# Scaling Verification Can Be More Effective than Scaling Policy Learning for Vision-Language-Action Alignment

Jacky Kwok[1,†] Xilun Zhang[1,†] Mengdi Xu[1] Yuejiang Liu[1,§]
Azalia Mirhoseini[1,§] Chelsea Finn[1,§] Marco Pavone[1,2,§]
[1]Stanford University [2]NVIDIA Research

https://cover-vla.github.io

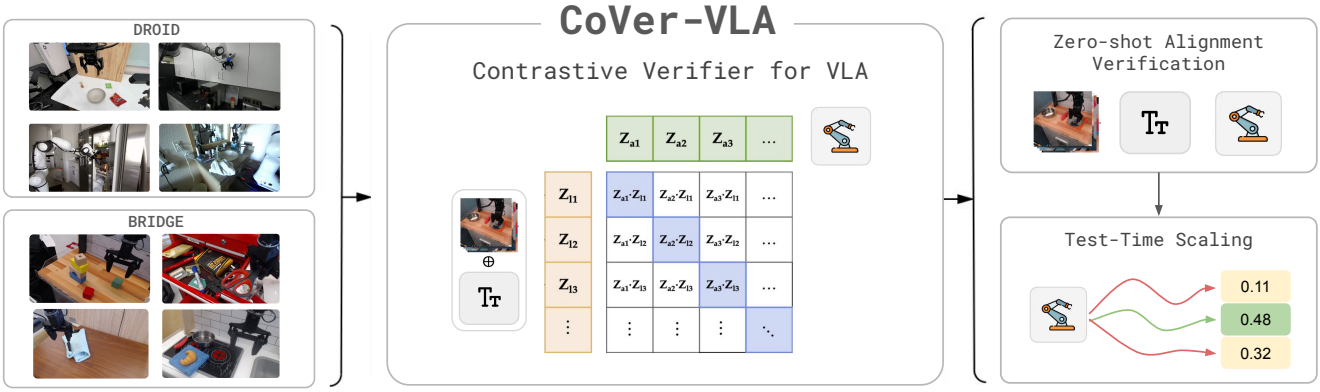


Figure 1. We present **CoVer-VLA**, a contrastive verification framework for vision–language–action alignment. CoVer is trained entirely offline on large-scale robotics datasets with contrastive representation learning. It supports zero-shot alignment verification for generalist robot policies out of the box. At test-time, CoVer can be used to perform instruction optimization and action verification, improving downstream performance for VLAs.

## Abstract

*The long-standing vision of general-purpose robots hinges on their ability to understand and act upon natural language instructions. Vision-Language-Action (VLA) models have made remarkable progress toward this goal, yet their generated actions can still misalign with the given instructions. In this paper, we investigate test-time verification as a means to shrink the "intention-action gap." We first characterize the test-time scaling law for embodied instruction following and demonstrate that jointly scaling the number of rephrased instructions and generated actions greatly increases test-time sample diversity, often recovering correct actions more efficiently than scaling each dimension independently. To capitalize on these scaling laws, we present CoVer, a contrastive verifier for vision–language–action alignment, and show that our architecture scales gracefully with additional computational resources and data. We then introduce "boot-time compute" and a hierarchical verification inference pipeline for VLAs. At deployment, our framework precomputes a diverse set of rephrased instructions from a Vision-Language-Model (VLM), repeatedly generates action candidates for each instruction, and then uses a verifier to select the optimal high-level prompt and low-level action chunks. Compared to scaling policy pre-training on the same data, our verification approach yields 22% gains in-distribution and 13% out-of-distribution on the SIMPLER benchmark, with a further 45% improvement in real-world experiments. On the PolaRiS benchmark, CoVer achieves 14% gains in task progress and 9% in success rate.*

## 1. Introduction

For robots to be useful in human-centric environments, they must be able to interpret and act upon natural language instructions. Vision-Language-Action (VLA) models, pre-trained on large-scale robotic datasets, have made significant progress towards this goal [4, 20]. However, their widespread deployment is hindered by a critical "intention-action gap": the misalignment between generated actions and the given language instructions. When the policy fails to follow the instruction, this gap can result in costly errors. For instance, a robot tasked with "putting a plastic container into a drawer" might correctly grasp the container but then fail to discriminate between the oven and a nearby drawer, mistakenly placing the container inside the oven. The container could melt or even catch fire. Addressing this fundamental misalignment is essential for deploying robots in real-world settings.

Existing efforts to close this gap have largely focused on scaling policy pre-training, such as augmenting training data with rephrased instructions [42] or employing larger VLM backbones [2, 11]. However, these approaches typically yield only incremental gains, and performance still degrades severely under simple perturbations [10, 18]. Moreover, scaling policy pre-training often leads to *catastrophic forgetting*, where learning action generation diminishes the VLM's multimodal understanding and reasoning, hindering generalization and

† denotes Equal contribution and § indicates Equal advising.

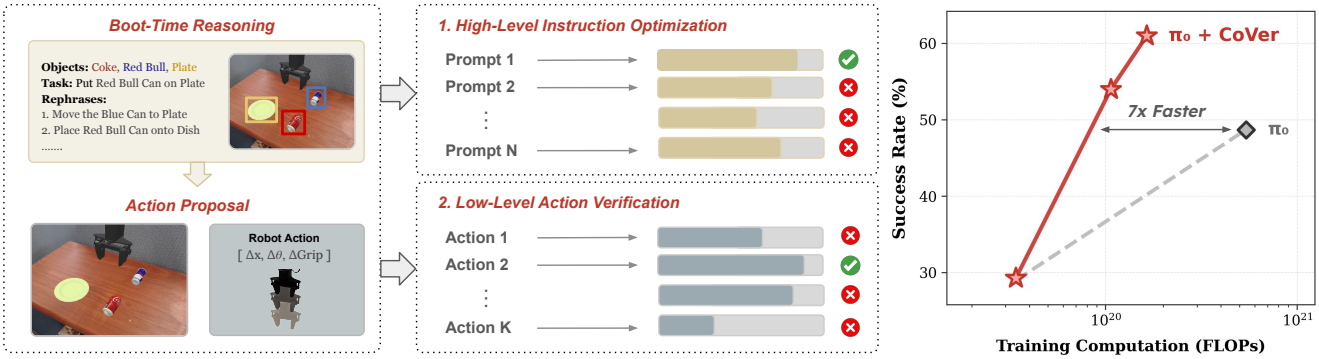


Figure 2. **Hierarchical Test-Time Verification Pipeline. Left:** Given the initial observation and language instruction, a VLM performs structured reasoning over the scene and precomputes a set of rephrased instructions during boot time. At each step during deployment, our framework generates a batch of action candidates for each instruction using a VLA. **Middle:** CoVer then scores all instruction–action pairs and selects the optimal high-level instruction and low-level action chunk for execution. **Right:** Compared to prior work on scaling policy learning [3], our approach achieves stronger performance while requiring substantially less compute. The reported training compute for $\pi_0$ includes both pre-training and fine-tuning on augmented instruction sets, whereas $\pi_0$ + CoVer accounts for pre-training $\pi_0$ and training the CoVer verifier on the same data.

semantic understanding [10, 14]. In this paper, we argue that VLA alignment can be more effectively improved through test-time scaling. More specifically, we ask in this work:

*Can we enable VLAs to leverage additional computation at test time to improve the alignment between their generated actions and the provided language instructions?*

The implications of answering this question extend not only to the generalization capabilities of VLAs, but also to how practitioners should trade off pre-training and test-time compute in robotics. To this end, we first characterize the test-time scaling law for embodied instruction following. Assuming the presence of an oracle verifier, we observe that action error consistently decreases as we scale the number of rephrased instructions, establishing a clear relationship between linguistic diversity and performance gains. Moreover, we demonstrate that jointly scaling the number of rephrased instructions and the generated actions constructs a more diverse action proposal distribution. This hybrid sampling approach often recovers correct actions more efficiently than scaling each dimension independently.

To leverage these scaling laws, we seek to develop a robust verifier for both instruction optimization and action verification. Existing verifiers often focus on low-level dynamics [22, 28] and require costly interactions with the environment [25]. To address this, we draw insights from cross-modal alignment [32, 37] and introduce CoVer, a **co**ntrastive approach for **ver**ifying the alignment across vision, language, and action. Our architecture employs two key components: a text-aware visual encoder that selectively extracts task-relevant features, and an action encoder that captures long-range temporal dependencies within action chunks. The results show that scaling the number of synthetic instructions, model parameters, negative samples, and verifiers in an ensemble consistently improves verification and downstream retrieval accuracy of CoVer. We train CoVer on 20 million offline samples using a 1B parameter backbone, producing a robust verifier for test-time scaling.

During deployment, our framework first leverages "boot-time compute" to let the robot reason offline. Given the initial observation and language instruction, a VLM performs structured reasoning over the scene—identifying relevant objects, spatial relations, and plausible task decompositions. The resulting reasoning traces are then used to precompute a diverse set of rephrased instructions, allowing the robot to avoid redundant rephrase generation during execution. At test time, we employ a hierarchical verification pipeline. This pipeline generates a batch of action candidates for each precomputed instruction with a VLA, scores all instruction-action pairs using CoVer, and then selects the optimal high-level instruction and low-level action chunks for execution. In summary, our contributions are as follows:

1. We characterize the test-time scaling law for embodied instruction following and propose a compute-efficient action sampling method.
2. We present a contrastive verifier for vision–language–action alignment and show that our architecture scales gracefully with additional computational resources and data.
3. We introduce boot-time compute for offline embodied reasoning and a hierarchical test-time verification pipeline that couples high-level prompt optimization with low-level action chunk selection.
4. We show that pairing VLAs with CoVer substantially improves downstream performance, achieving a 45% absolute improvement on real-world tasks, 18% on SIMPLER environments, and 9% on the PolaRiS benchmark.

## 2. Related Work

**Vision-Language-Action Models.** Recent VLA models, pre-trained on large-scale multimodal data and fine-tuned for visuomotor control, have demonstrated impressive generalization across tasks, objects, and environments [3, 20, 29, 34, 36]. Yet, they still struggle with instruction following: semantically equivalent rephrases can cause sharp drops in success [10, 18]. Some

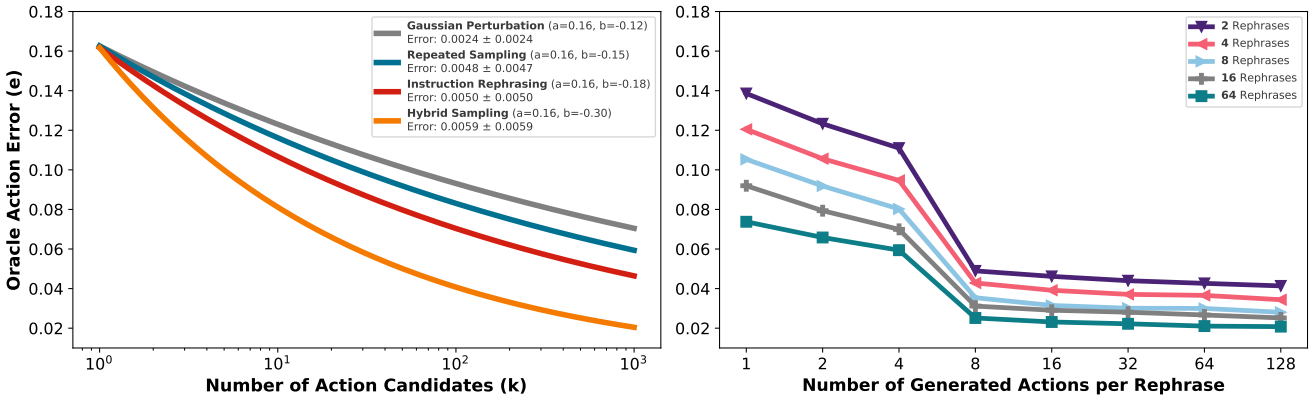


Figure 3. **Test-Time Scaling Law for Embodied Instruction Following.** Compared to prior methods that obtain diverse actions through repeated sampling [28] or Gaussian perturbations [22], we find that instruction rephrasing produces a broader set of action candidates, leading to improved recovery of the correct action. Furthermore, a hybrid test-time scaling strategy that increases both the number of rephrases and the number of sampled actions per rephrase is more effective than either strategy alone. We characterize each sampling approach using a power law, where the logarithm of oracle action error $e$ is a function of the number of action candidates $k$: $\log(e) \approx \log(a) + b \cdot \log(k)$.

recent work seeks to mitigate this issue by scaling up model capacity [24], expanding training data [12, 43], and introducing auxiliary objectives to preserve linguistic knowledge [8, 21]. Orthogonal to these training approaches, our work takes a test-time perspective: we treat a user instruction as a distribution over phrasings and verify resulting actions before execution.

**Test-Time Scaling.** Inference with additional compute has emerged as a promising paradigm for tackling challenging problems across diverse domains, including language reasoning [5, 27, 33, 35], visual understanding [40], and agentic planning [44]. In the context of robot learning, recent studies have demonstrated the effectiveness of optimizing over multiple candidate action sequences to enhance performance [28, 41], consistency [26], and robustness [22]. Such sampling processes can be further accelerated via guidance mechanisms in the latent space [38, 45]. Despite these advances, existing approaches still struggle with instruction following and often incur substantial computational overhead. Our method addresses these challenges through an action verification mechanism explicitly designed for instruction following while enabling acceleration through pre-computation.

**Action Verification.** Early work on action verification derives signals directly from the policy itself, e.g., prediction uncertainty [13, 42] and temporal consistency [1, 26], yielding lightweight ways to convert prior knowledge into a quality estimator. More recently, a growing body of work has focused on training explicit models for action verification, such as value functions [7, 15] and preference models [22]. Another line of work decomposes verification into two stages: predicting future states with a dynamics model [31, 41], and then assessing task progress in the predicted states. However, these techniques are still largely centered on low-level dynamics, while high-level instruction following remains a challenge. We instead formulate action verification as a contrastive alignment problem between language and behavior, explicitly targeting instruction-following quality.

## 3. Test-Time Scaling Analysis

In this section, we characterize the test-time scaling law for embodied instruction following, revealing how linguistic diversity in instructions affects downstream robot policy performance. Following the scheme introduced by Kwok et al. [22], we uniformly sample 1,000 $(s,\ a,\ I)$ tuples from the Bridge V2 dataset [39]. For each tuple, we scale the number of generated action candidates using different sampling strategies and compute the Normalized Root Mean Squared Error (NRMSE) between the ground-truth action $a^*$ and each of the sampled actions $\{a_1, a_2, ..., a_m\}$.

We evaluate four sampling approaches: **Repeated sampling**: actions are repeatedly sampled from a robot policy $\pi(a \mid s,\ I)$ with a positive temperature. **Gaussian perturbation**: a small batch of actions is sampled from the policy $\pi(a \mid s,\ I)$, from which a Gaussian distribution is fit and used to draw all candidate actions. **Instruction rephrasing**: actions are sampled from the policy $\pi(a \mid s,\ I)$ conditioned on rephrased instructions $\{l_1, l_2, ..., l_k\}$ generated by a VLM. **Hybrid sampling**: instead of generating a single action candidate per rephrased instruction, we fan out and repeatedly sample multiple actions per rephrase. We also find that the relationship between action error and total inference FLOPs follows an exponentiated power law across these sampling methods. For power law fitting, we model the logarithm of action error $e$ as a function of the allocated inference compute.

The results in Figure 3 reveal two key findings: (1) instruction rephrasing consistently yields lower action error compared to vanilla repeated sampling and Gaussian perturbation; and (2) the hybrid approach combining instruction rephrasing with repeated

sampling achieves even greater diversity by exploring radically different actions rather than getting stuck in a local minimum.

# 4. Method

While prior works focus either on policy learning or on atomic-level action verification, our approach introduces a general hierarchical test-time verification and scaling framework (Section 4.1) that integrates *scalable verifier training* (Section 4.2) and *hierarchical instruction-action verification* (Section 4.3). Instead of treating the base model's output as final, we jointly select high-level language prompts and low-level action alignment through an optimized latency-aware inference pipeline.

## 4.1. Hierarchical Prompt-Action Optimization

We consider a sequential decision-making problem with observation space $\mathcal{O}$, action space $\mathcal{A}$, and natural-language instruction space $\mathcal{L}$. At timestep $t$, the robot receives an observation $o_t \in \mathcal{O}$ and a user instruction $l \in \mathcal{L}$. A chunk-based VLA policy $\pi$ produces an action chunk $a_t \sim \pi(a_t \mid o_t, l)$, where $a_t$ may correspond to multiple low-level control steps. Natural language permits many semantically equivalent rephrases, yet VLA policies are notoriously sensitive to phrasing. For a rephrased instruction $l'$, the induced action $a'_t \sim \pi(a'_t \mid o_t, l')$ may deviate significantly from the intended behavior, revealing a brittleness to linguistic drift. This motivates treating the instruction itself as a decision variable that can be optimized at test time.

**Language-level optimization.** Rather than committing to a single phrasing, we construct a set with $K$ number of rephrases:

$$\mathcal{L}_r(l') = \{l'_1, \ldots, l'_K\},$$

all expressing the same user intent. Each $l'_k$ conditions a different action distribution under the fixed base policy. To formalize the objective, we use a conceptual reward function $r(o_t, a, l)$ that measures how well an action $a$ fulfills the semantics of the original instruction $l$; this reward is *not* computed at test time but serves to define the ideal target behavior. We then aim to select the rephrase whose induced behavior best aligns with the original intent:

$$l^* = \arg\max_{l' \in \mathcal{L}_r} \mathbb{E}_{a \sim \pi(\cdot \mid o_t, l')} \big[ r(o_t, a, l) \big].$$

This reformulates VLA inference as an optimization problem in *language space*, not parameter space.

**Action-level optimization.** Given a selected rephrase $l^*$, sampling a single action from $\pi$ is unreliable due to bias and noise. We therefore draw $M$ candidate action chunks from the policy, conditioning on the current observation $o_t$ and the selected instruction $l^*$:

$$a'_j \sim \pi(\cdot \mid o_t, l^*), \qquad j = 1, \ldots, M,$$

We then select the candidate that maximizes semantic alignment with the instruction:

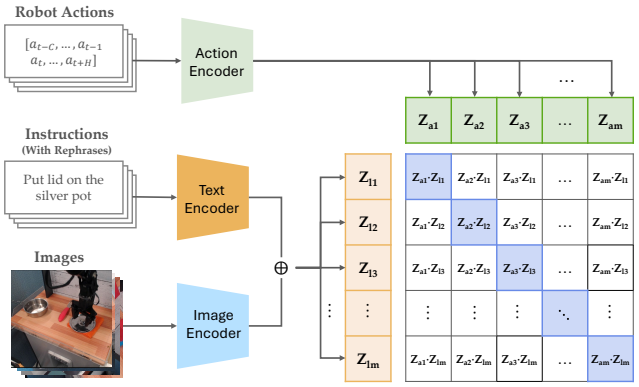


Figure 4. **Overview of CoVer Training Strategy.** CoVer learns a joint embedding space aligning visual observations, language instructions, and robot actions through contrastive pre-training. Image and text encoders extract task-relevant visual–linguistic features, which are fused into text-aware visual representations. An action encoder maps action sequences into the same embedding space, enabling cross-modal alignment between instructions and executed behaviors.

$$a_t^* = \arg\max_{j \in [M]} \mathcal{V}_\theta\big(o_t, h_t, l^*, a'_j\big),$$

where $\mathcal{V}_\theta$ estimates vision–language–action alignment, and $h_t \in \mathcal{A}^W$ denotes the recent action history (e.g., the past $C$ actions), providing temporal context to the verifier.

This view unifies language refinement and action verification: the system first searches for the rephrase whose induced action distribution aligns with the user intent, then verifies individual action candidates within that distribution. Overall, developing verifier $\mathcal{V}_\theta$ is essential. In the next section, we will describe how to develop a *robust* and *scalable* verifier from available robotics datasets.

## 4.2. Offline Verifier Training

The objective of verifier $\mathcal{V}_\theta$ is to assess the semantic alignment between visual observations, instruction language, and action sequences. A central challenge in training a VLA verifier is that robotic datasets contain only successful demonstrations, providing no direct supervision indicating when an action is semantically *misaligned* with an instruction. Constructing negative examples is non-trivial [42]: synthesizing incorrect actions often produces unrealistic motions, while manually annotating failures is prohibitively expensive. Contrastive learning [32, 37] offers a natural solution by treating other actions in the batch as implicit negatives, allowing the model to learn alignment structure without curated failure labels. Our training pipeline consists of two stages: (i) augmenting the instruction space with diverse rephrases and (ii) contrastive learning on the augmented dataset. The detailed algorithm is shown in Algorithm 1.

**Rephrase Augmentation.** To address the linguistic sensitivity of VLA policies, we expand each original instruction solely in language space, leaving observations and actions fixed. The training language augmentation is obtained from Open-X

**Algorithm 1** Verifier Training with Rephrase Augmentation

**Require:** Offline trajectories $\mathcal{D}=\{(o_t,h_t,l,a_t)\}_{t=1}^{T}$; batch size $B$; Augmented Instruction Set $\mathcal{I}$

Initialize augmented dataset $\mathcal{D}_{\text{aug}} \leftarrow \emptyset$

**Stage 1: Rephrase Augmentation**

**for** $(o_t,h_t,l,a_t) \in \mathcal{D}$ **do**

  **for** $l_n^{oxe}$ in $\mathcal{I}(l)$ **do**

    $\mathcal{D}_{\text{aug}} \leftarrow \mathcal{D}_{\text{aug}} \cup \{(o_t,h_t,l_n^{oxe},a_t)\}$

**Stage 2: Verifier Training**

Initialize parameters $\theta$

**while** not converged **do**

  Sample minibatch $\{(o_i,h_i,l_i,a_i)\}_{i=1}^{B} \sim \mathcal{D}_{\text{aug}}$

  **for** $i=1..B$ **do**

    $\mathbf{F}_i = \mathbf{F}_{\text{combined}}(o_i,l_i)$

    $\mathbf{A}_i = \mathbf{A}(h_i,a_i)$

    Normalize: $\mathbf{f}_i = \mathbf{F}_i/\|\mathbf{F}_i\|_2, \quad \mathbf{a}_i = \mathbf{A}_i/\|\mathbf{A}_i\|_2$

    Compute pairwise similarities $s_{i,j} = \langle \mathbf{f}_i, \mathbf{a}_j \rangle$

    $\mathcal{L}_i^{f\to a} = -\log \frac{\exp(s_{i,i})}{\sum_{j=1}^{B}\exp(s_{i,j})}$

    $\mathcal{L}_i^{a\to f} = -\log \frac{\exp(s_{i,i})}{\sum_{j=1}^{B}\exp(s_{j,i})}$

  $\mathcal{L}_{\text{InfoNCE}} = \frac{1}{2B}\sum_{i=1}^{B}(\mathcal{L}_i^{f\to a} + \mathcal{L}_i^{a\to f})$

  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{InfoNCE}}$

Embodiment [6] datasets $\mathcal{I}$, where each original task instruction set $\mathcal{I}(l)$ corresponds to $N$ rephrases. Each selected rephrase $l_n^{oxe}$ from $\mathcal{I}(l)$ is then paired with the same observation $o_t$, and ground-truth action sequence that consists of short-term action history $h_t$ and future action chunk $a_t$ to form additional training tuples. Rephrases enable the verifier to encounter multiple linguistic realizations of the same underlying intent. This procedure enlarges the effective language coverage of the dataset without altering the action distribution, and equips the verifier with the ability to distinguish true semantic equivalence from phrasing-induced discrepancies that often mislead the base VLA policy. Though the same rephrase augmentation technique has been developed for policy learning [9, 10], we demonstrate that using the same data budget to train a verifier would be more effective than directly augmenting the policy training dataset.

**Verifier Training and Architecture.** The verifier aims to estimate the alignment between visual–textual and action representations. Visual inputs and language tokens are encoded with pre-trained SigLIP2 encoders [37], then fused via text-aware visual attention to obtain instruction-relevant features. Vision and text encoders are frozen during verifier training to preserve the web-scale knowledge [16]. The resulting fused representation $\mathbf{F}_{\text{combined}}$ captures visual–language context. The action sequence, which contains short-term history and future chunks, is processed by a transformer encoder to better capture the temporal features of low-level behaviors [26]. The fused vision–language representation $\mathbf{F}_{\text{combined}}$ and the action embedding $\mathbf{A}$ are then $\ell_2$-normalized to get $\mathbf{f}$ and $\mathbf{a}$ respectively. Their similarity defines the alignment score: $s(\mathbf{f},\mathbf{a}) = \langle \mathbf{f},\mathbf{a} \rangle$. Given a minibatch of $B$ tuples $\{(o_i,h_i,l_i,a_i)\}_{i=1}^{B}$, the verifier is trained with bi-directional InfoNCE [30] objective. This symmetrical formulation aligns vision–language embeddings $\mathbf{f}$ with action embeddings $\mathbf{a}$ in both directions. By treating all other pairs in the batch as *implicit negatives*, it leverages the diversity of each minibatch to learn robust fine-grained correspondences without requiring explicit failure labels or hand-crafted counterexamples. Such in-batch contrastive structure enables the verifier to discover meaningful distinctions between semantically aligned and misaligned behaviors, leading to more stable and cycle-consistent vision–language–action grounding during test-time verification. The verifier structure is shown in Figure 4. Given a robust verifier that can score the alignment between intentions and actions, we develop a general verification framework that can adapt to any VLA policies without additional training.

### 4.3. Test-time Verification

In Section 4.2, we explored the advantages of contrastive training for vision-language-action alignment, which enables zero-shot verification for both instruction and actions. Such bidirectional features make the verification process more flexible. In this section, we propose CoVer-VLA, a test-time verification framework that is robust to language-induced action drift, while adding only minimal latency from proposal generation and verification. CoVer-VLA casts inference as a hierarchical verification problem as shown in Figure 5. The system first evaluates on the language level, selecting the instruction whose induced action distribution is most semantically reliable, and then selects the optimal action chunk conditioned on that instruction. This hierarchical structure enables the robot to update its active language prompt online and to filter action proposals using a learned alignment score, improving robustness without altering the underlying VLA policy. To support this procedure, we first introduce boot-time rephrase generation and caching that significantly boosts runtime efficiency by bringing scene reasoning offline. We follow with the details on batched action proposals that enable efficient search over both languages and actions. The resulting pipeline preserves robustness without compromising real-time control, and the full procedure is summarized in Algorithm 2.

**Boot-time rephrase generation and caching.** To efficiently handle linguistic variability, we expand each free-form instruction $l'$ into $K$ rephrases using an off-the-shelf VLM. The VLM takes the initial scene image $o_0$ and user instruction $l'$ as input. It generates both scene-level reasoning and rephrased command variants $\{l'_k\}_{k=1}^{K}$. Leveraging the VLM's reasoning capabilities incorporates web-scale knowledge into the rephrase generation process. Running the VLM on-the-fly, however, is computationally expensive and can introduce undesirable latency or motion discontinuities during robot control. Given that user intent is typically consistent throughout an episode, generating new rephrases mid-rollout offers limited benefits. Instead, we perform rephrase generation and embedding computation entirely at boot time. By caching rephrase embeddings before

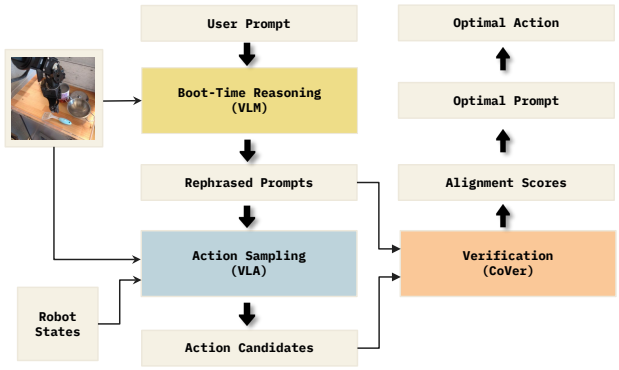


Figure 5. **Overview of Test-Time Verification Pipeline.** At deployment, the system performs hierarchical optimization over language and action spaces. Given a user prompt and the initial observation, a VLM first reasons over the scene and generates a set of rephrased prompts at *boot time*. For each rephrase, a VLA samples action candidates conditioned on the corresponding instruction. The trained CoVer verifier then scores all instruction–action pairs and selects the optimal prompt and action for execution.

**Algorithm 2** Hierarchical Test-time Verification

**Input:** base policy $\pi$, verifier ensemble $\mathcal{V}_\theta$, user instruction $l'$, num of rephrases $K$, num of action samples $M$
**Boot-time:** generate rephrases $\{l'_k\}_{k=1}^K \leftarrow \text{VLM}(o_0, l')$; cache embeddings
**while** episode not finished **do**
  **1. Sample action proposals**
  **for** $k=1$ to $K$ **do**
    **for** $j=1$ to $M$ **do**
      $a'_{k,j} \sim \pi(\cdot \mid o_t, l'_k)$
  **2. Score proposals**
  $s_{k,j} = \mathcal{V}_\theta(o_t, h_t, l', a'_{k,j})$
  **3. Select rephrase (language-level)**
  $S_k = \frac{1}{M}\sum_{j=1}^M s_{k,j} \qquad k^* = \text{argmax}_k S_k$
  **4. Select action (action-level)**
  $j^* = \text{argmax}_j s_{k^*,j}$
  Execute $a'_{k^*,j^*}$ and update $(o_{t+\Delta}, h_{t+\Delta})$

execution, we shift the heaviest computations off the critical path and ensure that retrieving rephrase features at inference time incurs negligible overhead. This allows the controller to evaluate paraphrastic variants efficiently at test time, enabling robust test-time optimization without compromising control smoothness. Detailed implementations of boot-time reasoning can be found in Appendix 8.9, and VLM prompts in Appendix 8.10.

**Inference with batched action proposals.** With rephrases cached and a verifier in place, we perform chunk-level optimization by jointly searching over rephrased instructions and candidate action chunks. Let $\{l'_k\}_{k=1}^K$ denote the $K$ rephrases generated at boot time, with $l'_1 = l'$. At each chunk boundary, the base VLA policy induces a distribution over action chunks, $a \sim \pi(\cdot \mid o_t, l'_k)$, from which we sample $M$ candidates for each rephrase. This yields $K \times M$ proposals:

$$a'_{k,j} \sim \pi(\cdot \mid o_t, l'_k), \qquad k = 1,...,K,\ j = 1,...,M.$$

Each proposal is then evaluated by the verifier ensemble with respect to the *user instruction* $l'$,

$$s_{k,j} = \mathcal{V}_\theta(o_t, h_t, l', a'_{k,j}),$$

producing a semantic alignment score for every (rephrase, action) pair. To determine which rephrase induces the most reliable action distribution, we take the average scores across all $M$ actions from the same language:

$$S_k = \frac{1}{M}\sum_{j=1}^{M} s_{k,j}, \qquad k^* = \text{argmax}_k S_k.$$

The chosen rephrase $l'_{k^*}$ becomes the active language for this chunk. Within the selected rephrase, the controller chooses the highest-scoring action candidate:

$$j^* = \text{argmax}_j s_{k^*,j}.$$

The selected action chunk $a'_{k^*,j^*}$ is executed, and the state $(o_{t+\Delta}, h_{t+\Delta})$ is updated accordingly. This procedure repeats at each chunk boundary, forming a closed-loop optimization that continually adapts both the instruction and the executed action.

## 5. Experiments

### 5.1. Verifier Scaling Results

In this section, we investigate the scaling behavior of the CoVer verifier. We conduct thorough studies to explore the impact of five key dimensions: model size, dataset size, batch size, training compute, and ensemble size. Detailed specifications regarding architecture and compute usage are provided in the Appendix.

We first evaluate how scaling synthetic instructions and model parameters affects verifier performance. As shown in Figure 4, CoVer exhibits consistent scaling trends: every increase in dataset size (from $8\times$ to $64\times$) or in model capacity (from 250M to 1B parameters) leads to steady improvements in top-1 retrieval accuracy. This provides strong empirical evidence that our contrastive approach effectively capitalizes on scaling.

We also investigate the effects of scaling batch size and training epochs. Because our verifier relies on contrastive learning, the number of in-batch negative samples is critical for learning robust decision boundaries. We find that larger batch sizes (scaling from 2,048 to 8,192) provide a richer set of negative examples, thereby facilitating better convergence. Similarly, extending training epochs exposes the model to more diverse negative samples, leading to improved results.

Finally, we explore test-time ensembling as a scaling dimension. Specifically, we train multiple verifiers with

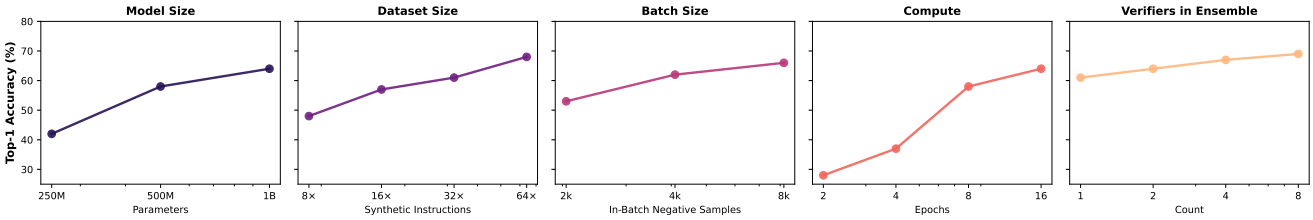


Figure 6. **Verifier Scaling Results.** We show that our architecture scales gracefully with additional compute and data. The top-1 action-retrieval accuracy consistently improves as we scale the number of synthetic instructions, model parameters, negative samples, training compute, and the number of verifiers in the ensemble. This result strongly indicates that our approach benefits from scaling, which we exploit for training CoVer.

identical architectures and data budgets, differing only in their random seeds. During inference, we average the image, text, and action embeddings across these verifiers before computing the cosine similarity between modalities. We find that action retrieval accuracy consistently improves as the ensemble size increases (from 1 to 8). These gains stem from variance reduction, as the ensemble averages out individual model biases.

## 5.2. Implementation Details

Our final CoVer verifier is a 1B-parameter model trained with a batch size of 32,768 on the augmented Bridge V2 dataset [39] containing 16× synthetic instructions. Training was conducted for a total of 2k steps using 8 NVIDIA H200 GPUs. For deployment, we utilize an ensemble of 3 verifiers to balance robustness and computational overhead.

## 5.3. Evaluation Setup

We evaluate CoVer-VLA across both simulated and real-world settings, focusing on robustness to linguistic variation and generalization on out-of-distribution environments (Appendix 8.1). Our primary benchmark is the SIMPLER benchmark [23], which includes four in-distribution (ID) manipulation tasks and three OOD variants containing distractor objects and clutter [9]. We evaluate on four representative tasks from the SIMPLER environment and adopt three challenging OOD tasks from Interleave-VLA [9], includes "Redbull on Plate", "Zucchini on Towel", and "Tennis in basket". The OOD environments contain multiple objects in the scene, where the VLA cannot rely solely on visual inputs and must also reason over the object information in the instructions. For real-world experiments, we use the WidowX robot to evaluate two tasks "pepto bismol on plate" and "redbull on plate". We use $\pi_0$ as the base model for tasks in BridgeV2. To assess how our approach performs with a stronger base policy, we also evaluate using $\pi_{0.5}$ and CoVer on the PolaRiS benchmark [17]. All evaluations are conducted under challenging red-teaming instructions generated by ERT [18] (Appendix 8.8). Our framework samples 8 rephrased instructions and generates 5 action candidates per rephrase.

**Baselines and Ablations.** We compare CoVer-VLA against five variants built on the same $\pi_0$ backbone to disentangle the effects of training-time augmentation and test-time verification (Appendix 8.2). (1) $\pi_0$ denotes the generalist robot policy fine-tuned on BridgeV2 without instruction augmentation or verification. (2) $\pi_0$ **(rephrase)** [10] represents $\pi_0$ finetuned on instruction-augmented datasets. (3) **RoboMonkey** [22] applies a 7B-scale verifier with action resampling for test-time scaling, serving as the strongest prior method without hierarchical reasoning. (4)$\pi_0$**+ CoVer** introduces our verifier-based inference that jointly optimizes over rephrases and action chunks at test time. (5) $\pi_0$**+Rand. Reph.** uses a single random rephrase without verification to isolate the role of language selection. (6) $\pi_0$ **(rephrase)+ CoVer** combines both training-time augmentation and our hierarchical test-time verifier to examine their complementarity. Together, these baselines allow us to examine the effectiveness of (i) training-time instruction augmentation, (ii) test-time verification of instructions and actions, and (iii) verifier-guided hierarchical optimization. This allows us to systematically assess CoVer 's robustness and ability to generalize across tasks.

## 5.4. Simulation Evaluation Results

Figure 7 summarizes performance across four ID tasks and three OOD tasks under red-teaming instructions. Detailed numerical values are described in the Appendix 8.3. Due to training distribution shift, Robomonkey fails to select optimal actions given challenging instructions. For all the other ablations, we observe different levels of performance gain over the base robot policy $\pi_0$. We highlight three key findings below:

**(1) Training-time augmentation alone provides modest performance gain.** We show that fine-tuning $\pi_0$ on augmented instruction sets can indeed improve robustness to challenging rephrases. However, this approach yields only minimal gains on in-distribution environments (41.5→ 44) and provides modest improvements on OOD tasks.

**(2) Random rephrases can improve performance on some tasks but lack consistency without language-level verification.** Using a randomly generated VLM rephrase slightly improves ID performance over the base policy $\pi_0$ (41.5 → 42.3), confirming that rephrasing can enhance policy performance in some cases. However, OOD performance declines (29.7 → 28.7), and the variance across tasks is substantial. For example, the model achieves a 78% success rate on *Eggplant in Basket* but only 1% on *Redbull on Plate*. This reveals a key insight: while certain

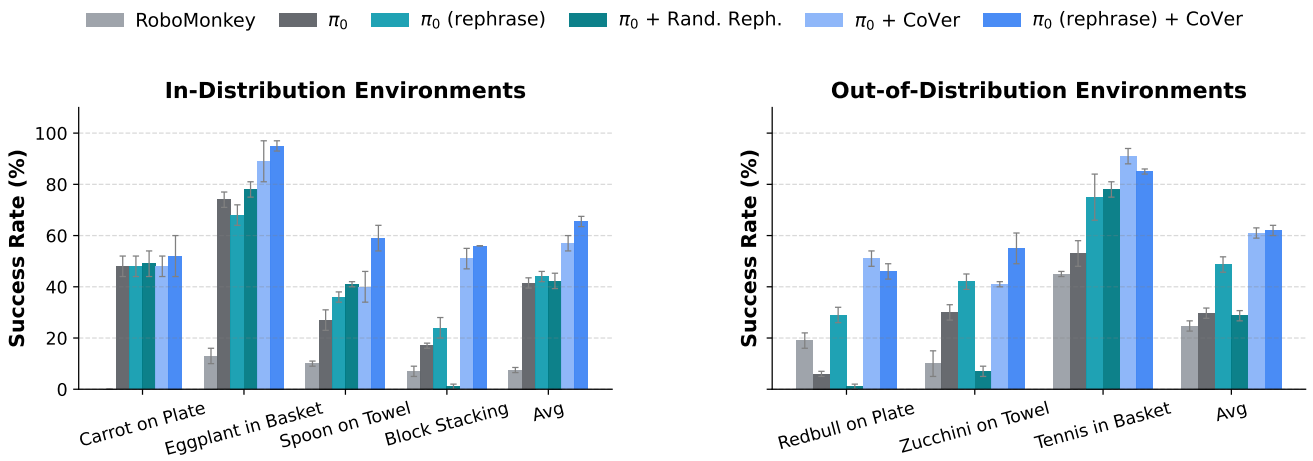


Figure 7. **SIMPLER Evaluation Results.** We demonstrate that scaling test-time verification with CoVer significantly enhances the robustness of VLAs across diverse manipulation tasks. Compared to scaling policy pre-training on the same data, our verification-based approach achieves a 22% improvement on in-distribution tasks and a 13% improvement on OOD tasks.

| | Models | Task Progress (%) | Success Rate (%) |
|---|---|---|---|
| PanClean | $\pi_{0.5}$ | $48.4 \pm 1.9$ | $10.7 \pm 0.9$ |
| | $\pi_{0.5}$ + CoVer | $70.4 \pm 4.0$ | $33.3 \pm 6.6$ |
| BlockStack | $\pi_{0.5}$ | $33.1 \pm 1.3$ | $0.0 \pm 0.0$ |
| | $\pi_{0.5}$ + CoVer | $44.3 \pm 2.5$ | $0.7 \pm 0.9$ |
| FoodBussing | $\pi_{0.5}$ | $38.3 \pm 2.4$ | $0.7 \pm 0.9$ |
| | $\pi_{0.5}$ + CoVer | $47.0 \pm 4.1$ | $5.3 \pm 1.9$ |
| Average | $\pi_{0.5}$ | $40.0 \pm 6.4$ | $3.8 \pm 4.9$ |
| | $\pi_{0.5}$ + CoVer | **53.9** $\pm$ **11.7** (**+13.9**↑) | **13.1** $\pm$ **14.1** (**+9.3**↑) |

Table 1. **PolaRiS benchmark evaluation.** Mean task progress and success rate ($\pm$ standard deviation) across 50 episodes and 3 seeds on three PolaRiS environments. $\pi_{0.5}$ serves as the stronger base VLA policy. $\pi_{0.5}$ **+ CoVer** consistently improves performance across all tasks, achieving a 13.9% gain in task progress and a 9.3% increase in success rate on average, demonstrating that hierarchical verification complements stronger base models.

rephrased instructions can be beneficial, others may catastrophically mislead the policy. These results underscore the potential of VLM-generated rephrasings, but also expose their inconsistency.

**(3) CoVer-VLA substantially enhances generalization and complements policy learning.** Pairing CoVer with $\pi_0$ significantly enhances robustness, yielding a 16% improvement on in-distribution tasks and a 31% gain in OOD environments. Notably, we find that scaling verification ($\pi_0$ + CoVer) outperforms scaling policy learning ($\pi_0$ fine-tuned with augmented instructions), achieving 15% gains on ID tasks and 12% on OOD, while requiring substantially less compute, as illustrated in Figure 2. Interestingly, our approach is complementary to scaling policy learning. Combining $\pi_0$ (rephrase) and CoVer achieves the strongest overall performance: 65.5% on ID tasks and 62.0% on OOD tasks. We further evaluate our method with a stronger base model, $\pi_{0.5}$, on the PolaRiS benchmark [17]. Pairing $\pi_{0.5}$ with CoVer leads to a 14% improvement in task progress and a 9% gain in success rate. By jointly selecting the semantically aligned instruction and verifying action chunks, our method reliably recovers correct behavior even under heavily perturbed instructions and in challenging OOD environments.

### 5.5. Real-World Evaluation Results

We further evaluate CoVer-VLA in two real-world manipulation tasks as shown in Figure 9. CoVer-VLA substantially outperforms the baselines, improving the success rate by 30% and 60%, respectively. CoVer-VLA consistently shows the correct intention to accomplish the task, whereas the other baselines often fail to identify the correct object. We observe that the base $\pi_0$ model often failed to initiate motion under challenging scenes and instructions, resulting in 0% success. Overall, these results demonstrate that scaling test-time verification with CoVer provides an effective and scalable pathway toward building a robust robotics foundation model.

### 5.6. Latency Analysis and Optimizations

While our approach introduces additional computational overhead from action sampling and verification, we mitigate these costs through several key optimizations. Concretely, we decouple the image-text encoder and action encoder within our verifier architecture. This design enables the image-text embedding to be computed in parallel with the forward pass of the base robot policy. As a result, the end-to-end latency of our pipeline consists only of batched inference with $\pi_0$ (or $\pi_{0.5}$) and a lightweight action encoder from CoVer. As shown in Table 2, the action encoder consistently adds only $\sim$8 ms even at larger batch sizes. In addition, repeated sampling can exploit KV cache optimizations and batch processing to achieve higher throughput than greedy decoding, allowing CoVer-VLA to sample and verify 16 candidate actions in approximately 453 ms ($\sim$2.2 Hz). We also avoid online rephrase generation by shifting reasoning to boot time. Specifically, we precompute and cache a set of diverse rephrased instructions before deployment. This eliminates redundant runtime calls to the VLM, thereby minimizing inference-time latency. Our full latency and throughput analysis can be found in Appendix 8.7.

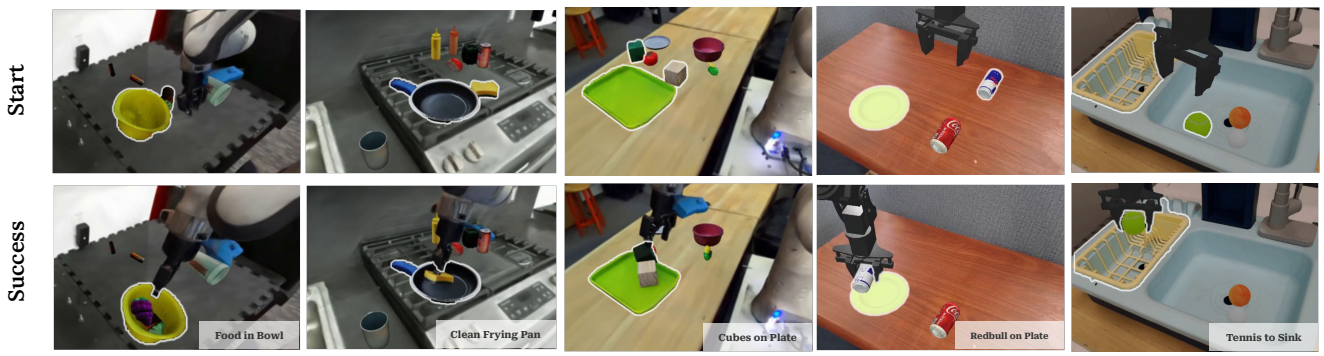


Figure 8. Example tasks across DROID [19] and Bridge V2 [39] environments.

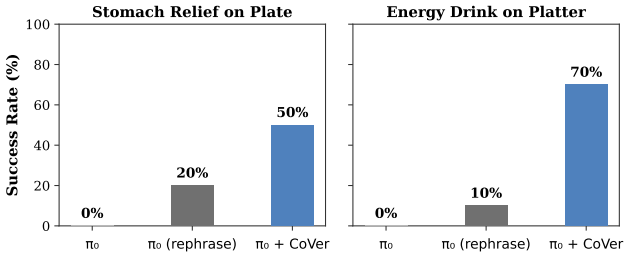


Figure 9. **Real-World Evaluation Results.** $\pi_0$ + CoVer outperforms the baseline policy $\pi_0$ (rephrase), achieving on average 45% performance gain in terms of task success rate compared to the baseline policy.

## 6. Conclusion

In this paper, we present CoVer, a novel contrastive-based verifier and hierarchical test-time scaling framework that bridges the "intention–action gap" for generalist robot policies. CoVer achieves substantial performance improvements across both simulated and real-world settings, particularly under out-of-distribution conditions. Our findings demonstrate that allocating compute to reasoning and verification at deployment can be more effective than scaling policy training alone, providing a promising direction for robust policy deployment in the real world. While our study focuses on applying the verifier for test-time scaling, the same design and principle can extend beyond inference optimizations such as post-training with reinforcement learning or run-time monitoring. Future work could also explore more efficient architectures for both the base policy and verifier to further reduce latency and enable broader use of test-time scaling in real-world robotic settings.

## 7. Acknowledgments

We thank the members of the Stanford Autonomous Systems Lab, Scaling Intelligence Lab, and IRIS Lab for their constructive feedback and informative discussions. We gratefully acknowledge the support of DARPA; NASA ULI; Google DeepMind; Google Research; Google Cloud; SNSF; IBM and Felicis.

| Batch Size | $\pi_{0.5}$ (ms) | CoVer (ms) | $\pi_{0.5}$ + CoVer (ms) |
|---|---|---|---|
| 1 | 56 | 7 | 63 |
| 2 | 84 | 7 | 91 |
| 4 | 138 | 8 | 146 |
| 8 | 243 | 8 | 251 |
| 16 | 445 | 8 | 453 |
| 32 | 865 | 8 | 873 |

Table 2. Latency (milliseconds) across batch sizes running on RTX-5090 GPU. Since the image-text encoder can run in parallel with the forward pass of $\pi_{0.5}$, the end-to-end latency of our pipeline only consists of batched inference with $\pi_{0.5}$ and the lightweight CoVer action encoder

## References

[1] Christopher Agia, Rohan Sinha, Jingyun Yang, Zi-ang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *8th Annual Conference on Robot Learning*, 2024. 3

[2] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024. 1

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_0$: A Vision-Language-Action Flow Model for General Robot Control. *arXiv preprint arXiv:2410.24164*, 2024. 2, 13

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin

Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. 1

[5] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. *arXiv preprint arXiv:2407.21787*, 2024. 3

[6] Open X.-Embodiment Collaboration, Abhishek Padalkar, Acorn Pooley, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Animesh Garg, Anthony Brohan, Antonin Raffin, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Brian Ichter, Cewu Lu, Charles Xu, Chelsea Finn, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Chuer Pan, Chuyuan Fu, Coline Devin, Danny Driess, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Federico Ceola, Fei Xia, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Giulio Schiavi, Gregory Kahn, Hao Su, Hao-Shu Fang, Haochen Shi, Heni Ben Amor, Henrik I. Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jiajun Wu, Jialin Wu, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jitendra Malik, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Joseph J. Lim, João Silvério, Junhyek Han, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Zhang, Krishan Rana, Krishnan Srinivasan, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Lionel Ott, Lisa Lee, Masayoshi Tomizuka, Max Spero, Maximilian Du, Michael Ahn, Mingtong Zhang, Mingyu Ding, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J. Joshi, Niko Suenderhauf, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Pannag R. Sanketi, Paul Wohlhart, Peng Xu, Pierre Sermanet, Priya Sundaresan, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Sudeep Dasari, Suneel Belkhale, Takayuki Osa, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xuanlin Li, Yao Lu, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zhuo Xu, and Zichen Jeff Cui. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. *arXiv preprint arXiv:2310.08864*, 2023. 5, 13

[7] Perry Dong, Suvir Mirchandani, Dorsa Sadigh, and Chelsea Finn. What Matters for Batch Online Reinforcement Learning in Robotics? *arXiv preprint*, 2025. 3

[8] Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z. Ren, Homer Walke, Quan Vuong, Lucy Xiaoyang Shi, and Sergey Levine. Knowledge Insulating Vision-Language-Action Models: Train Fast, Run Fast, Generalize Better. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 3

[9] Cunxin Fan, Xiaosong Jia, Yihang Sun, Yixiao Wang, Jianglan Wei, Ziyang Gong, Xiangyu Zhao, Masayoshi Tomizuka, Xue Yang, Junchi Yan, et al. Interleave-vla: Enhancing robot manipulation with interleaved image-text instructions. *arXiv preprint arXiv:2505.02152*, 2025. 5, 7

[10] Irving Fang, Juexiao Zhang, Shengbang Tong, and Chen Feng. From intention to execution: Probing the generalization boundaries of vision-language-action models. *arXiv preprint arXiv:2506.09930*, 2025. 1, 2, 5, 7, 13, 15, 16

[11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1

[12] Shresth Grover, Akshay Gopalkrishnan, Bo Ai, Henrik I. Christensen, Hao Su, and Xuanlin Li. Enhancing Generalization in Vision-Language-Action Models by Preserving Pretrained Representations. *arXiv preprint arXiv:2509.11417*, 2025. 3

[13] Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *arXiv preprint arXiv:2506.09937*, 2025. 3

[14] Asher J. Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning vlms into vlas without catastrophic forgetting, 2025. 2

[15] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies. *arXiv preprint arXiv:2304.10573*, 2023. 3

[16] Huang Huang, Fangchen Liu, Letian Fu, Tingfan Wu, Mustafa Mukadam, Jitendra Malik, Ken Goldberg, and Pieter Abbeel. Otter: A vision-language-action model with text-aware visual feature extraction. *arXiv preprint arXiv:2503.03734*, 2025. 5

[17] Arhan Jain, Mingtong Zhang, Kanav Arora, William Chen, Marcel Torne, Muhammad Zubair Irshad, Sergey Zakharov, Yue Wang, Sergey Levine, Chelsea Finn, et al. Polaris: Scalable real-to-sim evaluations for generalist robot policies. *arXiv preprint arXiv:2512.16881*, 2025. 7, 8

[18] Sathwik Karnik, Zhang-Wei Hong, Nishant Abhangi, Yen-Chen Lin, Tsun-Hsuan Wang, Christophe Dupuy, Rahul Gupta, and Pulkit Agrawal. Embodied red teaming for auditing robotic foundation models, 2025. 1, 2, 7, 15, 16

[19] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R. Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul

Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J. Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. *arXiv preprint arXiv:2403.12945*, 2024. 9

[20] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. 1, 2, 13

[21] Taeyoung Kim, Jimin Lee, Myungkyu Koo, Dongyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Contrastive Representation Regularization for Vision-Language-Action Models. *arXiv preprint*, 2025. 3

[22] Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. *arXiv preprint arXiv:2506.17811*, 2025. 2, 3, 7, 13, 15

[23] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024. 7

[24] Xinghang Li, Peiyan Li, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Tao Kong, Hanbo Zhang, and Huaping Liu. Towards Generalist Robot Policies: What Matters in Building Vision-Language-Action Models. *arXiv preprint arXiv:2412.14058*, 2024. 3

[25] Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can rl bring to vla generalization? an empirical study. *arXiv preprint arXiv:2505.19789*, 2025. 2

[26] Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional Decoding: Improving Action Chunking via Guided Test-Time Sampling. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. 3, 5

[27] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. S1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025. 3

[28] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *Conference on Robot Learning (CoRL)*, 2024. 2, 3

[29] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. *arXiv preprint arXiv:2503.14734*, 2025. 2

[30] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 5

[31] Han Qi, Haocheng Yin, Aris Zhu, Yilun Du, and Heng Yang. Strengthening Generative Robot Policies through Predictive World Modeling. *arXiv preprint arXiv:2502.00622*, 2025. 3

[32] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 2, 4

[33] Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Kumar Guha, E. Kelly Buchanan, Mayee F. Chen, Neel Guha, Christopher Re, and Azalia Mirhoseini. Archon: An Architecture Search Framework for Inference-Time Techniques. 2024. 3

[34] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. *arXiv preprint arXiv:2506.01844*, 2025. 2

[35] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314*, 2024. 3

[36] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann, Arunkumar Byravan, Serkan Cabi, Ken Caluwaerts, Federico Casarini, Oscar Chang, Jose Enrique Chen, Xi Chen, Hao-Tien Lewis Chiang, Krzysztof Choromanski, David D'Ambrosio, Sudeep Dasari, Todor Davchev, Coline Devin, Norman Di Palo, Tianli Ding, Adil Dostmohamed, Danny Driess, Yilun Du, Debidatta Dwibedi, Michael Elabd, Claudio Fantacci, Cody Fong, Erik Frey, Chuyuan Fu, Marissa Giustina, Keerthana Gopalakrishnan, Laura Graesser, Leonard Hasenclever, Nicolas Heess, Brandon Hernaez, Alexander Herzog, R. Alex Hofer, Jan Humplik, Atil Iscen, Mithun George Jacob, Deepali Jain, Ryan Julian, Dmitry Kalashnikov, M. Emre Karagozler, Stefani Karp, Chase Kew, Jerad Kirkland, Sean Kirmani, Yuheng Kuang, Thomas Lampe, Antoine Laurens, Isabel Leal, Alex X. Lee, Tsang-Wei Edward Lee, Jacky Liang, Yixin Lin, Sharath Maddineni, Anirudha Majumdar, Assaf Hurwitz Michaely, Robert Moreno, Michael Neunert, Francesco Nori, Carolina Parada, Emilio Parisotto, Peter Pastor, Acorn Pooley, Kanishka Rao, Krista Reymann, Dorsa Sadigh, Stefano Saliceti, Pannag Sanketi, Pierre Sermanet, Dhruv Shah, Mohit Sharma, Kathryn Shea, Charles Shu, Vikas Sindhwani, Sumeet Singh, Radu Soricut, Jost Tobias Springenberg,

Rachel Sterneck, Razvan Surdulescu, Jie Tan, Jonathan Tompson, Vincent Vanhoucke, Jake Varley, Grace Vesom, Giulia Vezzani, Oriol Vinyals, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Fei Xia, Ted Xiao, Annie Xie, Jinyu Xie, Peng Xu, Sichun Xu, Ying Xu, Zhuo Xu, Yuxiang Yang, Rui Yao, Sergey Yaroshenko, Wenhao Yu, Wentao Yuan, Jingwei Zhang, Tingnan Zhang, Allan Zhou, and Yuxiang Zhou. Gemini Robotics: Bringing AI into the Physical World. *arXiv preprint arXiv:2503.20020*, 2025. 2

[37] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025. 2, 4, 5

[38] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering Your Diffusion Policy with Latent Space Reinforcement Learning. *arXiv preprint arXiv:2506.15799*, 2025. 3

[39] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023. 3, 7, 9, 13

[40] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully Test-Time Adaptation by Entropy Minimization. In *International Conference on Learning Representations*, 2020. 3

[41] Yilin Wu, Ran Tian, Gokul Swamy, and Andrea Bajcsy. From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment. In *Robotics: Science and Systems (RSS)*, 2025. 3

[42] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? uncertainty-aware runtime failure detection for imitation learning policies. *arXiv preprint arXiv:2503.08558*, 2025. 1, 3, 4

[43] Shuai Yang, Hao Li, Yilun Chen, Bin Wang, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. InstructVLA: Vision-Language-Action Instruction Tuning from Understanding to Manipulation. *arXiv preprint*, 2025. 3

[44] Xiangcheng Zhang, Haowei Lin, Haotian Ye, James Zou, Jianzhu Ma, Yitao Liang, and Yilun Du. Inference-time Scaling of Diffusion Models through Classical Search. *arXiv preprint arXiv:2505.23614*, 2025. 3

[45] Yang Zhang, Chenwei Wang, Ouyang Lu, Yuan Zhao, Yunfei Ge, Zhenglong Sun, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. Align-Then-stEer: Adapting the Vision-Language Action Models through Unified Latent Guidance. *arXiv preprint*, 2025. 3

# 8. Appendix

## 8.1. Evaluation Tasks

As described in Section 5.3, we evaluate our method on 7 tasks from the SIMPLER environments, 3 tasks from the PolaRiS benchmark, and 2 real-world tasks using the WidowX robot. Representative task executions for the benchmarks and real-world rollouts are shown in Figure 10. The Out-Of-Distribution (OOD) environments contains multiple distractors and several novel objects not present in BridgeV2 [39]. Real-world evaluations introduce additional distribution shifts due to unavoidable differences in camera placement, workspace, lighting, and background. We provide task-specific details below.

### 8.1.1. Bridge V2 Task Descriptions

- **Put Redbull Can on Plate (SIMPLER).** This task highlights a frequent language–vision ambiguity: the word "red" appears in "Redbull," which often causes VLA policies to grasp the *red* Coca-Cola can instead of the correct *blue* Redbull can. The robot must therefore ground the instruction precisely and place the correct can on the plate.
- **Put the Zucchini on the Towel (SIMPLER).** This environment tests fine-grained object discrimination in OOD scenes. The robot must identify the zucchini among multiple novel objects, including a carrot. Because both are vegetables, rephrases (e.g., replacing "zucchini" with "vegetable") become ambiguous, making this task a direct test of whether instruction rephrasing helps when objects share semantic categories.
- **Put Tennis Ball into Yellow Basket (SIMPLER).** The sink contains a tennis ball, a ping-pong ball, and an orange. The robot must correctly identify the tennis ball in this cluttered scene and place it inside the yellow basket while ignoring the other spherical distractors.
- **Put Redbull Can on Plate (Real World).** The setup contains multiple cans with textures and color variations not present in the simulation. The robot must select the correct can and place it onto a plate despite inherent camera and lighting variation.
- **Put Pepto Bismol on Plate (Real World).** This task introduces a completely unseen object, a pepto bismol bottle and an advil bottle, whose appearance differs substantially from all training objects. The robot must ground the novel object and place it onto a plate while ignoring other distractors.

### 8.1.2. PolaRiS Task Descriptions

PolaRis benchmark is developed based on DROID dataset, which contains more challenging and realistic tasks. Evaluation demonstrated on PolaRis further proved the benefits of CoVer. The successful task executions are shown in Figure 10.

- **Place and stack the blocks on top of the green tray (PolaRis).** The table contains several distractors for both the target objects and location. The scene contains a corn, a tomato, a wooden block, a green block, a blue plate, a red bowl, and a green tray. The policy needs to accurately identify the wooden and green block, and put both object on the tray.
- **Put all the foods in the bowl (PolaRis).** The scene contains two batteries, one ice cream, one grape, one cup, and one bowl. The model needs to identify the food catagery (ice cream and grape), and put them sequentially into the target container.
- **Use the yellow sponge to scrub the blue handle frying pan (PolaRis).** The scene represents a standard kitchen setting with cluttered objects on the stove, including two condiment bottles, a latte cup, a sushi, a coke, a sponge, and a frying pan. The task is to pick up the yellow sponge and move it to the frying pan.

## 8.2. Baselines

To make the baseline design and corresponding results explicit, we summarize each evaluated setting below:

- $\pi_0$**.** The base $\pi_0$ checkpoint [3] fine-tuned on BridgeV2 [39]. This represents a vanilla generalist robot policy with *no* instruction augmentation and *no* test-time verification.
- $\pi_0$ **(rephrase) [10].** Incorporates training-time instruction augmentation using the OpenX-Embodimenet dataset [6].
- **RoboMonkey [22].** A test-time scaling framework that uses a 7B VLM-based verifier and an action resampling strategy. For fairness, we changed RoboMonkey's base policy from OpenVLA [20] to $\pi_0$. This baseline reflects the strongest existing test-time verification method for VLAs.
- $\pi_0$ **+ CoVer** Our verifier-driven test-time pipeline applied directly to $\pi_0$. This isolates the contribution of CoVer 's hierarchical optimization—jointly selecting the most suitable rephrase and the best action chunk—without any training-time augmentation. We evaluate using 8 sampled rephrases and 5 repeated action samples per step. The generated 8 rephrases for each tasks are summarized in Table 8.
- $\pi_0$ **+ Rand. Reph.** Uses a single random VLM-generated rephrase, fixed for the entire rollout and without any verification. The selected rephrases for each tasks are presented in Table 8. This isolates the effect of CoVer's test-time language optimization: if rephrase choice mattered little, random rephrases would perform similarly to $\pi_0$**+ CoVer**.
- $\pi_0$ **(rephrase) + CoVer** Combines training-time instruction augmentation with CoVer 's inference-time optimization. This setting examines whether linguistic diversity during training and hierarchical verification at inference are complementary.

In Section 5.4, we observe that the prior test-time verification baseline, RoboMonkey, fails catastrophically on most tasks—often performing even worse than the base $\pi_0$ model. We attribute this to two primary factors. First, RoboMonkey's action verifier is trained on an action preference dataset derived from OpenVLA; however, the action distribution of $\pi_0$ differs substantially from OpenVLA. Second, due to the nature of flow-based robot policies, which generate *action chunks* rather than stepwise actions, RoboMonkey's step-level verification disrupts the structure within each chunk and frequently selects incorrect actions, resulting in lower success rates.

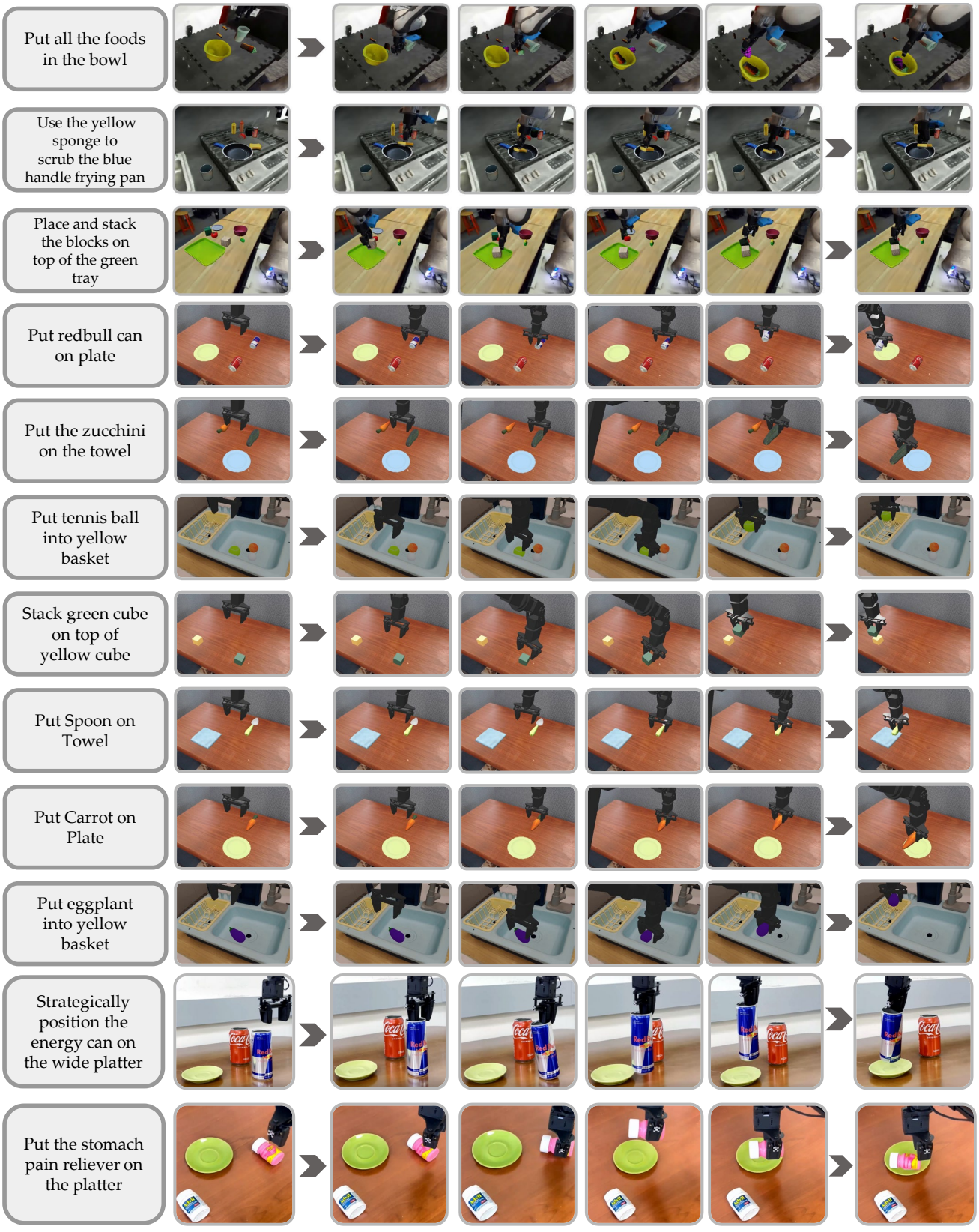


Figure 10. Task execution examples for PolaRiS, SIMPLER, and Bridge-V2 environments with corresponding original task instructions.

| Model | In-Distribution Env | | | | | Out-of-Distribution Env | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Carrot on Plate | Eggplant in Basket | Spoon on Towel | Block Stacking | Avg | Redbull on Plate | Zucchini on Towel | Tennis in basket | Avg |
| $\pi_0$ | $48 \pm 4$ | $74 \pm 3$ | $27 \pm 4$ | $17 \pm 1$ | **41.5** | $6 \pm 1$ | $30 \pm 3$ | $53 \pm 5$ | **29.7** |
| $\pi_0$ w/ Inst. Aug. [10] | $48 \pm 4$ | $68 \pm 4$ | $36 \pm 2$ | $24 \pm 4$ | **44.0** | $29 \pm 3$ | $42 \pm 3$ | $75 \pm 9$ | **48.7** |
| $\pi_0$ w/ random | $49 \pm 5$ | $78 \pm 3$ | $41 \pm 1$ | $1 \pm 1$ | **42.3** | $1 \pm 1$ | $7 \pm 2$ | $78 \pm 3$ | **28.7** |
| RoboMonkey [22] | $0 \pm 0$ | $13 \pm 3$ | $10 \pm 1$ | $7 \pm 2$ | **7.5** | $19 \pm 3$ | $10 \pm 5$ | $45 \pm 1$ | **24.7** |
| $\pi_0$+ CoVer | $48 \pm 4$ | $89 \pm 8$ | $40 \pm 6$ | $51 \pm 4$ | **57.0** | $51 \pm 3$ | $41 \pm 1$ | $91 \pm 3$ | **61.0** |
| $\pi_0$ (rephrase)+ CoVer | $52 \pm 8$ | $95 \pm 2$ | $59 \pm 5$ | $56 \pm 0$ | **65.5** | $46 \pm 3$ | $55 \pm 6$ | $85 \pm 1$ | **62.0** |

Table 3. Success rates across in-distribution and out-of-distribution tasks on the SIMPLER benchmark under red-team instructions.

## 8.3. Evaluation Results

Table 3 reports the success rates for the SIMPLER benchmarks described in Section 5.4. All evaluations are conducted using red-teaming language instructions generated by ERT [18]. The corresponding task descriptions and the full set of generated rephrases are provided in Table 8.

| Verifier Size | Backbone |
|---|---|
| 250M Verifier | ViT-B/16-CLIP |
| 500M Verifier | ViT-B/16-SigLIP2 |
| 1B Verifier | ViT-L/16-SigLIP2 |

Table 4. Verifier model size specifications.

## 8.4. Verifier Scaling Details

In Section 5.1, we investigate the scaling behavior of the CoVer verifier. Below, we detail the model architectures, dataset generation pipeline, and evaluation protocols used in these studies. For our model scaling ablation, we evaluate three distinct verifier sizes, as detailed in Table 4. We employ pre-trained image and text encoders as the backbone for all verifiers, keeping both encoders frozen during training. Notably, we observe that increasing the size of the text encoder improves downstream verification. While the 250M and 500M variants both utilize a 90M parameter image encoder, the SigLIP2-based model leverages a $7\times$ larger text encoder (280M) compared to the CLIP text encoder (40M). This indicates that the performance gains observed in the 500M model are driven primarily by improved language representation. To construct the synthetic instruction datasets, we prompt GPT-4o to generate 128 instruction variations for each original instruction in the BridgeV2 dataset. We then embed all instructions using Qwen3-Embedding-0.6B and apply $k$-means clustering to curate rephrased subsets of varying sizes ($8\times$, $16\times$, $32\times$, and $64\times$). For evaluation, we uniformly sample 1,000 $(s,a,I)$ tuples from held-out trajectories containing unseen environments and instructions from the Bridge V2 dataset. We employ GPT-4o to generate rephrased instructions, creating a fixed action pool size of 64. We report the Top-1 Action Retrieval Accuracy. Specifically, given an observation and a task description, we evaluate how often the verifier's highest-scoring action matches the ground-truth action $a$.

## 8.5. Verifier Performance Analysis

To thoroughly evaluate CoVer's capability to select the optimal action, we conduct both quantitative and qualitative analyses.

### 8.5.1. Binary Classification Performance

We first evaluate CoVer as a binary classifier to measure its ability to discriminate between aligned (ground-truth) actions and misaligned (randomly sampled) actions. The verifier demonstrates robust discriminative performance, achieving a precision of 0.765, a recall of 0.780, and an F1 score of 0.772. These results highlight CoVer's effectiveness in identifying correct actions while reliably filtering out low-quality candidates.

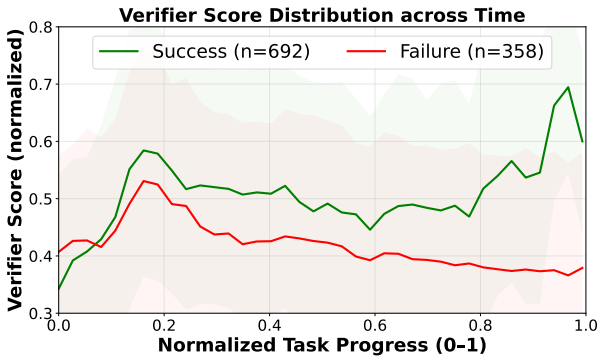


Figure 11. Visualization of verifier scores over episodes. Successful trajectories show distinct peaks during approach and completion, while failed trajectories show a steady decline.

### 8.5.2. Temporal Dynamics of Verifier Scores

To visualize the verifier's behavior over the course of a rollout, we analyze the scoring distribution across episodes (see Figure 11). We observe distinct behavioral patterns between successful and failed trajectories:

- Successful trajectories consistently receive higher scores. Notably, scores peak during two critical phases: the initial approach toward the object and the final stages of task completion.
- Failed trajectories often exhibit a steady decline in verifier scores as the rollout progresses.

This clear separation confirms the verifier's effectiveness in identifying aligned actions and highlights its potential utility as a runtime monitor for detecting and rejecting low-confidence actions during deployment.

| Batch Size | VLA ($\pi_0$) | | Image+Text Encoder | | Action Encoder | |
|---|---|---|---|---|---|---|
| | Latency (ms) | Throughput | Latency (ms) | Throughput | Latency (ms) | Throughput |
| 1 | 344.0 | 2.91 | 39.0 | 25.66 | 7.0 | 145.49 |
| 2 | 346.0 | 5.79 | 65.0 | 30.82 | 7.0 | 273.76 |
| 4 | 374.0 | 10.71 | 79.0 | 50.82 | 8.0 | 529.01 |
| 8 | 411.0 | 19.46 | 122.0 | 65.68 | 8.0 | 1060.28 |
| 16 | 523.0 | 30.58 | 173.0 | 92.48 | 8.0 | 2100.90 |
| 32 | 748.0 | 42.80 | 304.0 | 105.36 | 8.0 | 4204.31 |

Table 5. Latency (milliseconds) and throughput (samples/second) comparison across batch sizes

#### 8.5.3. Ablation Over Number of Samples.

We further investigate how the number of action candidates sampled from a VLA affects the quality of the selected action. Specifically, we define action error as the RMSE between the selected action and the ground-truth action on held-out trajectories. As shown in Table 6, increasing the number of sampled candidates consistently reduces action error. Compared to greedy decoding ($N=1$), sampling $N=16$ candidates and selecting the optimal action via CoVer reduces the action error by $11\%$.

| # Actions | RMSE |
|---|---|
| 1 | 0.166 |
| 2 | 0.155 |
| 4 | 0.149 |
| 16 | 0.147 |

Table 6. Action error (RMSE) consistently decreases as we scale the number of generated action candidates.

### 8.6. Training Computational Cost Analysis

To quantify the efficiency gains of our approach, we estimate the training FLOPs (floating-point operations) required for the base policy ($\pi_0$), the instruction-augmented policy ($\pi_0$ (rephrase)), and the CoVer verifier. We utilize the standard transformer training compute approximation $C \approx 6ND$, where $N$ denotes the number of parameters and $D$ represents the total number of training tokens, based on the hyperparameters provided by Fang et. al [10]. The verifier compute is derived from the precise forward and backward pass costs per sample. Notably, because the image and text encoders are frozen during training, the backward pass does not require gradient computation for these large backbones. This results in significantly lower compute costs: the backward pass ($\approx 1.0 \times 10^9$ FLOPs) is orders of magnitude cheaper than the forward pass ($\approx 3.3 \times 10^{11}$ FLOPs).

| Configuration | Total FLOPs | Relative Cost |
|---|---|---|
| $\pi_0$ (Base Policy) | $3.4 \times 10^{19}$ | $1.0\times$ |
| $\pi_0$ (rephrase) ($16\times$ Data) | $5.4 \times 10^{20}$ | $16.0\times$ |
| CoVer (Ours) | $1.3 \times 10^{20}$ | $3.8\times$ |

Table 7. Comparison of training computational costs.

### 8.7. Latency and Throughput Analysis

As shown in Table 5, the $\pi_0$ batch forward pass dominates latency, rising from 344ms (batch size 1) to 748ms (batch size 32). Conversely, the CoVer action encoder incurs a constant, negligible overhead of 7–8ms. Since the image-text encoder operates in parallel with $\pi_0$, the total latency of $\pi_0$+CoVer exceeds the base model by less than 10ms in all configurations. This confirms that the verifier introduces minimal cost compared to the underlying VLA policy.

Importantly, this small overhead has minimal impact in real-world settings. At the slowest tested configuration (batch size 32), the combined latency of 756 ms corresponds to a control frequency of approximately 1.3 Hz, which works for most quasi-static manipulation scenarios.

Overall, the measurements demonstrate that CoVer provides substantially improved robustness while preserving real-time feasibility. Note that we adopt the LeRobot implementation of $\pi_0$, available at: https://huggingface.co/juexzz/INTACT-pi0-finetune-bridge .

### 8.8. Generated Rephrases from Red-Teaming Instructions

Table 8 presents all instructions used across our evaluations, including original task instruction, red-team instruction, generated rephrases, and random rephrase for both SIMPLER and PolaRis.

**Original instruction.** These are template task instructions from the BridgeV2 and DROID dataset, used solely for task labeling and not included in any evaluations.

**Red-team instruction.** These challenging rephrases of bridge instructions are generated using ERT [18]. We use these generated OOD instructions to evaluate the model robustness with respect to more flexible user instructions.

**Generated rephrases.** These rephrases are produced by an off-the-shelf VLM (GPT-4o) and serve as alternative instructions during the verification process. It is worth to note that the quality of generated rephrases, does not explicitly affect the verifier performance, given that the similarity score is calculated between generated actions from rephrases and the original user instruction.

**Random rephrase.** This represents a randomly selected rephrase from the generated rephrases list, used for the baseline $\pi_0$ + random rephrase evaluation.

## 8.9. Boot-time Reasoning Implementation

**Boot-time latency.** Rephrase generation is performed once at boot time and does not incur any latency during inference, ensuring smooth execution. As such, boot-time latency is excluded in the per-step inference time reported in Table 2. For reference, generating 8 rephrases with off-the-shelf VLM takes approximately 11 seconds.

**VLM-based vs. LLM-based Rephrase Generation.** Given a user instruction, we employ an off-the-shelf VLM to interpret the scene and generate instruction rephrases. We choose a VLM for two main reasons: (i) it provides stronger scene grounding through visual inputs, and (ii) its boot-time inference cost is negligible since it is queried only once per episode. Representative rephrases produced by both the VLM and a purely text-based LLM are shown in Table 9. We observe that the VLM generated rephrases are generally more concise compared to LLM-based rephrases, which benefits the downstream VLA instruction understanding. For the task *"put the zucchini on the towel"*, LLM-generated rephrases often include ambiguous references such as "the vegetable," which is problematic in scenes containing multiple vegetables. In contrast, the VLM reliably grounds the instruction to the correct object. Similarly, for the task *"put redbull can on plate"*, the VLM produces color-specific rephrases (e.g., "blue can") that significantly improve downstream VLA performance. The LLM, lacking visual grounding, instead generates category-level terms such as "beverage," which introduces semantic drift and confuses the policy.

## 8.10. VLM Prompts for Rephrase Generation

As discussed in Section 4.3, performing rephrase generation at boot time substantially reduces inference latency by shifting both scene reasoning and linguistic diversification offline. The overall VLM prompt design follows a lightweight structure that encourages semantic preservation without imposing strong stylistic priors. The system prompt defines the high-level objective (rewriting manipulation instructions while keeping intent invariant), while the user prompt provides the specific instruction, the observed image, and a small set of minimally guiding examples. These examples serve purely as format demonstrations rather than prescriptive templates, avoiding heavy prompt engineering or over-constraining the VLM. In practice, this balance ensures that the model focuses on the objects and relations grounded in the scene rather than memorizing linguistic patterns from the exemplars. To encourage accurate grounding and reduce hallucination, the user prompt explicitly asks the VLM to (i) describe the scene in its own words, (ii) reinterpret the instruction in the context of that scene, and (iii) enumerate potential lexical variations (nouns, verbs, adjectives). This intermediate reasoning step leads to more diverse yet semantically aligned rephrases and empirically reduces the frequency of instruction drift. The full prompts used for generating rephrases are provided below.

**System Prompt.**

```
You are a text-transformation assistant for robot manipulation tasks.

You will be given:
- A user-provided instruction describing a manipulation goal,
  which may involve single or multi-step actions.

Your task is to:
1. Understand the meaning of the original instruction.
2. Reword the instruction into multiple alternatives that preserve
   the original intent, and are grammatically correct and easy to follow.
3. Try to generate easy and diverse rephrases.

Guidelines:
- Reworded instructions can be diverse in terms of words, but the
  meaning should be the same.
- Ensure all reworded instructions are semantically equivalent
  to the original.
- Use correct grammar and clear structure.
- Keep outputs concise, consistent, and logically sound.
```

**User Prompt.**

```
Given the original instruction: "{instruction}", and the appended image,
generate {batch_number} reworded instructions that convey the same objective.

Guidelines for rephrasing:
1. Use simple, clear words and actions (focus on verbs and nouns)
2. Remove adverbs whenever possible
3. Keep descriptions concise but complete
4. Infer and include object colors when they can be reasonably deduced
   (e.g., apples are typically red, strawberries are red)
5. Use diverse vocabulary across rephrases (vary nouns, verbs, and adjectives)
6. Ensure each rephrase maintains the same core meaning and task objective
7. Try to generate as diverse as possible rephrases.
8. Consider the image when generating the rephrased instructions.

Examples:
Original: "put apple on the desk"
Reworded: "pick up the red apple and place it on the desk",
          "take the apple and put it on the desk",
          "place the red fruit on the desk"

Original: "put cooking pot in the green basket"
Reworded: "move the silver cooking pot to the green basket",
          "take the cooking pot and put it in the green basket",
          "put the utensil into the green basket"

Original: "put strawberry on top of the fridge"
Reworded: "put the red fruit on the fridge",
          "place the red berry on the top of the fridge",
          "set the red berry on the top of the refrigerator"
```

```
Original: "lift the water bottle and place it on the desk"
Reworded: "pick up the transparent bottle and place it on the wooden desk",
          "take the hydration bottle and put it on the desk",
          "place the water on the desk"

Guidelines for generation:
1. You need to consider both image and instruction when generating
   the rephrased instructions.
2. You need to first generate a description of the image in your own words,
   and then think about what the language instruction means in the
   context of the image.

Format your response as:
<Description of the image>
<Meaning of the instruction in the context of the image>
Original: <Nouns> as many as possible potential replacements: <Nouns>
Original: <Verbs> as many as possible potential replacements: <Verbs>
Original: <Adjectives> as many as possible potential replacements: <Adjectives>
Original: <Adverbs>

Original Instruction:
{instruction}

Reworded Instructions:
1. <Alternative phrasing 1>
2. <Alternative phrasing 2>
...
{batch_number}. <Alternative phrasing {batch_number}>

Important: Ensure all rephrased instructions avoid adverbs, use
diverse vocabulary, and maintain the same objective as the original.
```

Table 8. **Detailed description of language instructions.** The table lists the red-team instructions used as inputs to the VLA model, along with their corresponding rephrases generated by the VLM in both SIMPLER and PolaRis. We also include one randomly selected rephrase to represent the baseline CoVer + random rephrases condition.

| Original Instruction | Red-Team Instruction | Generated Rephrases | Random Rephrase |
|---|---|---|---|
| put the zucchini on the towel | Set the green squash straight on the patterned cloth. | 1. Place the green squash upright on the blue plate.<br>2. Put the zucchini vertically on the cloth.<br>3. Position the squash straight on the fabric.<br>4. Place the zucchini upright on the plate.<br>5. Set the green vegetable vertically on the patterned fabric.<br>6. Put the squash straight on the cloth.<br>7. Position the zucchini on the blue cloth, standing up.<br>8. Place the squash vertically on the plate. | Put the squash straight on the cloth. |
| put redbull can on plate | Strategically position the energy can on the wide platter. | 1. Place the blue can on the yellow plate.<br>2. Put the energy drink on the large plate.<br>3. Set the blue drink can onto the dish.<br>4. Lay the blue can on the wide plate.<br>5. Position the energy drink on the yellow platter.<br>6. Put the blue can onto the big dish.<br>7. Place the drink can on the spacious plate.<br>8. Set the blue can on the yellow platter. | Put the energy drink on the large plate. |
| stack the green block on the yellow block | Arrange the lush green element atop the yellowish-orange element. | 1. Place the green cube on the yellow block.<br>2. Put the grassy block on top of the yellow piece.<br>3. Set the green block over the yellow cube.<br>4. Stack the green piece onto the yellowish block.<br>5. Position the lush cube above the yellow element.<br>6. Place the green cube on the yellow cube.<br>7. Set the green block on top of the yellow block.<br>8. Put the green element over the yellow cube. | Put the grassy block on top of the yellow piece. |
| put the spoon on the towel | Set the spoon exactly in the middle of the towel. | 1. Place the spoon in the center of the towel.<br>2. Put the utensil on the middle of the cloth.<br>3. Position the scoop at the center of the fabric.<br>4. Lay the green-handled spoon in the middle of the blue towel.<br>5. Set the utensil in the central part of the towel.<br>6. Position the spoon on the center of the blue towel.<br>7. Lay the green-handled spoon in the middle of the fabric.<br>8. Set the spoon in the center of the towel. | Put the utensil on the middle of the cloth. |
| put carrot on plate | Balance the carrot on the ceramic platter. | 1. Place the carrot on the green dish.<br>2. Set the orange vegetable on the small plate.<br>3. Position the carrot upright on the platter.<br>4. Stand the carrot on the green plate.<br>5. Set the vegetable on the round dish.<br>6. Position the carrot on the small platter.<br>7. Balance the orange carrot on the green dish.<br>8. Place the vegetable on the ceramic plate. | Place the carrot on the green dish. |

*Continued on next page*

Table 8 – *Continued from previous page*

| Original Instruction | Red-Team Instruction | Generated Rephrases | Random Rephrase |
|---|---|---|---|
| put eggplant into yellow basket | Arrange the eggplant neatly in the yellow bin. | 1. Place the vegetable in the yellow rack.<br>2. Put the eggplant in the yellow holder.<br>3. Set the purple vegetable into the yellow bin.<br>4. Organize the eggplant in the yellow rack.<br>5. Position the vegetable in the yellow holder.<br>6. Place the purple vegetable in the yellow container.<br>7. Set the eggplant into the dish rack.<br>8. Move the purple produce to the yellow bin. | Position the vegetable in the yellow holder. |
| put tennis ball into yellow basket | Rest the tennis equipment in the basket that is of yellow hue. | 1. Place the tennis ball in the yellow rack.<br>2. Set the tennis ball into the yellow container.<br>3. Lay the tennis ball inside the lemon-colored basket.<br>4. Put the green ball in the golden rack.<br>5. Position the tennis equipment in the yellow dish rack.<br>6. Rest the green ball into the yellow basket.<br>7. Place the tennis ball inside the yellow holder.<br>8. Lay the tennis ball in the yellow dish rack. | Position the tennis equipment in the yellow dish rack. |
| Put all the foods in the bowl | Unload all foods into the bowl | 1. Put every food item in the bowl.<br>2. Place all the food into the bowl.<br>3. Move the edibles into the yellow bowl.<br>4. Transfer all food objects to the bowl.<br>5. Gather the snacks and place them in the bowl.<br>6. Take all food pieces and put them in the bowl.<br>7. Pick up the food and drop it into the bowl.<br>8. Deposit all food items into the bowl. | N/A |
| Use the yellow sponge to scrub the blue handle frying pan | Make contact between the cleaning sponge and the frying pan with the blue handle to scrub it | 1. Scrub the blue-handled frying pan with the sponge.<br>2. Use the yellow sponge to scrub the pan with the blue handle.<br>3. Clean the blue-handled skillet using the sponge.<br>4. Rub the sponge against the frying pan with the blue handle.<br>5. Wipe the blue-handled pan with the yellow sponge.<br>6. Use the sponge to clean the frying pan having a blue handle.<br>7. Scrub the pan with the blue handle using the yellow sponge.<br>8. Apply the sponge to the blue-handled frying pan to clean it. | N/A |
| Place and stack the blocks on top of the green tray | Carefully lay the square cubes upon the green tray | 1. Put the cubes on the green tray.<br>2. Place the square blocks onto the green tray.<br>3. Move the cubes to the green tray.<br>4. Set the blocks down on the green tray.<br>5. Position the square cubes on the green tray.<br>6. Place the cubes into the green tray.<br>7. Transfer the square blocks to the green tray.<br>8. Put the square cubes on top of the green tray. | N/A |

| Original Instruction | VLM Rephrases | LLM Rephrases |
|---|---|---|
| put the zucchini on the towel | • Place the green squash upright on the blue plate.<br>• Put the zucchini vertically on the cloth.<br>• Put the squash straight on the cloth.<br>• Position the zucchini on the blue cloth, standing up. | • Place the green squash upright on the decorative fabric.<br>• Position the vegetable vertically on the designed textile.<br>• Put the green produce straight on the patterned material.<br>• Arrange the squash upright on the decorative cloth. |
| put redbull can on plate | • Place the blue can on the yellow plate.<br>• Put the energy drink on the large plate.<br>• Set the blue drink can onto the dish.<br>• Lay the blue can on the wide plate. | • Place the energy drink on the large tray.<br>• Set the beverage can on the broad dish.<br>• Arrange the energy can onto the wide tray.<br>• Put the energy drink on the large dish. |
| stack the green block on the yellow block | • Place the green cube on the yellow block.<br>• Put the grassy block on top of the yellow piece.<br>• Set the green block over the yellow cube.<br>• Stack the green piece onto the yellowish block. | • Place the green object on top of the orange object.<br>• Set the green item over the yellow-orange piece.<br>• Position the verdant piece above the amber component.<br>• Put the green component on top of the gold item. |
| put eggplant into yellow basket | • Place the vegetable in the yellow rack.<br>• Put the eggplant in the yellow holder.<br>• Set the purple vegetable into the yellow bin.<br>• Organize the eggplant in the yellow rack. | • Place the vegetable in an orderly fashion into the yellow container.<br>• Organize the eggplant inside the lemon-colored basket.<br>• Position the produce tidily in the yellow box.<br>• Set the vegetable neatly in the yellow bin. |

Table 9. Representative tasks where VLM and LLM rephrases differ significantly in semantics, color grounding, and linguistic drift.

## 9. Notation

| Symbol | Description |
|---|---|
| $\mathcal{O}$,$\mathcal{A}$,$\mathcal{L}$ | Observation, action, and natural-language instruction spaces |
| $o_t$ | Visual observation at timestep $t$ |
| $h_t$ | Recent action history window (temporal context) |
| $l$ | Original user instruction |
| $l'_k$ | $k$-th language rephrase generated at boot-time |
| $\mathcal{L}_r(l)$ | Set of $K$ candidate rephrases |
| $\pi$ | Base Vision-Language-Action (VLA) policy |
| $a'_{k,j}$ | $j$-th action chunk sampled from $\pi$ conditioned on rephrase $l'_k$ |
| $\mathcal{V}_\theta$ | Contrastive verifier parameterized by $\theta$ |
| $s_{k,j}$ | Alignment score $\mathcal{V}_\theta(o_t,h_t,l,a'_{k,j})$ |
| $S_k$ | Average semantic reliability for the $k$-th rephrase distribution |
| $l^*$ | Selected optimal rephrase for current inference step |
| $a_t^*$ | Final verified action chunk selected for execution |
| $\mathbf{f}_i$,$\mathbf{a}_i$ | Normalized vision-language and action embedding vectors |
| $B$ | Training minibatch size |
| $K$,$M$ | Number of rephrases and action samples per rephrase |
| $\mathcal{D}_{\text{aug}}$ | Training dataset augmented with $N$ rephrases per task |