

A. Related Work

Generative Imitation-Based Policies. With the rise of large-scale open-source datasets of expert demonstrations [2, 4, 5, 8, 13, 44], imitation learning (IL) has become a popular way to learn low-level robot control policies from data. In particular, recent advances in generative modeling have unlocked policy architectures that can model diverse, multi-modal behaviors directly from high-dimensional observations [6, 23, 33]. At the same time, generative IL policies still exhibit nuanced, hard-to-anticipate performance degradations during deployment time. These degradations range from complete task failures (e.g., inability to grasp a cup, knocking it down, or dropping it [27]) potentially due to distribution shifts or visual distractors [19, 46], to inappropriate behaviors that are misaligned with the deployment context or an end-user’s preferences (e.g., placing the gripper inside of a cup filled with water during grasping) [1]. In this work, our goal is to leverage the diverse low-level behavior distribution that the base policy has learned, but prevent these nuanced performance degradations at runtime via our novel policy steering method.

Failure Detection, Monitoring & Prediction. The handling of generative policy failures can be grouped into three broad categories: *posthoc* detection, *runtime* monitoring, and failure *prediction*. *Posthoc* approaches identify and explain failures present in offline robot datasets, and have recently leveraged Vision Language Models (VLMs) to accelerate this process via video captioning, highlighting critical data frames, and providing human-interpretable summaries of failures [10, 16, 26, 48]. In contrast, *runtime* monitoring aims to detect failures as they happen during robot deployment. To quickly identify nuanced failures, recent methods propose a “fast and slow” approach: a fast online detector flags unusual situations (e.g., binary anomaly classifier), while a slower VLM-based reasoner provides a deeper understanding of the event and if it is a relevant failure [1, 39]. Although these strategies can effectively identify failures, they fundamentally require the robot to start failing for the runtime monitor to activate. The final category, failure *prediction* methods, anticipate failures before they occur and unlock the potential for preemptive correction of the base policy. Here, existing approaches [22, 24, 25] often rely on out-of-distribution (OOD) detection in a latent space or dense human labels to train a binary classifier that distinguishes failures from successes. In this work, we contribute to the *predictive* category of methods. Our method anticipates future outcomes of the policy’s actions via a latent world model, and reasons about the outcomes via a VLM that is aligned with the predicted latent states.

Policy Steering. A traditional method to improve a base IL policy is to fine-tune it with additional intervention data [25] or recovery behaviors [37]. However, recently, runtime policy steering has become an attractive alternative to improving a generalist IL policy [27, 47] without needing any additional and expensive demonstration data. Runtime policy steering

assumes that the base policy is capable of generating the correct behaviors, but fails to select them reliably. Here, an external verifier can be used to re-rank (i.e., “steer”) the generations towards ones with good outcomes. Previous methods have explored humans-in-the-loop [47] or a Q -function learned from very large offline datasets labeled with sparse rewards. [27] as the verifier. In our framework, by using a VLM-in-the-loop as our verifier, we can perform policy steering autonomously after finetuning VLM on a small dataset and provide human-like interpretable guidance.

Learning to Search. From an algorithmic perspective, our approach fits within the paradigm of *learning to search* (L2S) [31]. In L2S, one learns the components required to, at test time, plan a sequence of actions, rather than merely imitating what was in the training data. L2S provides two key advantages over direct imitation algorithms like behavioral cloning. First, the agent gains the ability to reason about the consequences of its actions and potentially recover from mistakes, avoiding *compounding errors* [35, 41]. We see this manifest in our system’s ability to avoid or correct from failures the base policy would have produced otherwise. Second, *verifying* whether a plan is good is often easier to learn than *generating* a good plan in the first place [43, 28]. We see this manifest in the fact that our system only requires a limited amount of data to fine-tune our verifier VLM, rather than the larger amount of data an end-to-end approach would have required [38].

In contrast to more classical L2S approaches that require extensive *global* search in the real world [31, 52], we instead perform *local* search [3] against a learned verifier [42, 12] inside a learned world model [32]. This allows us to avoid the computational expense of global search and the potential safety violations incurred by real-world interaction. As argued by [42, 32], doing so still matches many of the guarantees of classical L2S methods if one fits the world model on a mixture of on-policy and off-policy trajectories (i.e., in a *hybrid* fashion [40, 45]), as we do consistently across all of our experiments.

B. Limitations

While **FOREWARN** exhibits strong policy steering across diverse task settings, it is not without limitations. First, it assumes base policy is sufficiently competent—i.e., already containing the correct behavior. Future work should investigate how to detect if none of the policy’s generated action plans are suitable for the deployment context, and how to improve the base policy via targeted fine-tuning data.

A second limitation lies in modeling real-world interaction dynamics. Our component-level analysis in App. D2 revealed that our system’s primary failures stem from the world model’s imprecise “imagination”, exacerbated by our limited training data. More advanced visual features (e.g., DINO features [30]) might improve the world model’s robustness to visual distractors. Ongoing research on large-scale, generalizable world models [49] for manipulation may also inform future extension of this framework.

Another bottleneck of the current system is the inference overhead, a common challenge for large autoregressive models

like LLMs and world models. Some approaches to this include: 1) hierarchical decomposition of *FOREWARN* running high-level reasoning at low frequency while maintaining low-level control at high frequency like the common practice of hierarchical Vision Language Action Models (VLAs), 2) using advancement in quantization and caching techniques, 3) distilling our latent-aligned VLM into a model with smaller size, 4) strategically allocate inference time by identifying keypoints when it is “worth” reasoning for longer or deliberately.

Finally, our VLM is built upon an open-source Llama-3.2-11B-Vision-Instruct model, whose visual reasoning capabilities lag behind its language-based commonsense reasoning. A stronger backbone could yield higher-quality behavior descriptions, especially if it excels at video captioning tasks.

C. Algorithm & Implementation Details

1) Base Policy:

We use Diffusion Policy [6] as our robot action generation model due to its strong ability to capture multimodal and complex robot behaviors.

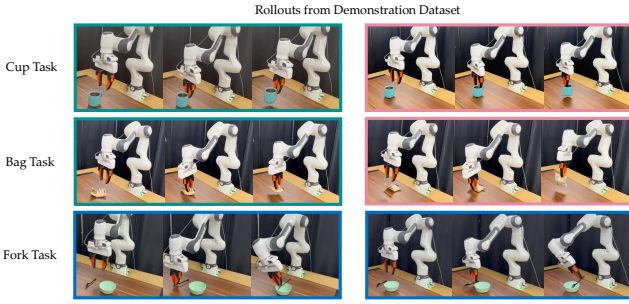


Fig. 4: **Multimodality in Demonstration Datasets.**

Training dataset. For each task, we collect 100 multimodal demonstrations and train the policy on an NVIDIA A6000 GPU following the procedure in [6]. In **Cup Task**, the training dataset consists of 50 demonstrations where the robot grasps the cup by the rim while touching the inner surface, and 50 demonstrations where the robot grasps it by the handle. In **Bag Task**, the training dataset consists of 50 demonstrations where the robot grasps the middle part of the bag and 50 demonstrations where it grasps the top edge without crushing the chips. In **Fork Task**, the training dataset consists of 4 modes, including 1) picking up the fork by tines and dropping high; 2) picking up the fork by the handle and dropping high; 3) picking up the fork by the tines and dropping low; 4) picking up the fork by the handle and dropping low. Each mode has 25 demonstrations. Figure 4 presents multimodal demonstrations for each task, and Table II details the hyperparameters used for training.

Observation and action spaces. The robot’s policy receives inputs from a wrist-mounted camera, a third-person camera, and proprioceptive states—including the end-effector position,

Hyperparameter	Value
State Normalization	Yes
Action Normalization	Yes
Image Chunk	2
Image Size	256
State Dimension	8
Action Dimension	10
Action Execution Horizon (Bag)	140
Action Execution Horizon (Cup)	120
Prediction Horizon (Bag)	120
Prediction Horizon (Cup)	140
Batch Size	100
Training Epochs	600
Learning Rate	1e-5
Number of Worker	16
Train Diffusion Step	100
Inference Diffusion Iteration	16

TABLE II: **Base Policy Hyperparameters.**

orientation quaternion relative to the base, and gripper opening—to predict actions that control the end-effector’s absolute pose and gripper opening.

Action plan aggregation. During policy steering, we aggregate 100 sampled action plans into 6 modes. We represent each action trajectory as a $T \times 7$ array, where T is the trajectory length, and each action consists of the gripper’s position (3D coordinates) and orientation (quaternion with four components). To cluster these trajectories into 6 modes, we apply Time Series K-Means with Dynamic Time Warping (DTW) as the distance metric. This allows us to group similar motion patterns while accounting for temporal variations in trajectory execution. The average trajectory within each cluster is used as the aggregated action plan. In the subsequent stages of our system, these 6 aggregated action plans serve as candidate options for policy steering selection.

2) World Model:

Motivation. The effectiveness of world models has been demonstrated across various embodied domains [25, 50]. In our problem setting, it provides several key advantages: 1) it grounds low-level actions, difficult for a VLM to interpret—by predicting future image observations from action plans, bridging the gap between low-level actions and visual observations; 2) it compresses information into latent states that not only retain essential details for high-quality image decoding but also effectively predict the next latent state given an action.

Architecture. We use DreamerV3, a state-of-the-art recurrent world model from [18]. Our world model, denoted as $\mathcal{W}_\phi = (\mathcal{E}_\phi, f_\phi, \mathcal{D}_\phi)$, consists of three key components: an encoder network \mathcal{E}_ϕ , a recurrent dynamics model f_ϕ that operates over a stochastic continuous latent space, and a decoder network \mathcal{D}_ϕ that projects latent embeddings back into observations.

Below, we provide a detailed definition of each module:

$$z_\tau := \begin{cases} \mathcal{E}_\phi(o_\tau^i), & \tau = t. \\ \mathcal{E}_\phi(o_\tau^i, z_{\tau-1}^i, a_{\tau-1}^i), & t < \tau < t + T. \end{cases} \quad (4)$$

$$\hat{z}_{\tau+1}^i \sim \begin{cases} f_\phi(z_\tau^i, a_\tau^i), & \tau = t. \\ f_\phi(\hat{z}_\tau^i, a_\tau^i), & t < \tau < t + T. \end{cases} \quad (5)$$

$$\hat{o}_\tau^i := \mathcal{D}_\phi(\hat{z}_\tau^i), \quad t \leq \tau < t + T. \quad (6)$$

Hyperparameter	Value
Observation Normalization	Yes
Action Normalization	Yes
Image Size	(64, 64)
Batch Length	64
Batch Size	16
Training Step (Cup)	100000
Training Step (Bag)	150000
Hidden State h Dimension	512
Stochastic Representation z Dimension	32
Dynamics Loss Ratio α_{dyn}	0.5
Representation Loss Ratio α_{rep}	0.1
Reconstruction Loss Ratio α_{pred}	1.0
CNN Encoder Depth	32
CNN Encoder Kernel Size	4

TABLE III: World Model Hyperparameters.

Training. The world model \mathcal{W}_ϕ is pretrained using an offline dataset of policy’s rollouts and demonstrations $\mathbb{D}_{WM} = \{\{(o_\tau^j, a_\tau^j, o_{\tau+1}^j)\}_{\tau=0}^{H-1}\}_{j=1}^M$. Apart from 100 demonstrations, we collect 250 rollouts for each task, including both the success and failure experiences of the generative policy π deployed on the real robot.

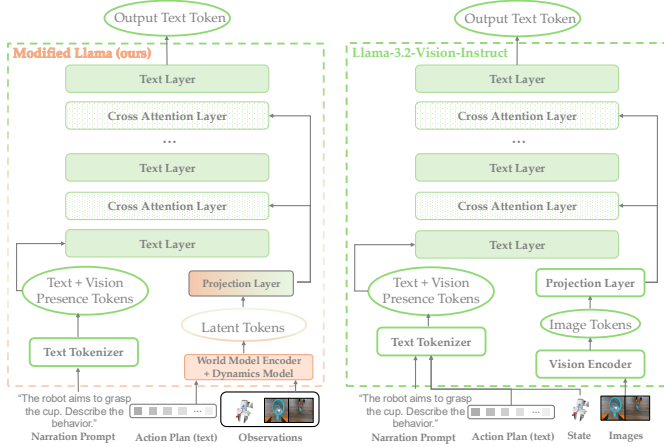


Fig. 5: VLM Architecture. On the left is detailed architecture **FOREWARN**, an modified VLM with an explicit world model. On the right is original Llama-3.2-Vision-Instruct model, which is used as our baseline **VLM-Act**. This is an elaborated version of Fig. 2

Since offline datasets lack reward signals, we omit the reward loss. Instead, the pretraining of the world model is supervised using a modified loss function: $\mathcal{L}_{\mathcal{W}} = \alpha_{dyn} \times \mathcal{L}_{dyn} + \alpha_{rep} \times \mathcal{L}_{rep} + \alpha_{pred} \times \mathcal{L}_{pred}$. Here, the dynamic loss

\mathcal{L}_{dyn} incentivizes accurate forward predictions in the latent space while \mathcal{L}_{rep} ensures that latent states are informative for reconstructing observations and learning good representation. Finally, the prediction loss \mathcal{L}_{pred} minimizes the error between decoded observations of the predicted latent states and the ground-truth observations. The exact loss calculation is shown in [18]. The hyperparameters used for training world model is shown in Table III.

During training, the decoder \mathcal{D}_ϕ reconstructs observations from the latent space to compute the prediction loss and the visualization helps us select the best world model.

Hyperparameter	Value
LoRA (rank)	8
LoRA (dropout)	0.05
LoRA (alpha)	32
Precision	bfloat16
Batch Size	10
Learning Rate	1e-4
Epoch (Cup)	10
Epoch (Bag)	15
Finetuned Layers	["down_proj", "up_proj", "o_proj", "k_proj", "q_proj", "v_proj", "gate_proj", "linear_proj"]

TABLE IV: VLM Hyperparameters of **FOREWARN**

Deployment. Because we cannot reset the real-world environment to the exact same state, each action sequence \mathbf{a}_t^i yields only a single observation sample o_t^i . As a result, the expectation in Eq. 3 is approximated in practice. During finetuning, we make the dynamics model f_ϕ deterministic by taking the mean of its predicted distribution. Although the model provides a 64-step prediction horizon, we downsample these future latent states to 16 to reduce redundancy from minimal changes across adjacent steps. Each latent state is formed by concatenating the hidden state and the stochastic representation, and we keep the world model frozen throughout VLM finetuning and deployment.

You are an assistant for monitoring the robot's behavior.

Given the provided task description input and the images in time order of robot execution(combined wrist and third person view), respond with only one sentence that best describes the robot's behavior.

Your response must strictly align with the image observations. Do not include any additional text, explanations, or information.

Input: {general task description}

<latent token> x 16

Fig. 6: Prompt Template for Behavior Narration in **FOREWARN**.

3) Vision Language Model:

Architecture. We use Llama-3.2-11B-Vision-Instruct model as our VLM backbone. We modify the original Llama Model to incorporate the explicit world model to predict outcomes of the action plans first and then use VLM to reason about

the latent states to generate behavior narrations. The modified architecture is visualized in Fig. 5. Specifically, we replace the original vision encoder (ViT) of Llama with our world model’s encoder as well as dynamics model, and project latent states as text tokens. We use one single linear layer to project latent tokens to text tokens.

You are an assistant providing help for household tasks. Based on the task description and the possible modes provided, your job is to select the best behavior mode that fulfills the task goal. The most important rule that needs to be considered first is in task description. Assume that each mode will be executed exactly as described, with no modifications.

Output Requirements:

- If neither mode fully meets the task goal or all modes present a high risk, you reject all of them.
- If exactly one mode is suitable, you choose this one.
- If multiple modes are equally suitable and do not conflict with constraints, you can choose any of them.

Response Format:

- Return the selected behavior mode by repeating the mode description. Use 'none of them' for mode description if no mode is selected.
- Include one sentence of explanation for the choice made.

Task Description:[general task description]

Task Condition: [task context]

Behavior Modes:

- 'Mode 1. [Action Plan 1]'
- 'Mode 2. [Action Plan 2]'
- 'Mode 3. [Action Plan 3]'
- 'Mode 4. [Action Plan 4]'
- 'Mode 5. [Action Plan 5]'
- 'Mode 6. [Action Plan 6]'

Fig. 7: Prompt Template for Policy Steering.

Finetuning. We adopt LoRA to finetune our modified model, loading the base weights from Llama-3.2-Vision-Instruct and randomly initializing the new linear projection layer. This approach updates only 0.2664% of the original model’s parameters. The finetuning hyperparameters are listed in Table IV, and we select the model with the lowest evaluation loss for deployment.

You are an assistant responsible for failure monitor during household tasks. Your task is to evaluate whether the policy’s behavior is a success or failure based on the task description and task condition. The most important rule is in task condition. Assume the robot policy is going to be executed as described, except there is a clear failure stated in behavior mode. Do not infer unintended consequences or failure scenarios beyond what is explicitly described.

Output Requirements: - If the behavior mode is likely to achieve the desired goal in the task description, you confirm it is normal. Remember only when the behavior mode indicated failure given task condition or clearly against the task goal, you output abnormal.

Response Format:

- Return either 'normal' or 'abnormal' as the final judgment. Provide a one-sentence reason for the decision made, without speculating on additional failure scenarios. Your estimation should reason only with task condition. Don’t be overconservative.

Provided Behavior Mode: 'Mode 1. [Action Plan]'

Task Condition:[task context]

Task Description:[general task description]

Fig. 8: Prompt Template for Failure Monitoring.

Prompt. The modified VLM is finetuned to generate behavior narration with the prompt template in Fig. 6, explaining the predicted latent states from the world model.

In policy steering, we sample and aggregate action sequences into 6 modes to query VLM for the best choice. Each mode’s behavior narration replaces Action Plan in the prompt template in Fig. 7.

Another potential usage of our system is to evaluate and monitor different policies’ performances under specific task description before execution. To showcase our system can be a reliable failure monitor across different tasks, we conduct additional experiments in App. D2. We list the prompts used for failure monitor in Fig. 8.

Different tasks have different task descriptions, and specifications (i.e., contexts), which are put in the prompt for policy monitoring and policy steering can be found in Sec. IV

You are an assistant for monitoring the robot’s behavior. Given the provided task description input, the image of robot at the first step (combined wrist and third person view), the robot gripper state at the first step and the next 16 actions of the robot, respond with only one sentence that best describes the robot’s future behavior of these 16 actions.

Note:

1. Remember to pay attention to the details of grasping part of the object, e.g. handle, rim. If gripper is not contacting the cup or the cup is lying down on the table, describe the behavior and must state it as a failure in the output.
2. Both the robot state and the actions have 8 dimensions. The first three dimensions are the x, y, z positions of the robot gripper. X direction is moving forward and backward in the image and Y is moving left and right. Z is moving up and down. The 4th to 7th values are the quaternion of the gripper. The last value is gripper opening or closing.
3. Your response must strictly align with the image observation. Do not include any additional text, explanations, or information.

Action Sequences: [Action Plan]

Current State: [proprioceptive states]

Input:[general task description]

Fig. 9: Prompt Template for Behavior Narration for **VLM-Act** in Cup Task

Hyperparameter	Value
LoRA (rank)	8
LoRA (dropout)	0.05
LoRA (alpha)	32
Quantization	4bit
Batch Size	1
Learning Rate	1e-5
Epoch (Cup)	10
Epoch (Bag)	15
Finetuned Layers	["down_proj", "up_proj", "o_proj", "k_proj", "q_proj", "v_proj", "gate_proj"]

TABLE V: VLM Hyperparameters of **VLM-Act**.

4) Additional Details of Baselines:

VLM-Act. This baseline is an ablated version of **FOREWARN** without the explicit world model. It uses the original Llama-3.2-11B-Vision-Instruct model as shown in right part of Fig. 5 and finetuned with the same labels as in **VQA Dataset**. The action plans and states are prompted as text and image observations are concatenated together and processed as image tokens. Details of the prompt are available in Fig. 9. Since VLM cannot directly interpret the low-level action control, we give some privileged information in the prompt, marked as red in the prompt template, to help it generate behavior narration. However, this baseline still struggles to generate accurate behavior narration as shown in Sec. D2, despite being finetuned on the same dataset. For policy monitoring and policy steering, **VLM-Act** uses the same prompt as **FOREWARN**. Hyperparameters used for finetuning are shown in Table. V

VLM-Img & VLM-Img-Oracle. We further evaluate an advanced VLM (GPT-4o) to interpret fine-grained motion details from a sequence of 16 images, either reconstructed from predicted latent states or recorded from actual execution. GPT-4o runs at the default temperature. As shown by the red text in Fig. 10, GPT-4o still struggles to comprehend subtle motion details, even when given privileged information to guide its focus.

Classifier-Dyn-Latent. For policy steering and monitoring, we adopt a causal transformer-based binary classifier [25] to

You are a helpful assistant that describes robot behaviors.

Task Description: {general task description}

Here are sixteen images(third person view and wrist camera view) of robot execution listed in time order and a task description. Please generate a one-sentence description of the robot's behavior.

Notes:

1. Pay attention to the grasping process as it shows the grip part.
2. Do not hallucinate over the contact position.
3. If the grasp is achieved, the sentence should focus on which part of the object it grasps and makes contact with, e.g. handle, inner surface, etc.
4. If the cup is not grasped in the robot's gripper, the sentence should describe the failure.

<image token> x 16

Fig. 10: Prompt Template for Behavior Narration for GPT-4o in Cup Task.

Hyperparameter	Value
Embedding Dimension	64
Number of Head	1
Attention Dropout	0.05
Embedding Dropout	0.05
Block Output Dropout	0.05
Context Length	16
Sinusoidal Embedding	True
Learning Rate	1e-4
Gradient Clip	$(-\infty, 100]$
Epochs	30

TABLE VI: Hyperparameters of Classifier-Dyn-Latent.

directly predict success or failure from future latent states. Table VI summarizes its hyperparameters. We manually label each rollout as success or failure under the first task description to train the classifier.

VLM-DynLat-Binary. This end-to-end baseline also predicts a binary success/failure label from future latent states, but employs the same modified VLM architecture as **FOREWARN**, which provides a larger capacity than the transformer-based classifier. It is trained on the same dataset as **Classifier-Dyn-Latent**, with the prompt template shown in Fig. 11.

You are an assistant for failure detection. Analyze the robot behaviors from the provided images presented in time order (combined wrist and third-person view). Compare the observed behaviors with the provided task description and task condition to determine whether the task was successful or a failure.

Respond with only one of the following numbers: 0 or 1. 0 means task failure. 1 means task success.

No additional words, explanations, or clarifications are allowed. Your output must be limited to a single digit (0 or 1) and nothing else.

Task Description: {general task description}

Task Condition: {task context}

<latent token> x 16

Fig. 11: Prompt Template for Policy Monitoring for VLM-DynLat-Binary.

VLM-DynLat-Category. Similarly, we develop an end-to-end approach that directly predicts a valid set of action plans for policy steering. The modified VLM is finetuned to output which action-plan indexes are valid under the current task description. Its input includes a text prompt and six sequences of predicted future latent states (Fig. 12), each corresponding to one candidate action plan.

D. Experiment

1) Real Robot Setup:

Fig. 13 demonstrates the setup of our real-world experiments.

You are an assistant for policy steering.

The first behavior mode is observed as the following images: <latent token> x 16.
The second behavior mode is observed as the following images: <latent token> x 16.
The third behavior mode is observed as the following images: <latent token> x 16.
The fourth behavior mode is observed as the following images: <latent token> x 16.
The fifth behavior mode is observed as the following images: <latent token> x 16.
The sixth behavior mode is observed as the following images: <latent token> x 16.

Analyze six sequences of robot behaviors from the provided images presented in time order (combined wrist and third-person view). Compare each sequence with the provided task description and task condition to determine which sequences successfully satisfy the task requirements.

Instructions:

- Respond with a list containing the indices of the valid sequences (e.g., [0,1,2]).
- If none of the sequences satisfy the task requirements, return an empty list: [].
- The indices should be between 0 and 5.
- No additional words, explanations, or clarifications are allowed.
- Your output must be strictly a list containing the indices of the valid sequences or an empty list.

Task Description: {general task description}

Task Condition: {task context}

Fig. 12: Prompt Template for Policy Steering for VLM-DynLat-Category.

We employ two cameras, a RealSense D435 camera on the Franka hand and a Zed mini 2i camera placed in front of the robot. In order to increase the contact region and compliancy, we replace the original Franka gripper finger with 3D printed gripper finger from [7].



Fig. 13: Real Robot Setup. The visualization of the real robot environment and the positions of the cameras.

2) Supplementary Experiments & Analyses:

From Action Rollouts to Behavior Narration.

As discussed in Sec. III-B, if we want to use the VLM as an open world verifier $R_v^{\text{VLM}}(\cdot; \ell)$, we need to enable the model to understand the underlying textual representation of low-level action outcomes. In this section, we study if our latent-aligned VLM can accurately describe the outcomes of low-level actions. We also compare our approach with several baselines to investigate the advantages of using an explicit world model for predicting action outcomes and decoding a robot's action plans into behavior narrations.

Baselines. We compare our approach, **FOREWARN**, against four baselines (more implementation details provided in App. C4). (1) **FOREWARN-Oracle**, is an upper-bound on

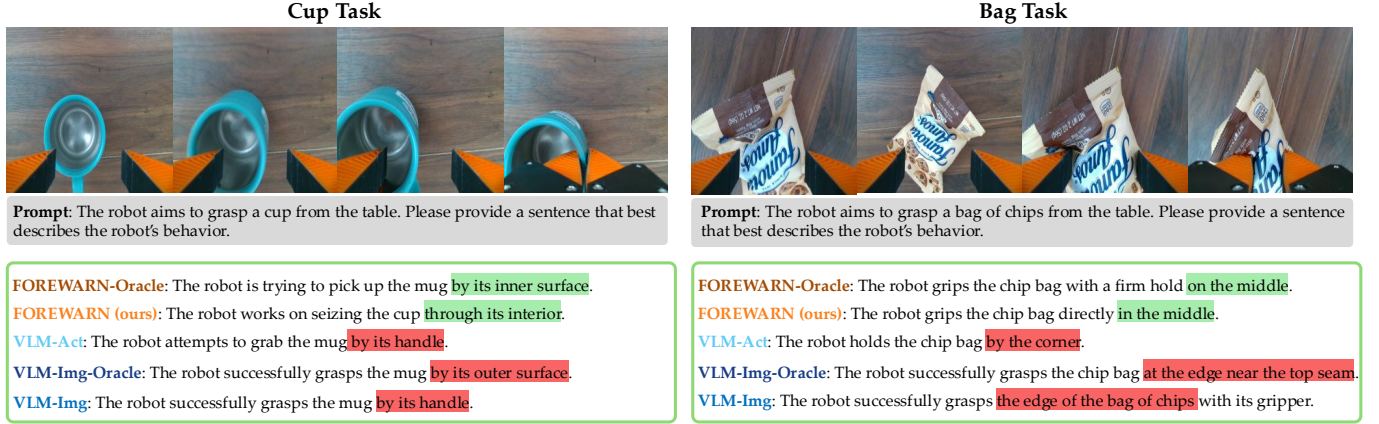


Fig. 14: **Examples of Behavior Narrations Predicted by Each Approach.** The top row displays the ground-truth robot observations and the prompt used for querying VLMs. Only **FOREWARN** and **FOREWARN-Oracle** consistently produce accurate outcome narrations, effectively capturing nuanced motion details. In contrast, the baselines frequently hallucinate or fail to capture critical contact details between the gripper and objects. For instance, in the **Bag** task, **VLM-Act**, **VLM-Img**, and **VLM-Img-Oracle** all hallucinate that the robot is grasping the edge of the bag, whereas it is actually grasping the middle.

our method’s performance assuming that we had access to *ground-truth* future observations (instead of relying on the latent dynamics f_ϕ to predict future outcomes). This method uses the encoder \mathcal{E}_ϕ on ground-truth future observations to get privileged (posterior) future latent states $z_{t:t+T}$ as input for the VLM. (2) **VLM-Act**, which directly fine-tunes the original Llama-3.2-11B-Vision-Instruct model to generate behavior narrations end-to-end from the current observation o_t and an action plan $a_{t:t+T}$ (represented as text), without explicitly predicting outcomes. (3) **VLM-Img**, which utilizes the *decoded* world model’s predictions (i.e., the predicted future visual observations) given a robot’s planned actions. We use GPT-4o [29] to process the predicted visual observations and generate behavior narrations in a zero-shot manner. (4) **VLM-Img-Oracle**, which is similar to **VLM-Img** but is an upper bound on this method by using ground-truth visual observations instead of predicted ones.

Metrics. We adopt the metrics from [10] to evaluate the alignment between predicted behavior narrations and ground-truth narrations: (1) **LLM Score**: A similarity score (ranging from 0 to 1) determined by the GPT-4o model. (2) **GT Accuracy**: A binary score (0 or 1) indicating whether the predictions match the ground-truth narrations, as determined by a human labeler (in this case, the authors). For further details on the motivation behind using these metrics for evaluation, please refer to App. D3.

Results: On the Value of Explicit Action Outcome Prediction. Table VII presents the **GT Accuracy** and **LLM Score** for our approach and each baseline. The results are averaged across 30 test rollouts for each task. The results show that **VLM-Act** performs poorly, achieving less than 50% **GT Accuracy** across all tasks. This underperformance is due to its inability to interpret low-level actions without the grounding provided by a world model’s future outcome

	GT Accuracy \uparrow			LLM Score \uparrow		
	Cup	Bag	Average	Cup	Bag	Average
FOREWARN-Oracle	0.92 \pm 0.02	0.77 \pm 0.03	0.85 \pm 0.03	0.86 \pm 0.01	0.76 \pm 0.03	0.81 \pm 0.02
FOREWARN (Ours)	0.87\pm0.02	0.75\pm0.03	0.82\pm0.03	0.82\pm0.02	0.72\pm0.02	0.76\pm0.02
VLM-Act	0.37 \pm 0.03	0.36 \pm 0.06	0.37 \pm 0.05	0.52 \pm 0.05	0.50 \pm 0.07	0.51 \pm 0.06
VLM-Img-Oracle	0.61 \pm 0.04	0.43 \pm 0.03	0.52 \pm 0.04	0.65 \pm 0.02	0.62 \pm 0.02	0.64 \pm 0.02
VLM-Img	0.36 \pm 0.05	0.33 \pm 0.04	0.35 \pm 0.05	0.56 \pm 0.03	0.60 \pm 0.04	0.58 \pm 0.04

TABLE VII: **Alignment Between Predicted Behavior Narrations and Ground-Truth Narrations.** **FOREWARN** outperforms all baselines across both tasks and achieves performance comparable to **FOREWARN-Oracle**, which has access to ground-truth action outcomes and represents the upper bound for our approach. We use 50 rollouts to evaluate the performance. For **FOREWARN**, **FOREWARN-Oracle** and **VLM-Act**, the mean and standard deviation are reported by running 3 seeds for the finetuning experiments while **VLM-Img** and **VLM-Img-Oracle**, report 3 queries of GPT-4o.

predictions. In contrast, **FOREWARN**, which leverages an explicit world model, outperforms **VLM-Act** by over 50% on every task, despite both being fine-tuned on the same dataset. These results demonstrate that decoupling the VLM’s burden of predicting action outcomes enables the model to produce more accurate outcome narrations than directly training the VLM to both predict outcomes and generate narrations end-to-end.

Results: On the Value of VLM Fine-Tuning. Interestingly, despite being among the most advanced VLMs, both **VLM-Img** and **VLM-Img-Oracle** struggle to accurately interpret robot behaviors directly from visual observations, even with access to ground-truth observations. As shown in Table VII, these methods fall behind **FOREWARN** by at least 30% in **GT Accuracy** and 16% in **LLM Score**. These results show that existing state-of-the-art VLMs struggle to decode fine-grained motion details from video observations, underscoring the importance of fine-tuning for improved performance in such tasks.

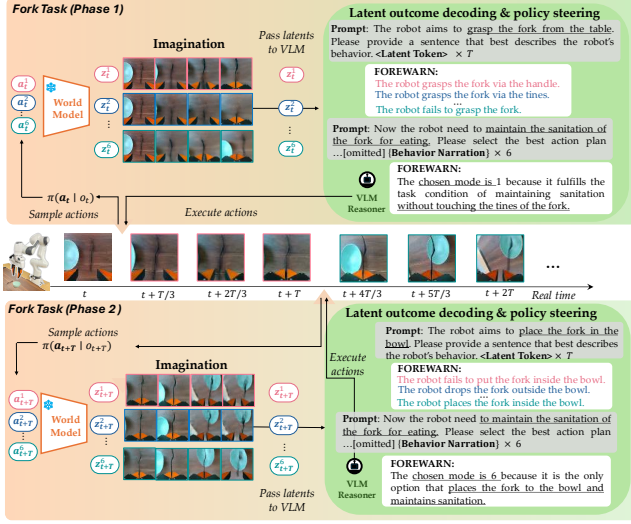


Fig. 15: **Policy Steering: Fork Task.** We visualize the steering process for the **Fork** task including two phases (Pick and Place). For each phase, we visualize the imagined T -step rollouts decoded from the world model for the 3 out of 6 action plans sampled from the base policy on the left. On the right, we show the behavior narrations generated from our finetuned VLM $\mathcal{T}_{\psi}^{\text{VLM}}$ and the VLM’s reasoning $R_{\psi}^{\text{VLM}}(\cdot; \ell)$ about the outcomes based on the task description ℓ and behavior narrations to select the best action plan to execute. The time axis shows the real-world execution of the selected behavior from the perspective of the wrist-camera.

Results: Qualitative Examples. In Figure 14, we visualize behavior narrations generated by our approach and the baselines. **FOREWARN** consistently produces more accurate outcome narrations, effectively capturing nuanced motion details. In contrast, the baselines often hallucinate or fail to capture critical contact details between the gripper and objects.

More Qualitative Examples for Policy Steering. We include additional qualitative examples for **Cup** and **Bag** tasks in Fig 16. These examples further demonstrate the effectiveness of our policy steering system for different tasks. The imagined image sequences from the world model show that our system is capable to predict the various outcomes for different action sequences and the VLM can reason about behavior narrations and correctly evaluate the action plans.

FOREWARN as a VLM-in-the-loop Failure Monitoring System. In this section, we present another application of our approach—preemptive failure monitoring—based on the behavior narrations generated in Sec. D2. Similarly, our modified VLM first decodes the predicted latent states from the world model, as behavior narrations $\hat{\ell}_b$. Then it reasons about behavior narrations under task description ℓ and decides whether the future action plan, translated as $\hat{\ell}_b$, is a failure. As both quantitative and qualitative results demonstrate, **FOREWARN** is a reliable and versatile failure monitoring framework across diverse task.

Baselines. We consider three methods as our baselines, which preemptively predict the outcome of the action plan before the execution. 1) **VLM-DynLat-Binary** takes the predicted latent states from the world model and task description ℓ as input to the VLM, directly generating binary output to indicate success or failure without the intermediate step of behavior narration; 2) **Classifier-Dyn-Latent** takes the predicted latent states as input and trains a transformer-based binary classifier to generate binary output with the same dataset as **VLM-DynLat-Binary**; 3) **VLM-Act** uses generated behavior narrations in Sec. D2 and queries the VLM again to decide if the behavior is a success or failure within the context of the task description ℓ . These baselines are equivalent to those in Sec. IV-A.

Metrics. To evaluate the overall performance of different methods as preemptive failure monitors, we report the standard detection metrics from prior work [1] including *Accuracy (ACC)*, *True Positive Rate (TPR)*, *True Negative Rate (TNR)*.

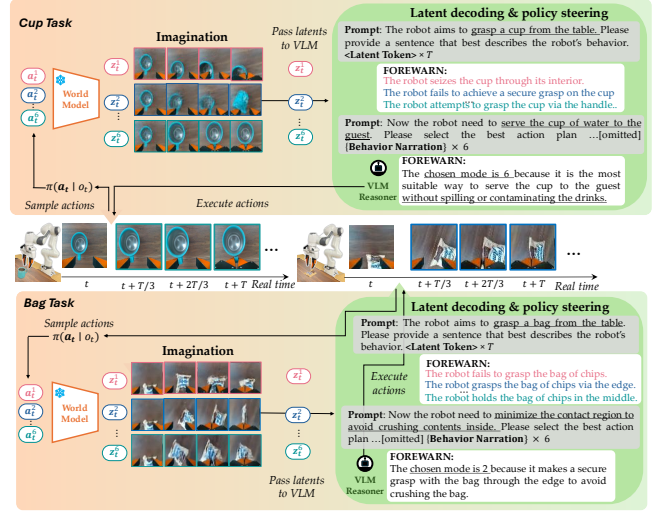


Fig. 16: **Policy Steering for Cup and Bag Task.** We visualize the steering process for the **Cup** task on the top and the **Bag** task on the bottom. For each task, we visualize the imagined T -step rollouts decoded from the world model for the 3 out of 6 action plans sampled from the base policy on the left. On the right, we show the behavior narrations generated from our finetuned VLM $\mathcal{T}_{\psi}^{\text{VLM}}$ and the VLM’s reasoning $R_{\psi}^{\text{VLM}}(\cdot; \ell)$ about the outcomes based on the task description ℓ and behavior narrations to select the best action plan to execute. The time axis shows the real-world execution of the selected behavior from the perspective of the wrist-camera.

Results: Failure Monitoring Performance. Both **VLM-DynLat-Binary** and **Classifier-Dyn-Latent** perform well when evaluation task description is the same as training. However, their performances drop sharply in novel task description, indicating poor generalization of end-to-end model even though they have the same model capacity as our method. In contrast, **FOREWARN** consistently attains high accuracy,

Method	Cup						Bag						Average					
	Training			Unseen			Training			Unseen			Training			Unseen		
	Acc↑	TPR↑	TNR↑	Acc↑	TPR↑	TNR↑	Acc↑	TPR↑	TNR↑	Acc↑	TPR↑	TNR↑	Acc↑	TPR↑	TNR↑	Acc↑	TPR↑	TNR↑
FOREWARN (Ours)	0.90	0.80	1.00	0.75	0.70	0.80	0.75	0.71	0.77	0.75	0.75	0.75	0.83	0.76	0.89	0.75	0.73	0.78
VLM-DynLat-Binary	0.85	0.77	0.91	0.15	0.11	0.27	0.75	0.86	0.69	0.40	0.38	0.42	0.80	0.82	0.80	0.28	0.25	0.35
Classifier-Dyn-Latent	0.90	0.80	1.00	0.10	0.10	0.10	0.80	0.71	0.85	0.35	0.13	0.50	0.85	0.76	0.93	0.23	0.12	0.30
VLM-Act	0.65	0.75	0.58	0.50	0.10	0.90	0.60	0.00	0.92	0.65	0.13	1.00	0.63	0.38	0.75	0.58	0.12	0.95

TABLE VIII: **Policy Monitoring.** The reported result in the table is averaged over 20 trajectories. **FOREWARN**, **VLM-DynLat-Binary** and **Classifier-Dyn-Latent** perform similarly well in training task description while **VLM-Act** has poor performance. In unseen task description, **FOREWARN** is the only method that maintains similar performance as training task description.



Fig. 17: **Policy Monitoring.** In the top (pink) and medium (green) row, the robot imagines correctly about the robot behaviors but only **FOREWARN** describes the behavior correctly and generates correct monitoring results with adequate explanations. In the bottom row (blue), all of the methods distinguish failure correctly from success.

Method	Accuracy ↑																	
	Independent						Training Task Description						Novel Task Description					
	World Model			Behavior Narration			Reasoning			Overall System			Reasoning			Overall System		
	Cup	Bag	Average	Cup	Bag	Average	Cup	Bag	Average	Cup	Bag	Average	Cup	Bag	Average	Cup	Bag	Average
FOREWARN	-	-	-	0.90	0.70	0.80	1.00	0.85	0.93	0.90	0.75	0.83	0.90	0.85	0.88	0.75	0.75	0.75
VLM-DynLat-Binary	0.80	0.75	0.78	-	-	-	0.95	0.80	0.88	0.85	0.75	0.80	0.20	0.45	0.33	0.15	0.40	0.28
Classifier-Dyn-Latent	-	-	-	-	-	-	1.00	0.90	0.95	0.90	0.80	0.85	0.05	0.25	0.15	0.10	0.35	0.23
VLM-Act	-	-	-	0.35	0.35	0.35	0.95	0.95	0.95	0.65	0.60	0.63	0.90	1.00	0.95	0.50	0.65	0.58

TABLE IX: **Performance Breakdown** for each component in our pipeline across four methods. **FOREWARN** have high accuracy across all components while **VLM-Act** has very low accuracy in narration, leading the poor performance of the overall system. both **Classifier-Dyn-Latent** and **VLM-DynLat-Binary** performs well in training task description and drop sharply in novel task scenarios.

higher than 75%, with balanced TPR and TNR across all tasks given different task descriptions, demonstrating its reliability and flexibility as a failure monitor.

The generalization capability of our method comes from decoupling the problem of failure monitoring as behavior narration and outcome evaluation.

This is also demonstrated by **VLM-Act**, which has the same intermediate step and shows balanced performance in both task descriptions, but **VLM-Act** often misclassifies behaviors in *Bag Task* as failures, resulting in low TPR and lower overall accuracy than **FOREWARN**.

In Fig. 17, we demonstrate monitoring results for all three different behaviors qualitatively. Across all three modes of behaviors, **FOREWARN** consistently generates accurate descriptions as well as correct monitoring results. **VLM-Act** is biased to generate failure narrations across all three modes, leading to wrong monitoring results. **Classifier-Dyn-Latent** and **VLM-DynLat-Binary** completely do not understand different action plans within different task descriptions. They generate the same monitoring results for totally different descriptions, contradictory to the actual execution.

Component-level Analysis for the System. We analyze each component in our method as well as all the baselines in (Table IX). **FOREWARN** shows high accuracy across all the components while the poor performance of **VLM-Act** is from the low-quality of narration directly generated from low-level action sequences. After finetuning, both methods preserve the strong reasoning capability of the VLM. However, **Classifier-Dyn-Latent** and **VLM-DynLat-Binary** has a huge performance drop in novel task scenarios because they are overfitting to the specific task description in the training.

3) Metric Ablations:

Metrics for Behavior Narration. We investigate four common text-generation metrics proposed in prior work [10]: *Cosine Similarity*, *ROUGE-L*, *LLM Fuzzy Matching*, and *Binary Success Rate*. To assess each metric’s correlation with ground-truth labels, we sample 16 narrations for each of three behaviors in the **Cup Task** (grasping by handle, grasping by rim, and grasp failures), yielding 360 intra-category and 768 inter-category comparisons. As shown in Figures. 18, 19, and 20, it is difficult for Cosine Similarity and ROUGE-L to cleanly distinguish narrations from the same category versus different categories, whereas *LLM Fuzzy Matching* with GPT-4o can easily separate them. This discrepancy arises because the narrations share similar high-level semantics (e.g., “grasping the cup”) and differ only in fine-grained details (e.g., grasp location). Consequently, we adopt **LLM Score** and **Ground-Truth Accuracy** (manual matching) as our final metrics for evaluating generated behavior.

Results: Policy Steering Speed. Our system queries the VLM twice to first generate behavior narrations and then select the best action plan. The overall inference time is 3.7 seconds among which the generation of behavior narrations

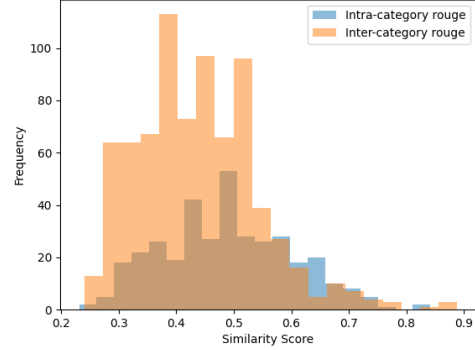


Fig. 18: **Distribution of ROUGE-L Score** shows that intra-category

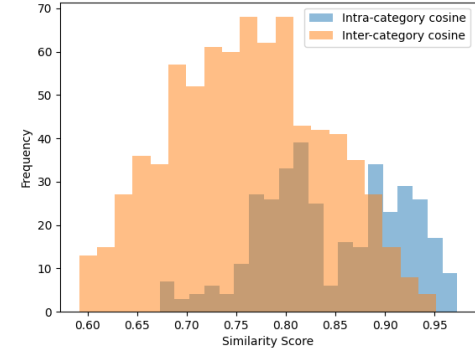


Fig. 19: **Distribution of Cosine Similarity Score** shows that intra-c

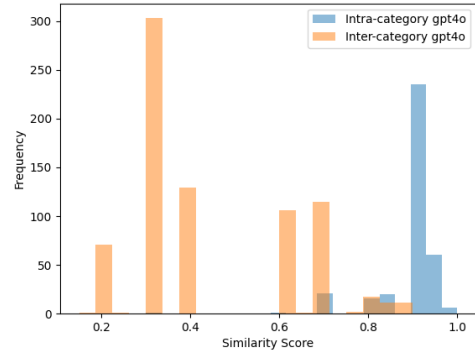


Fig. 20: **Distribution of LLM Score** shows inter-category and intra-category scores can be roughly separated at 0.7.

for 6 candidate action plans takes 1.3 seconds. In comparison, **VLM-Act** takes 22.0 seconds in total for VLM inference and behavior narration runs 19.7 seconds, much slower than **FOREWARN**, partially due to the usage of token per patch in images rather than a single token per state (image) like **FOREWARN**. Additionally, our world model and VLM communicate directly in latent space, avoiding image decoding from world model and encoding from VLM, which could further speeds up inference.

Method	Time (Seconds) ↓			
	World Model Prediction	Behavior Narration	Evaluation	Total
FOREWARN	0.1	1.3	2.3	3.7
VLM-Act	-	19.7		22.0

TABLE X: **Inference time for the Policy Steering System.** Inference time for each component in the system (averaged across 3 runs) shows that **FOREWARN** greatly reduces the time to generate behavior narrations from our modified VLM compared to **VLM-Act**.