress the continuous progress delta within the execution horizon H:

$$z_t^{(k)} = \eta_{t+H} - \eta_t \tag{3}$$

where $\eta_t \in [0, 1]$ is the frame-level progress score provided by Robometer (Liang et al., 2026). This continuous label intuitively reflects the true physical contribution of the action $u_t^{(k)}$ towards task completion. To ensure the network can unambiguously identify catastrophic consequences, we introduce a boundary penalty mechanism. If Robometer predicts an irreversible failure within the horizon (e.g., a dropping object or severe collision causing a precipitous drop in success probability μ_t), we forcefully truncate and reset its target label $z_t^{(k)}$ to -1. The final layer of the value model directly outputs the predicted progress delta $\hat{v}_t^{(k)}$ via a linear layer. Since the target label $z_t^{(k)}$ is strictly bounded within the $[-1, 1]$ range, the network implicitly learns to constrain its unnormalized outputs to a similar range during training.

C.3.3 FAILURE AUDIT

We use synchronized video and proprioceptive logs to review 44 failed trials of DA-ABL. Two trials, one Charger Plugging and one Screw Insertion, have no timely intervention because the trigger misses the model-annotated pre-contact window or activates too late. They are excluded from the denominator for evaluating a scored failure-causing action. For each of the remaining 42 trials, we attribute failure to the most recent evaluated action chunk whose executed interval directly leads to the first irreversible state transition. Six of these chunks score above the threshold, yielding a false-negative rate of 6/42 = 14.3% and hazard-detection recall of 36/42 = 85.7% (Wilson 95% CI [72.2, 93.3]).

Table 6: The six optimistic errors in the value-model audit. All were executed and subsequently caused failure.

Task	Count	Observed failure mode
Cup Sleeving	1	Contact with the sleeve rim
Charger Plugging	2	Socket-edge collision or lateral misalignment
Cap Opening	1	Gripper slippage
Screw Insertion	2	Subtle tip-to-hole misalignment

Most missed hazards involve fine contact geometry or occlusion insufficiently represented in the predicted video; two become apparent only near or beyond the prediction horizon. In the other 36 evaluated failures, the selected action is already below the diagnostic threshold, but is executed because it ranks highest. This exposes a limitation of relative ranking: a recognized hazardous action can still be the best available candidate. This recall measures recognition among evaluated failures.

D Real-World Evaluation Details

D.1 Real-World Deployment Setup

We deploy our base policy $\pi_{0.5}$ on a single NVIDIA RTX 4090 GPU. Table 7 details the task specifications, training steps (on 8 NVIDIA A100 GPUs), and demonstration data statistics (episodes and average lengths) for the real-world post-training of $\pi_{0.5}$.

Table 7: Detailed task information for the real-world evaluation tasks. All videos in the teleoperation dataset are recorded at 30 FPS.

Task Name	Task Instruction	$\pi_{0.5}$ Training Steps	Teleop Episodes	Avg. Frames
Cup Sleeving	Pick up the paper cup and put it into the cup sleeve.	40000	100	330.8
Charger Plugging	Plug the charger into the socket.	50000	100	336.5
Cap Opening	Open the cap of the spray.	50000	100	263.6
Screw Insertion	Insert the screw into the hole in the box.	50000	100	313.6

D.2 Multi-View Input Observations for Base Policy

Figure 6 shows three-view input observations for base policy $\pi_{0.5}$ in real-world tasks.

D.3 Latency Analysis and Smooth Transition

Table 8 compares latency at the trajectory level under the deployed scheduling policies. DreamAvoid invokes the evaluator only at 1–3 detected critical phases, whereas GPC-RANK replans every 25 frames. Over the observed 300–480-frame rollouts, DreamAvoid requires an estimated 2.69–7.33 s of cumulative inference, compared with 8.33–13.88 s for GPC-RANK.

Call Counts and Cumulative Inference. Observed evaluation rollouts contain approximately 300–480 frames, whereas the 260–340-frame range in the demonstration table describes training videos. At these rollout lengths, the base policy replans $\lceil T/50 \rceil = 6$–10 times and GPC-RANK replans $\lceil T/25 \rceil = 12$–20 times. DreamAvoid triggers 1–3 interventions per trajectory. Using the measured mean per-call costs, a conservative additive accounting is

$$\begin{aligned} L_{\text{base}}(T) &= 0.093\lceil T/50 \rceil, \\ L_{\text{DA}}(T, m) &\approx 0.093\lceil T/50 \rceil + 2.133m, \\ L_{\text{GPC}}(T) &= 0.694\lceil T/25 \rceil, \end{aligned} \tag{4}$$

in seconds, with m the intervention count. This yields 0.56–0.93 s for the base policy, 2.69–7.33 s for DreamAvoid, and 8.33–13.88 s for GPC-RANK across the stated ranges. The additive DreamAvoid estimate can overcount a base-policy call when candidate sampling replaces that call.



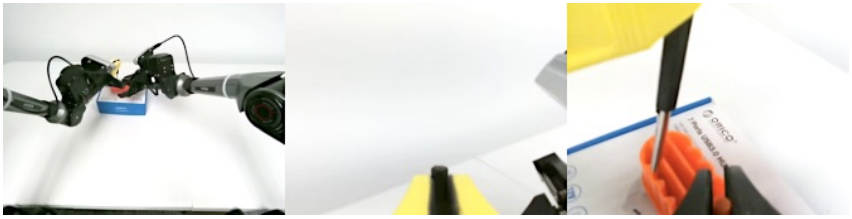
(a) Cup Sleeving



(b) Charger Plugging



(c) Cap Opening



(d) Screw Insertion

Figure 6: Multi-view input observations for base policy $\pi_{0.5}$ in real-world tasks (main view | left wrist view | right wrist view). Each row presents three synchronized camera views for a specific task, highlighting the need for accurate spatial alignment and contact establishment. These tasks are sensitive to minor action deviations compared to coarse-grained pick-and-place scenarios.

Table 8: Measured per-call costs and estimated cumulative inference latency under the observed rollout schedules ($T = 300$–480 frames).

Method	Deployment schedule	Calls / trajectory	Cumulative latency (s)
Base $\pi_{0.5}$	Replan every 50 frames (0.093 s/call)	6–10	0.56–0.93
DREAMAVOID (ours)	Base replanning every 50 frames; intervention only at detected critical phases (2.133 s/intervention)	6–10 base + 1–3 interventions	**2.69–7.33**
GPC-RANK	Always-on replanning every 25 frames (0.694 s/replan)	12–20	8.33–13.88

Per-call breakdown. A DREAMAVOID intervention comprises action proposal (0.093 s), video generation (2.000 s), and value scoring (0.040 s). A GPC-RANK replan comprises candidate sampling (0.093 s), world rollout (0.600 s), and reward prediction (0.001 s). The asynchronous DREAMAVOID trigger (0.050 s/poll) runs in a separate process and is excluded from the critical path.

The scheduling-dependent evaluator overhead is mC_{eval} rather than $\lceil T/r \rceil C_{\text{eval}}$ for an always-on schedule with replan interval r.

Holding and Resuming Execution. During dreaming, high-level ROS command publication pauses while the low-level servos track the last position setpoint. After inference, the action sequence is reinitialized using the latest proprioceptive state; linear trajectory blending maintains C^0 command continuity. Resetting the high-level timer and adding a stabilization buffer prevents accumulated timer deficits from releasing commands in a burst. These implementation measures reduce command discontinuities.

E Simulation Benchmark Details

Simulation follows the real-world annotation and training procedure: Gemini 3.6 Flash annotates critical-phase onsets offline from training videos and task instructions. These annotations supervise a lightweight Dream Trigger that schedules interventions using only causal visual observations and benchmark-provided proprioception at test time. Trigger and SDE sampling parameters are selected on a separate validation split and fixed before testing. GPC-RANK is always-on. Table 9 reports success rates and Wilson 95% intervals. The accounting is 200 per SimplerEnv task (1400 Bridge and 1000 Fractal episodes). The intervals and tests are computed from these aggregate counts. Figure 7 illustrates four predicted futures.

Table 9: SimplerEnv success (%) and Wilson 95% intervals, with 200 episodes per task. Averages pool 1400 Bridge or 1000 Fractal episodes. Task labels are shortened for readability.

Task	GR00T N1.6	GPC-RANK	DA-Vanilla	DA-ABL
Bridge (WidowX robot)				
Spoon on towel	62.5 [55.6, 68.9]	64.5 [57.7, 70.8]	65.0 [58.2, 71.3]	67.0 [60.2, 73.1]
Carrot on plate	63.0 [56.1, 69.4]	64.0 [57.1, 70.3]	64.0 [57.1, 70.3]	65.5 [58.7, 71.7]
Eggplant in basket	89.5 [84.5, 93.0]	91.0 [86.2, 94.2]	92.5 [88.0, 95.4]	95.0 [91.0, 97.3]
Stack cube	4.0 [2.0, 7.7]	6.0 [3.5, 10.2]	5.5 [3.1, 9.6]	7.0 [4.2, 11.4]
Eggplant in sink	38.0 [31.6, 44.9]	38.0 [31.6, 44.9]	40.5 [33.9, 47.4]	41.0 [34.4, 47.9]
Close drawer	68.0 [61.2, 74.1]	70.5 [63.8, 76.4]	70.0 [63.3, 75.9]	73.5 [67.0, 79.1]
Open drawer	94.0 [89.8, 96.5]	94.0 [89.8, 96.5]	95.5 [91.7, 97.6]	96.0 [92.3, 98.0]
Average	59.9 [57.3, 62.4]	61.1 [58.6, 63.7]	61.9 [59.3, 64.4]	63.6 [61.0, 66.1]
Fractal (Google robot)				
Pick coke can	97.0 [93.6, 98.6]	97.5 [94.3, 98.9]	97.0 [93.6, 98.6]	98.5 [95.7, 99.5]
Pick object	85.0 [79.4, 89.3]	87.0 [81.6, 91.0]	88.0 [82.8, 91.8]	89.5 [84.5, 93.0]
Move near	73.5 [67.0, 79.1]	74.0 [67.5, 79.6]	76.0 [69.6, 81.4]	78.5 [72.3, 83.6]
Open drawer	44.0 [37.3, 50.9]	45.0 [38.3, 51.9]	48.5 [41.7, 55.4]	51.0 [44.1, 57.8]
Close drawer	82.5 [76.6, 87.1]	87.5 [82.2, 91.4]	85.0 [79.4, 89.3]	86.0 [80.5, 90.1]
Average	76.4 [73.7, 78.9]	78.2 [75.5, 80.6]	78.9 [76.3, 81.3]	80.7 [78.1, 83.0]

RoboTwin 2.0 uses the Aloha-AgileX bimanual robot. We evaluate eight tasks with 100 episodes per task in the demo_clean setting. All methods re-plan every 25 steps. Table 10 reports success rates and Wilson 95% intervals.

F Statistical Reporting

Table 11 summarizes aggregate real-world results across four tasks, with 40 trials per task per method. Table 12 provides per-task success rates and confidence intervals, including ablations.

Counts and Confidence Intervals. Real-world rates use 40 trials per task per method. Table 12 includes all methods and ablations. For k successes among n trials, $\hat{p} = k/n$ and $z_{.975} = 1.959964$,



Figure 7: On the google_robot_pick_coke_can task of the SimplerEnv benchmark, future videos are generated by DreamDojo based on 4 action chunk candidates produced by GR00T N1.6 via SDE sampling (13 frames in total; frames 0, 3, 6, 9, and 12 are shown here). Among these, (4) has the highest corresponding value score.

Table 10: RoboTwin 2.0 success (%) and Wilson 95% intervals, 100 episodes per task.

Task	$\pi_{0.5}$	DA-ABL
Hanging mug	30.0 [21.9, 39.6]	19.0 [12.5, 27.8]
Handover block	82.0 [73.3, 88.3]	94.0 [87.5, 97.2]
Stack bowls (3)	77.0 [67.8, 84.2]	74.0 [64.6, 81.6]
Stack blocks (3)	96.0 [90.2, 98.4]	98.0 [93.0, 99.4]
Place phone stand	75.0 [65.7, 82.5]	78.0 [68.9, 85.0]
Stamp seal	63.0 [53.2, 71.8]	79.0 [70.0, 85.8]
Place can in basket	74.0 [64.6, 81.6]	73.0 [63.6, 80.7]
Move stapler to pad	33.0 [24.6, 42.7]	51.0 [41.3, 60.6]
Average	66.3 [62.9, 69.4]	70.8 [67.5, 73.8]

Table 11: Real-world success over 160 trials per method. Confidence intervals are Wilson 95%; CMH tests compare each method with DA-ABL while stratifying by task.

Method	Successes	Success (%) [95% CI]	p vs. DA-ABL
Base $\pi_{0.5}$	78/160	48.8 [41.1, 56.4]	7.4×10^{-6}
GPC-RANK	87/160	54.4 [46.6, 61.9]	5.3×10^{-4}
DA-Vanilla	107/160	66.9 [59.3, 73.7]	0.251
DA-ABL	116/160	72.5 [65.1, 78.8]	—

the Wilson interval is

$$\frac{\hat{p} + z_{.975}^2/(2n) \;\pm\; z_{.975}\sqrt{\hat{p}(1-\hat{p})/n + z_{.975}^2/(4n^2)}}{1 + z_{.975}^2/n}. \tag{5}$$

Aggregate intervals describe the pooled episode success rate across the fixed evaluated tasks.

Table 12: Real-world success (%) with Wilson 95% intervals, 40 trials per cell.

Method	Cup	Charger	Cap	Screw
Base $\pi_{0.5}$	62.5 [47.0, 75.8]	47.5 [32.9, 62.5]	55.0 [39.8, 69.3]	30.0 [18.1, 45.4]
GPC-RANK	60.0 [44.6, 73.7]	55.0 [39.8, 69.3]	67.5 [52.0, 79.9]	35.0 [22.1, 50.5]
DA-Vanilla	82.5 [68.1, 91.3]	60.0 [44.6, 73.7]	80.0 [65.2, 89.5]	45.0 [30.7, 60.2]
Repeated ODE	67.5 [52.0, 79.9]	55.0 [39.8, 69.3]	70.0 [54.6, 81.9]	37.5 [24.2, 53.0]
Random Select	65.0 [49.5, 77.9]	35.0 [22.1, 50.5]	55.0 [39.8, 69.3]	25.0 [14.2, 40.2]
DA-ABL	90.0 [76.9, 96.0]	67.5 [52.0, 79.9]	80.0 [65.2, 89.5]	52.5 [37.5, 67.1]

Comparative Tests. We use a two-sided Cochran–Mantel–Haenszel (CMH) test without continuity correction, stratifying by the reported task or evaluation setting. Each stratum is a 2×2 table of method by success/failure. This treats episodes as the statistical units and does not assume matched trial pairs. Real-world DA-ABL versus base and GPC-RANK gives $p = 7.3848 \times 10^{-6}$ and 5.2979×10^{-4}, respectively. DA-ABL versus DA-Vanilla gives $p = 0.2512$; versus Repeated ODE, $p = 0.003616$; and versus Random Select, $p = 1.9048 \times 10^{-7}$. For simulation, task-stratified CMH tests comparing DA-ABL with the base policy yield $p = 0.01225$ on Bridge, 0.01015 on Fractal, and 0.02761 on RoboTwin 2.0.

G Limitations and Future Work

First, although DreamAvoid strictly limits dense computations to sparse critical phases, generating high-fidelity future videos and scoring multiple candidate actions still introduces latency. For high-precision but slower-paced manipulation tasks, this overhead can be acceptable. However, in highly dynamic environments requiring split-second reactive control, this latency may hinder the robot's smooth execution. In addition to employing various video generation acceleration techniques, future research could also perform forward consequence simulation within a compact latent space. It must be emphasized, however, that explicitly generating pixel-level videos holds practical value: its intuitive interpretability not only enables human researchers to quickly diagnose and optimize the model, but these synthesized failure scenarios and corrective trajectories can also be validated for use as negative samples or preference data for the continuous fine-tuning of subsequent VLA or world models (e.g., via DPO or RLHF).

Second, although this paper primarily uses DreamAvoid as an enhancement module for VLA models, its core test-time dreaming mechanism is modular with respect to the underlying base policy. Future work could integrate this failure-aware framework as a test-time intervention layer into broader embodied control architectures, such as the emerging World Action Models or more traditional control policies. For specific implementations, adaptive modifications to the action sampling methods of different architectures are required, together with task-specific trigger training and evaluator adaptation: for instance, SDE sampling can be adopted for action trajectory search in flow matching-based policies, while diverse candidate actions can be generated by injecting Gaussian perturbations in diffusion-based models.

H Broader Impacts

Improved empirical reliability in contact-sensitive manipulation could reduce some failed executions and wasted robot interaction. The present evidence does not establish safety in homes, factories, or out-of-distribution scenes, and the system may still select hazardous actions or pause in unfavorable

contact states. Appropriate human supervision and independent hardware safeguards remain necessary. More reliable automation can also affect workers performing related manual tasks; deployment should consider these social consequences alongside task performance.