

Our key insight is that visual distractors underrepresented in the training data often behave implausibly during world model rollouts—quickly distorting, disappearing, or warping unnaturally over time. To detect such cases, we prompt the VLM with two frames from the world model’s predicted rollout: the initial frame, which contains the full scene context including potential distractors, and the fifth frame, where empirical analysis shows that distractors are most likely to have degraded or disappeared. This selection is based on our observation that implausible distractor behavior typically manifests within the first four steps of rollout prediction. To enable precise spatial reasoning, we generate segmentation masks for the initial frame using Grounded-SAM2 [14] and overlay unique mask IDs directly onto the segmentation map. The VLM is then presented with the ID-labeled segmentation image alongside the initial and fifth frames, and is prompted to identify which regions (i.e., mask IDs) correspond to objects that have disappeared or changed unnaturally over time.

Prompt for novel distractor identification

You are a helpful assistant for a robotic system that analyzes images to detect changes and disappearing objects.

I’ll show you three images:

1. `img_init_dis` - The initial image
2. `img_wm` - The same scene after some changes
3. `mask_overlay` - The initial image with numbered masks/patches created by a segmentation algorithm

Your task is to identify which numbered patches from the `mask_overlay` image exist in `img_init_dis` but have disappeared in `img_wm`. Look carefully at each numbered patch and determine if the corresponding object still present in `img_wm`.

Hint: This is a kitchen environment with a robot arm, try only detect patches that the corresponding object are on the white table.

Please provide **ONLY** a list of the missing patch numbers with no additional explanation. For example, if patches 2, 5, and 7 have disappeared, just respond with: [2, 5, 7].

Once the VLM produces its response—typically in the form of a list of missing or degraded patch IDs (e.g., `Missing patches: [2, 8, 9]`)—we use the identified mask indices to extract the corresponding image patches from the initial observation. These patches represent regions that are likely to be visual distractors causing inconsistencies in the rollout. Figure 7 shows an example of these extracted distractor patches.

A. *DINO-WM* Training

We adopt **DINO-WM** [26] as our base world model. The hyperparameters used for training are listed in Table III. Training is performed on 500 robot–environment interaction trajectories, comprising 200 rollouts from a pre-trained diffusion policy and 300 from random exploration. Each trajectory spans 24 steps with a 0.75-second interval between steps in the real world. We omit frameskip during training, as the existing time gap is sufficient—predicting further into the future would impair model accuracy. All models are trained on a single Nvidia A6000.

TABLE III: Hyperparameters for **DINO-WM** training.

Name	Value
seed	0
epochs	300
batch_size	64
save_every_x_epoch	10
reconstruct_every_x_batch	500
num_reconstruct_samples	6
encoder_lr	1e-6
decoder_lr	2e-5
predictor_lr	5e-4
action_encoder_lr	5e-4
img_size	224
frameskip	1
concat_dim	1
saved_folder	null
normalize_action	True
action_emb_dim	10
num_action_repeat	1
proprio_emb_dim	10
num_proprio_repeat	1
num_hist	3
num_pred	1
has_predictor	True
has_decoder	True

All 500 training trajectories are collected in a controlled kitchen environment, where the robot is tasked with transporting a green pepper from a cutting board—randomly positioned on the left side of the table—to a cooking pan placed on an induction stove with varying orientations. To introduce additional variability, a spoon is also randomly placed on the table surface during data collection. As illustrated in Figure 8, **DINO-WM** is able to accurately predict future states with fine-grained visual detail, maintaining both temporal and spatial consistency across frames. It also demonstrates robustness to environmental randomization within the training domain.

However, when we introduce novel visual distractors—such as an Amazon box, a kettle, and a high-pressure pot—randomly positioned within the scene, the predictive performance of **DINO-WM** significantly deteriorates. In these out-of-distribution scenarios, parts of the predicted images become visibly distorted, disappear altogether, or warp in physically implausible ways across the rollout sequence, as shown in Figure 9.

Figure 10 presents the reimagined rollout results using modified input observations for the same 10 initial states containing visual distractors. While some artifacts remain—primarily due



Fig. 6: The VLM is prompted to identify visual distractors by reasoning about missing objects between the initial frame (middle) and the fifth frame (right) of the world model rollout, using mask IDs from the semantic segmentation (left).

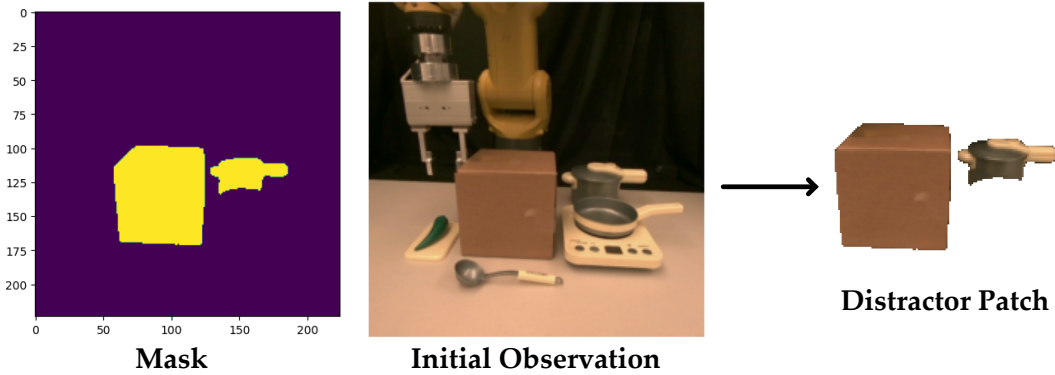


Fig. 7: Distractor patches extracted from the initial observation based on the identified mask.

to limitations in the segmentation and inpainting models—the severe hallucinations observed in the original rollouts have been effectively mitigated. Notably, the differences in row 4 and row 9 between Figure 9 and Figure 10 highlight the improved visual stability and consistency achieved through our intervention.

B. DINOv2 Feature Decoder

To enable accurate reconstruction of visual observations from predicted latent representations in the testing environment, we perform a dedicated fine-tuning of the DINOv2 feature decoder. This process is carried out separately from the original world model training. We first collect a new dataset consisting of 80 trajectories captured within the testing environment, which intentionally includes various visual distractors not present during the initial training phase. These trajectories are used to better adapt the feature decoder to the distributional shift introduced by the novel visual elements.

We then initialize the training pipeline by loading a pre-trained **DINO-WM** checkpoint, which contains the previously trained dynamics model and feature decoder. However, during fine-tuning, we freeze the parameters of the dynamics model entirely to preserve its learned temporal representations and focus solely on updating the parameters of the DINOv2 feature

decoder. This ensures that only the visual decoding pathway is adapted to the new environment, without altering the predictive capabilities of the dynamics model itself.

The fine-tuning procedure is conducted over a single epoch using the newly collected dataset. This one-pass update is sufficient to adjust the decoder’s mapping from latent space to image space, improving reconstruction fidelity in the presence of unfamiliar distractors while maintaining consistency with the world model’s learned representations.

Our approach mitigates the impact of novel environmental variations on downstream planning by applying test-time observation intervention and reimagination. Alternatively, another intervention strategy is to reject untrustworthy world model action outcome predictions at test time to ensure safety. In addition to comparing against the base **DINO-WM**, we compare our approach against **TrustRegion** [10], which finds a region where the learned visual dynamics model is deemed reliable for downstream planning and rejects world model predictions if the input observation and action plan pair falls outside of the trust region.

Lipschitz constant of a function quantifies the maximum rate at which the function’s output can change with respect to changes in its input. Formally, a function f is Lipschitz continuous if there exists a constant $L \geq 0$ such that for all



Fig. 8: Trajectories predicted with **DINO-WM** with 10 different initial observations without visual distractors.



Fig. 9: Trajectories predicted with **DINO-WM** with 10 different initial observations with visual distractors.



Fig. 10: Trajectories reimagined by DINO-WM with 10 modified initial observations.

inputs x_1 and x_2 , the following holds: $\|f(x_1) - f(x_2)\| \leq L \cdot \|x_1 - x_2\|$. Here, L denotes the Lipschitz constant, and it provides an upper bound on the function’s sensitivity.

TrustRegion leverages an estimated Lipschitz constant (bound how much outputs change with respect to a change in the inputs) of the world model visual outcome prediction error to obtain an error bound for a test-time input observation. By defining a maximum acceptable visual dynamics prediction error threshold, we could find the corresponding trust region such that any input observation and action plan pair within this region produces an error bounded by the threshold.

Let T denote a trust region (a union of r -balls about the subset of the world model training data), b_T denote the dispersion of the region, and e_T be the maximum training error of the trained world model. Then for any input (a pair of an initial observation and an action plan) within the T , its world model prediction error is bound by

$$\epsilon = L_T b_T + e_T. \quad (2)$$

Implementation. We estimate the Lipschitz constant of the world model’s visual action plan outcome prediction error. In this work, we use the L2 difference between the world model’s predicted latent state trajectory and the ground-truth latent state (DinoV2 embedding) trajectory given the same initial observation and the action plan as the prediction error. Following the procedure from [10], we begin by selecting training data points whose prediction error is below a threshold of 750 (an empirically chosen value below which the model’s predictions typically highly align well with ground truth). These filtered data points are used to initialize the trust region T_0 , defined as a union of r_0 -balls centered around the selected points. In our experiments, we set r_0 as a unit ball with a radius of 0.1, and the corresponding Lipschitz constant estimated within T_0 is 0.84. To maximize the flexibility of the trust region for planning, we progressively increase r to expand the region, while ensuring that the Lipschitz constant remains within acceptable bounds (< 1). The full procedure for this expansion follows Algorithm 2 in [10]. The final trust region has a Lipschitz constant of 0.93 and the error bound is 1160.

C. Action Plan Verification and Selection

We use a VLM as the verifier to evaluate each sampled action plan’s visual outcome in the context of the task instruction ℓ . The VLM receives a sequence of predicted observations from the world model (sampled as four concatenated frames) and is prompted to identify and reject plans that pose safety concerns—such as collisions with novel distractors or non-target objects—and to select the outcome that best aligns with the user’s intent. If none of the proposed plans are deemed safe or suitable, the VLM can optionally escalate by requesting human intervention.

Plan verification and selection accuracy. We evaluate two off-the-shelf vision-language models (VLMs), GPT-4o and LLaMA 4, as verifiers without any task-specific fine-tuning. Table IV reports two metrics for each VLM: verification accuracy and selection accuracy. Verification accuracy measures



(GPT-4o) Plan Analysis:

Conclusion: Plan is not successful.

Failure reason: The robot does not pick up the pepper, nor does it attempt to place it on the pan.

(Llama-4) Plan Analysis:

The gripper contacts the green pepper. No collisions occur with the pot, spoon, or scale, so the motion is both safe and effective.

Verdict: the plan is successful and meets the task requirements.



(GPT-4o) Plan Analysis:

Conclusion: Plan is not successful.

No attempt to grasp or move the pepper. No interference with spoon, pan, or pot.

(Llama-4) Plan Analysis:

Task status: Failure – the pepper is never picked up, let alone placed on the pan.

No obvious collisions with the pot, spoon, or scale, but the robot spends time gripping a large box that isn’t part of the task, so it’s both inefficient and incorrect.

Fig. 11: Qualitative comparison of VLMs’ plan verification capabilities.

	GPT-4o	LLaMa 4
Verification Accuracy	0.94	0.82
Selection Accuracy	0.85	0.64

TABLE IV: Plan verification and selection accuracy.

the percentage of trials in which the VLM correctly rejects an unsafe or failed action plan. Selection accuracy measures the percentage of trials in which the VLM either correctly selects the most desirable action plan or correctly rejects all options if none are safe. Both metrics are computed based on visual outcome predictions generated by **ReOI**, with ground-truth labels provided by a human expert. In Figure 11, we show response comparisons from the GPT-4o and LLaMa-4.

Prompt template. Figure 12 illustrates the system prompt used during the verification stage, along with example responses from the VLM.



You are a helpful assistant aiding a robot in selecting a safe and effective action plan. You will be shown multiple image grids, each depicting a possible future trajectory of the robot. The robot is a yellow arm with 7 joints and a gripper.

Your task is to analyze each predicted future and identify the plan that safely and successfully completes the task described below. Watch out for the potential collisions between the robot and other non-target objects.

If none of the plans are safe, return None. Otherwise, return the ID of the best plan.

Task description: The robot needs to pick up the pepper and place the pepper on the pan.

Plan 1



Plan 2



Plan 1 Analysis:

✅ **Attempts pickup:** The robot reaches toward the pepper and seems to lift it.

⚠️ **Possible collision risk:** In the second and third frame, the gripper appears very close to the nearby black cooker, raising mild concern about spatial safety.

Plan 2 Analysis:

✅ **Successful pickup:** The gripper clearly lifts the pepper.

✅ **Accurate placement:** The robot hovers over the pan and releases the pepper, which is no longer in the gripper in the final frame—strongly suggesting a successful drop into the pan.

✅ **No visible collisions:** Throughout the sequence, the arm avoids the spoon, bowl, and scale.

Recommended Plan: Plan 2

It safely and effectively completes the task without colliding with surrounding objects.



Fig. 12: Qualitative example of the VLM-based plan verification and selection.