OOPSIERECOVER: Learning Transferable VLA Corrections from Semantic Interventions

***Abstract*— Vision-Language-Action (VLA) models can perform a wide range of robotic manipulation tasks, but often fail during deployment. Teaching them to recover from these failures requires substantial corrective data: existing approaches learn to recover from failures by directly associating corrective actions with the observations in which they were demonstrated. For instance, DAgger-like fine-tuning can require tens to hundreds of interactions, yet the resulting policy may still fail to generalize the learned corrections beyond the demonstrated scenarios. We hypothesize that semantics can capture the underlying reasons for failures and their corrections, enabling policies to learn from only a handful of corrective demonstrations and generalize those corrections to novel situations. We propose OOPSIERECOVER – a framework that uses abduction to infer what a small set of corrective demonstrations is intended to convey, representing the resulting knowledge as dense spatial representations and conditional costs. These costs capture when a behavior is undesirable and steer the policy toward more desirable action regions. We experiment across 6 tasks and variations in object poses, scenes, and related tasks, and find that only a handful of corrective demonstrations can enable a VLA to overcome failures across a broad range of novel situations, without requiring new corrective data for each variation. website: https://oopsierecover.github.io/oopsierecover-anonymous.github.io/.**

I. INTRODUCTION

Vision-language-action models (VLAs) provide broad manipulation capabilities [1]–[3], but can still exhibit systematic failures during deployment [4], such as collisions, dropping objects, etc. Human interventions offer a natural source of corrective data [5]–[8], yet existing interactive imitation methods generally treat corrections as additional state-action supervision [5], [6], making generalization depend on collecting enough corrections in similar states. Humans can instead infer a reusable principle from only a few corrections. We ask whether a robot can likewise infer what prompted an intervention, how its behavior should be steered to avoid it, and when the same correction should apply again.

An intervention provides only partial evidence about this principle. Before takeover, the robot executes the VLA actions; afterward, the human provides a preferred continuation. Instead, the takeover time localizes the context that prompted the intervention and that generalizes to other similar situations, while the subsequent human-controlled segment provides a preferred continuation from the reached state towards a more optimal situation. A description such as *"keep the mug upright when placing it on the plate"* identifies the semantic property motivating the takeover, which may remain valid despite changes in object(s) poses, robot configuration, background, or even task. Trajectories alone leave ambiguous which property mattered; language alone lacks the sensorimotor grounding needed for visuomotor control. Together, they constrain a reusable correction.

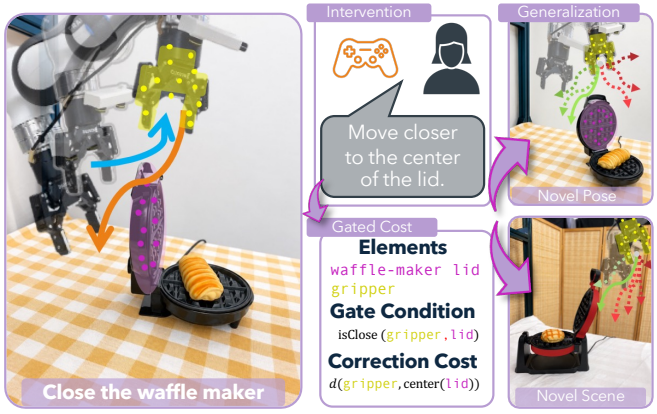


Fig. 1: Learning reusable corrections from human interventions with OOPSIERECOVER. A human corrects a lid-closure failure and describes the intended behavior. OOPSIERECOVER grounds this feedback in semantic point sets, synthesizes a gated correction cost, and uses it to steer a frozen VLA. (*Right*) The synthesized correction programs generalize: they can be reused under changed object poses and scene layouts, broadening the successful applicability of the VLA.

This ambiguity motivates OOPSIERECOVER, which formulates correction inference as *abduction*: finding a simple explanation consistent with the observed interventions despite missing or imperfect information. A vision-language model (VLM) leverages its visual-language priors, together with the scene, robot-controlled initial demonstration, human intervention, and language annotation, to propose candidate explanations. Since several explanations may fit one intervention, OOPSIERECOVER evaluates each hypothesis across all available interventions and selects explanations that capture the underlying corrective behavior without being overly narrow or permissive. In this formal-methods inspired procedure, the VLM proposes plausible semantic hypotheses, and the corrective evidence determines whether they are retained or refined.

We represent the selected explanation as one or more *gated costs*. Each combines a logical predicate specifying when the correction applies with a continuous cost specifying how the behavior should optimize/change. E.g., a gate may activate during placement, while its cost penalizes tilting the mug away from upright. This separation keeps corrections local rather globally biasing the policy. To execute these costs, OOPSIERECOVER grounds task-relevant robot parts, object parts, and spatial locations and regions as semantic point sets. At test time, the VLA samples candidate action chunks, and OOPSIERECOVER predicts how each candidate would move

these point sets, how they evaluate on the gated costs, and which candidate would lead to the lowest cost overall. The frozen VLA supplies plausible behaviors, while the inferred correction steers its choices whenever the same semantic situation reappears.

We evaluate OOPSIERECOVER on 3 tasks in both sim and real using pi0 in RoboCasa and DROID, with only 5-8 corrections per task. OOPSIERECOVER improves real-world task-progress from 35.1% to 61.0%, a 74% relative improvement over the next-best baseline, and transfers corrections to unseen poses, objects and scenes. We show that our gated costs generalize to novel variations in object poses and scenes zero-shot, and successfully steer the base policy. Our contributions are: i) a semantic, abductive formulation of few-shot VLA correction from human-in-the-loop interventions; ii) a generate-and-test procedure that synthesizes and verifies gated costs; and iii) a test-time steering mechanism that applies and generalizes them without retraining the VLA.

II. RELATED WORKS

Interactive Imitation Learning. Having a human intervene when a robot is about to fail and retraining on the resulting corrective segments is a common approach for improving control policies. DAgger [5] aggregates on-policy states labeled by an expert, while HG-DAgger [6] and IWR [9] improve sample efficiency by intervening only when the policy is likely to fail and up-weighting the resulting corrective segments during retraining. SIRIUS [7] extends this paradigm to continual learning. YAY Robot [10] incorporates semantics by training a high-level policy that produces language embeddings for a low-level policy, while OLAF [11] relabels intervened states with actions generated by a VLM conditioned on semantics to improve the base policy, requiring several dozen human corrections [12], [13].

However, these approaches primarily use interventions to relabel the actions and update the policy, limiting the ability of the corrections to generalize beyond the specific situations in which they were collected. In contrast, we focus on extracting a structured representation from the intervention and its associated semantics, allowing the correction to be applied to novel situations in a zero-shot manner.

Failure Detection for VLAs. Detecting failures is a key first step toward recovery. To this end, SAFE [14] trains a multitask failure detector on VLA latent representations to flag anomalous or unsafe executions. FailSafe [15] similarly focuses on failure reasoning and recovery signals, identifying failure modes that can trigger intervention. Failure detection is also a central component of OOPSIERECOVER, however, like most existing methods for failure detection [14], [16]–[20] these approaches require additional training or fine-tuning, whereas we target the extreme data-scarce regime of learning from only a handful of corrections. We instead leverage conformal prediction [21], [22] on our structured representations and guarded costs to detect when the policy is failing and steer it without requiring any additional training.

Inferring objectives/constraints from semantics. Another line of work is that of inverse reinforcement learning (IRL), where given a (correction induced) demonstration, the goal is to infer the reward function that defines it. One-at-a-time [23] extracts the reward by weighting predefined features, and use Bayesian IRL [24], [25] to maintain posterior over rewards. More recently, extracting reward models by leveraging the commonsense knowledge in VLMs has become an increasingly common approach [26]–[29]. Programmatic imitation learning (PRL) [30], [31] is another data-efficient approach for representing policies as programs from few demonstrations [32]. Code as policies [33] relaxes the demonstration assumption and entirely generates the policy code using an LLM. While these works represent policies as explicit, human-interpretable programs learned from demonstrations, they typically synthesize a new policy or program *de novo* for each task rather than preserving a base policy. In contrast, OOPSIERECOVER extracts a local, guarded correction that supplements a frozen, general-purpose VLA only when the relevant semantic context arises, preserving the base policy's broader capabilities.

Steering-based methods. One line of research focuses on steering generative policies using RL, either by autonomously steering the latent space [34], [35] or through human-in-the-loop interventions [8], [36]. These methods remain relatively sample-inefficient, often requiring several successful trajectories to learn an accurate Q-function for effectively steering the base policy. Works like RecoveryChaining [37] show direct sim2real transfer of recovery behaviors, however, they require re-training of recovery skills for every new scenario. In contrast, we focus on steering policies with as *few* corrections as possible, on the order of only a handful of interventions, while being able to transfer to new task instances zero-shot.

An alternative to RL-based steering is to steer the base policy with a verifier in the loop. FOREWARN [38] steers the policy by training a latent dynamics model and fine-tuning a VLM to align with it, enabling the VLM to predict the outcomes of imagined trajectories in language and select the most promising one. Do What You Say [39] generates multiple candidate trajectories using a simulator and employs a VLM as a verifier to rank them against the VLA's task-level intermediate plan. Most closely related to our work is VLS [40], which uses a VLM to extract reward functions for each task sub-stage, then generates diverse trajectories through gradient-based guidance and ranks them according to the extracted reward functions before executing the best action sequence. In contrast, OOPSIERECOVER focuses on steering the base policy with *as few corrections as possible*, using a few corrective interventions to learn reusable steering signals that generalize across tasks and scenes.

III. OOPSIERECOVER

OOPSIERECOVER learns reusable correction objectives from human interventions and uses them to steer a pretrained VLA. Fig. 2 summarizes our method. A base policy π_{base} executes a task instruction T, while a human monitors and intervenes to provide a semantic correction: a corrective

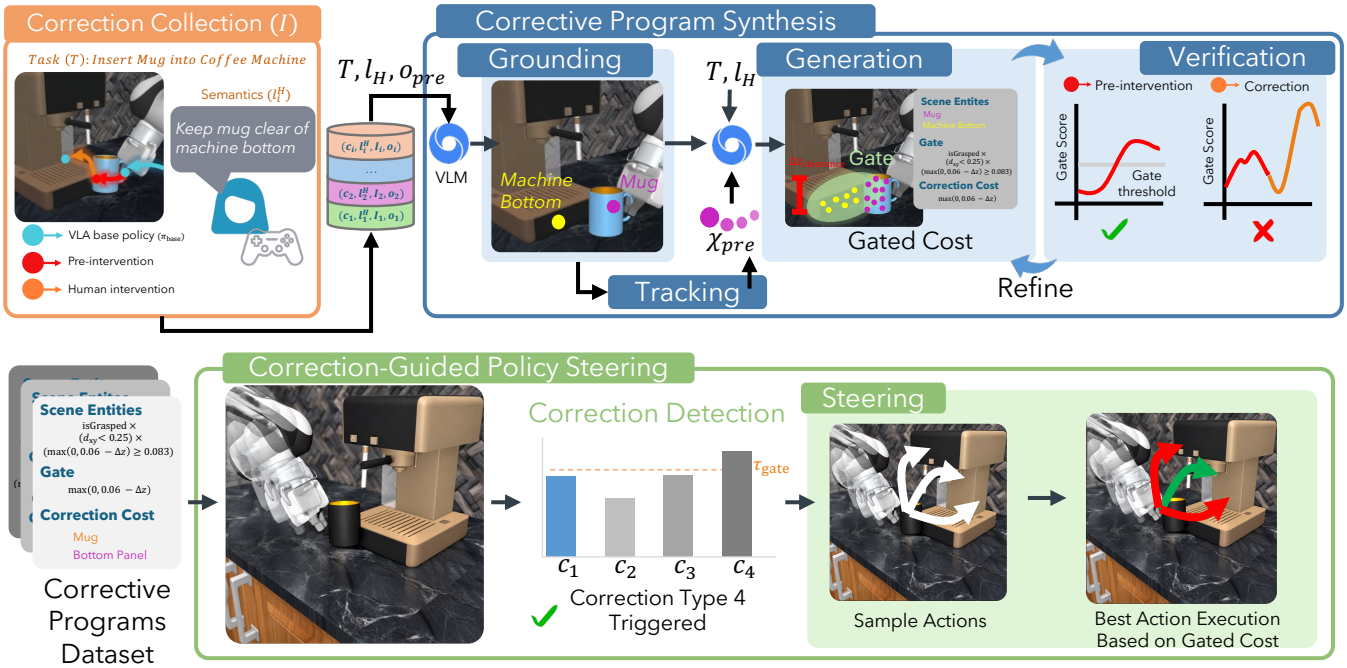


Fig. 2: OOPSIERECOVER: From semantic interventions to reusable VLA corrections. In OOPSIERECOVER, a human monitors VLA execution and intervenes to provide semantic corrections: a corrective trajectory and a language description for some type of correction, c_i, (*orange*). We then synthesize the collected set of interventions for each correction type into a *Semantically Grounded Gated Cost* by i) parsing the sensorimotor demonstration as temporal point sets tracked on semantic scene entities, ii) generating gated costs from the motion of the point sets, and iii) verifying these costs by evaluating their capability to explain all human corrections of the same type (*blue*). During deployment (*green*), these costs determine when steering is needed and rank candidate actions using the learned corrective programs. This separates the intended correction from its demonstrated trajectory, allowing correction programs to be used in the environment where they are collected and to generalize to new situations, while the base VLA remains frozen.

trajectory and correction semantics expressed as a free-form language description of the cause of the correction (Sec. III-A). OOPSIERECOVER uses these semantic corrections to synthesize programmatic spatiotemporal objectives over semantically grounded scene entities (Sec. III-C) in the form of gated correction costs (Sec. III-B). During subsequent execution, OOPSIERECOVER uses the learned gated correction costs to determine when policy steering is necessary and ranks candidate actions by their predicted costs under a learned dynamics model (Sec. III-D). Further interventions provide additional evidence for refining an existing correction when specified by the human, or introducing a new one. Throughout this process, the base VLA remains frozen, adaptation occurs through the learned correction objectives; generalization to other domains is thus achieved by using the generalization of the unaltered VLA with the transferrable nature of OOPSIERECOVER's gated correction costs.

A. Collecting Semantic Corrections

During VLA execution, a human can intervene, taking control when the robot's behavior requires correction, providing a corrective action sequence and the correction semantics ℓ_i^H, and returning control to the VLA when the situation has been corrected. For intervention episode i, we record all observations and actions as an intervention record $\mathcal{I}_i$. Within an intervention, we distinguish the human-controlled window, W^i_{human}, from a pre-intervention window, W^i_{pre}, the K steps immediately preceding human takeover [11]. The pre-intervention window, though part of autonomous execution, provides evidence for the failing behavior that prompted the intervention, while the human-controlled window provides a preferred continuation from the reached state.

After n interventions, the accumulated intervention dataset is $\mathcal{D}_n = \left\{(\mathcal{I}_i, \ell_i^H, c_i, T)\right\}_{i=1}^{n}$. where c_i denotes the human-provided correction type, a semantic category capturing the underlying corrective behavior. For each correction type c, we group the corresponding records as

$$\mathcal{D}_c = \left\{(\mathcal{I}_i, \ell_i^H) : (\mathcal{I}_i, \ell_i^H, c_i) \in \mathcal{D}_n,\ c_i = c\right\}. \qquad (1)$$

OOPSIERECOVER synthesizes a unifying correction objective from each type-based group of corrective interventions. To do so, we first define a representation capable of capturing transferable intent.

B. Corrections as Semantically Grounded Gated Costs

Human corrections can be expressed as spatiotemporal conditions among correction-relevant entities like objects, parts, and regions. For example, when a mug collides with a coffee machine during placement, the human's corrective action trajectory is specific to the current configuration but the intended correction is more general: maintain clearance from the machine while moving the mug toward the tray. Representing this underlying objective captures the transferable intent rather than the specific trajectory used in that

single intervention, reducing the need to collect action trajectories in many situations to generalize. We formalize this idea using costs: functions over geometric relations between semantically grounded entities, rather than raw states or actions, enabling transfer across object poses and scenes. Unlike general-purpose task-objectives in prior works [26], [38], [40], our costs capture behavior associated with a particular correction and should only influence execution when that correction is relevant; e.g., keeping the bottom of the mug higher than the base of the coffee machine is only *necessary* near placement.

We therefore represent each correction type as a **semantically-grounded gated cost** (also termed a **corrective program**) consisting of entity labels e (e.g., ``mug'' and ``bottom panel'') and three terms: a gate, a correction term, and a task term. The gate term, $C^c_{\text{gate}} \in 0, 1$, specifies when the correction is relevant. The correction term, $C^c_{\text{correction}}$, captures the spatiotemporal relations characterizing the human's resolution of the targeted failure, while the task term guides the policy toward actions that preserve task progress, with λ controlling this trade-off.

To ground and make such spatiotemporal costs computable from raw observations, we represent the correction-relevant entities as a tuple of entity semantic label and its associated 3D point set: $\mathcal{E}_i = (e_i, P_i^{(j)})$, where $P^{(j)} \subset \mathbb{R}^3$. An entity can correspond to an object, a selected part, a region, or a single location. Semantic association specifies which entities participate in the correction, while their point sets provide the geometry needed to ground it and evaluate spatiotemporal relations in the current scene. Unlike an object label or a single location, a point set retains the observed spatial extent of a region, allowing costs to depend on geometry rather than only a location. Singleton sets also accommodate individual locations within the same representation, such as the nozzle of the coffee machine. The total gate cost is computed as:

$$C^c(\mathcal{E}) = C^c_{\text{gate}}(\mathcal{E})\Big(C^c_{\text{correction}}(\mathcal{E}) + \lambda C^c_{\text{task}}(\mathcal{E})\Big). \quad (2)$$

C. Corrective Program Synthesis and Refinement

For a correction identifier c, let $\mathcal{D}_c$ contain the $n_c \geq 1$ interventions associated with that correction. Our inference problem is to obtain an executable corrective program that captures their shared intent. We organize synthesis into four stages: i) grounding the correction-relevant scene entities, ii) proposing a gated cost, iii) calibrating a threshold for the gate score, and iv) verifying and refining that proposal using intervention evidence.

1) Grounding relevant scene entities: The first step is to identify the entities involved in the correction and ground them with geometry in the recorded observations. For `PlaceMug`, these may include the mug, the obstructing machine region, and the target tray. We use the task instruction and correction description to localize these entities, then construct and track their 3D point-set representations.

We prompt a VLM with the task language instruction T, the correction semantic description ℓ_H, and the visual observations at the start of the pre-intervention window ($\mathcal{O}_{W_{\text{pre}_0}}$) to identify the correction-relevant scene entities (e). For each entity, the VLM specifies a unique seed point $\mathcal{K} \in \mathbb{R}^2$ in the image and determines whether the entity should be represented as a point set or a singleton. For entities requiring a point-set representation, we use the corresponding seed point to prompt a segmentation model [41], obtaining a 2D region from which we sample pixels and lift them into 3D using depth maps to construct grounded 3D point sets ($\mathcal{P}$). Singleton entites are instead lifted directly from their seed points and represented as individual 3D points.

2) Proposing a gated cost: Given the grounded scene entities, we now want to generate a gated cost. In order to generate a reasonable gate and correction cost, we need the VLM to understand how the correction-relevant entities evolved over time during the pre-intervention and human-correction windows. To obtain grounded point sets throughout the pre-intervention and human correction trajectories, we track the identified regions across frames and resample pixels before projecting them into 3D at each timestep. This avoids computationally expensive and noisy tracking of individual 3D points while preserving correspondence with semantic regions identified by the VLM.

Let $\mathcal{P}_t^{(j)}$ denote the point set corresponding to entity j at timestep t. Rather than providing the full sequence of point sets to the VLM, we summarize their temporal evolution using all their centroids $\mathbf{x}_t^{(j)} = \text{centroid}\Big(\mathcal{P}_t^{(j)}\Big)$ over time $\mathcal{X}^{(j)}_{\text{pre}} = \Big\{\mathbf{x}_t^{(j)}\Big\}_{t=t_0}^{t_1}$. We denote the collection of these trajectories by $\mathcal{X}_{\text{pre}}$. They provide a compact representation of the temporal evolution of the relevant scene entities while retaining their semantic correspondence across time.

We prompt the VLM with the task instruction T, correction description ℓ_H, the semantic entity labels e and pre-intervention & human-correction centroid trajectories $\mathcal{X}_{\text{pre}}$. The VLM synthesizes a grounded, executable cost C^c with the gate, correction, and task-specific terms in Eq. (2). The resulting corrective program is treated as an abductive interpretation of the intervention.

3) Intervention-based calibration: The proposed gated-cost specifies when and what to measure but not how much violation should count as triggering the correction; this threshold must be grounded in the evidence that motivated the intervention itself, rather than left to the VLM's guess. We therefore calibrate the gate against the recorded pre-intervention window before using it both to verify the proposal and, later, to detect when the correction applies and steer the policy at deployment (section III-D).

To measure whether a state exhibits the condition addressed by correction c, we combine its gate and correction terms into a gate score:

$$s^c_{\text{gate}}(\mathcal{P}) = C^c_{\text{gate}}(\mathcal{P}) C^c_{\text{correction}}(\mathcal{P}). \quad (3)$$

For a recorded trajectory, we write $s^{c,(t)}_{\text{gate}} = s^c_{\text{gate}}(\mathcal{P}_t)$.

We calibrate a threshold from a recorded pre-intervention window W_{pre}:

$$\tau^c_{\text{gate}} = \min_{t \in W_{\text{pre}}} s^{c,(t)}_{\text{gate}}. \quad (4)$$

This sets the threshold to the lowest gate score observed in the calibration window immediately preceding the human intervention, assuming that reaching this level of cost violation triggered the human intervention. When synthesizing a gated cost from more than one corrections ($n_c > 1$), we use the most restrictive threshold.

4) Gated cost verification and refinement: The initial gated-cost proposal is synthesized before it is checked against the actual intervention actions and may therefore fail to capture the intended correction. The human's recorded intervention offers a natural test for the original proposal: a correct C_{gate} and $C^c_{\text{correction}}$ should agree with what the human actually did. The task terms C^c_{task} are excluded from refinement, as our verification checks target the evidence for the human correction. Given the grounded 3D point sets, we formalize this as two empirical consistency checks:

Ranking check. First, we check whether the gate score ranks the human correction as an improvement over the pre-intervention behavior. We divide the recorded trajectory into equal-length windows for the pre-intervention segment, the human correction, and VLA controlled segments. For each window W, we compute its mean gate score:

$$\bar{s}^c(W) = \frac{1}{|W|} \sum_{t \in W} s^{c,(t)}_{\text{gate}}. \tag{5}$$

We require $\bar{s}^c(W_{\text{pre}}) > \bar{s}^c(W_{\text{human}})$ and, when the corresponding base policy's execution window is available, $\bar{s}^c(W_{\text{pre}}) > \bar{s}^c(W_{\pi_{\text{base}}})$. These inequalities test whether the synthesized score assigns larger costs to the pre-intervention behavior than to the comparison segments (human correction and π_{base}'s execution window). Passing this check indicates that the synthesized cost is consistent with the observed correction direction, assigning lower cost to behaviors that follow or reflect the correction.

Trigger check. Second, we require the gate score to exceed the calibrated threshold at least once within the pre-intervention segment:

$$\max_{t \in W_{\text{pre}}} s^{c,(t)}_{\text{gate}} > \tau^c_{\text{gate}}. \tag{6}$$

This checks whether the calibrated trigger would activate on at least one state preceding the human intervention. Both checks concern consistency with recorded behavior; they do not guarantee that the program captures the intended correction in every possible situation.

If the verification criteria are satisfied, the gated cost is accepted. Otherwise, we refine it using $(T, \ell_H, \mathcal{X}_{\text{fail}}, C^c, \mathcal{V})$, where $\mathcal{X}_{\text{fail}}$ is the motion of the point sets in the failed segment, and $\mathcal{V}$ contains the verification evidence, including ranking failures, trigger failures, and gate-calibration failures.

Refinement. An unsuccessful verification doesn't mean starting over, it signals which part of the proposal could be wrong. We decompose refinement into two decisions, made sequentially within a single VLM pass: which terms need fixing, and what kind of fix each needs. First, the VLM determines which cost terms require refinement: C_{gate}, $C_{\text{correction}}$, or both. Second, for each selected term, it determines whether to modify its parameters, its logical structure, or both. The VLM then generates a revised gated-cost based on these decisions and the available evidence. We repeat verification and refinement until all criteria are satisfied or a maximum refinement budget B is exhausted. If the budget is exhausted without passing both checks, we retain the parameters and logic from the final round. When multiple corrections share the same type, verification is performed jointly across correction episodes: the ranking and trigger checks are evaluated for each episode and combined using a logical AND ($\wedge$), such that failure on any episode causes the verification round to fail and triggers further refinement. The gating threshold is likewise calibrated jointly across episodes using the minimum pre-intervention gate score, imposing a single threshold that must satisfy the gating constraint for all corrections of that type.

D. Correction-Guided Policy Steering

Transferring corrections. At test time, when π_{base} encounters a previously seen failure, we cannot simply replay the recorded correction recorded correction — it was tied to that one scene, and the new one differs. What transfers is not the trajectory but the semantics: the entities the correction referred to, such as a mug or a machine's obstructing edge What transfers is the semantics, not the trajectory. We therefore task the VLM with re-finding these same correction-relevant entities in the new scene. To do so, the VLM predicts new seed points conditioned on the reference images with their seed points overlaid, jointly across all correction types. Each transferred seed is then expanded into a 3D point set (or singleton) and tracked as described in section III-B. For correction type c, we compute $s^c_{\text{gate}}(\mathcal{P}_t)$ and trigger correction-based steering when $s^c_{\text{gate}}(\mathcal{P}_t) > \tau^c_{\text{gate}}$.

Policy Steering. When steering, we first sample K candidate action chunks of horizon H: $\{a^{(k)}_{1:H}\}_{k=1}^K \sim \pi_{\text{base}}$, and roll each chunk out using a point-based dynamics model.

$$\widehat{\mathcal{P}}^{(k)}_{1:H} = f_{\text{dyn}}\left(\mathcal{P}_0, a^{(k)}_{1:H}\right), J^{(k)} = \sum_{h=1}^{H} \gamma^{h-1} C^c_{\text{guard}}\left(\widehat{\mathcal{P}}^{(k)}_h\right). \tag{7}$$

Here, $\mathcal{P}_0$ denotes the current point sets, f_{dyn} is the dynamics model, and $\gamma \in [0, 1)$ is the discount factor which places greater weight on cost violations occurring earlier in the rollout.

Given these predictions $\widehat{\mathcal{P}}$, we rank the candidate action chunks according to their predicted gated costs and select the lowest-cost candidate for execution:

$$k^\star = \underset{k \in \{1,\dots,K\}}{\arg\min}\ J^{(k)}, \qquad a^\star_{1:H} = a^{(k^\star)}_{1:H}. \tag{8}$$

To increase action diversity, we optionally augment the candidate set with cardinal actions [36]. While other approaches can be used to generate diverse candidate chunks [35], [40], we use cardinal actions for their simplicity. Further, if multiple corrections are triggered simultaneously, we rank candidate chunks using the weighted sum of their



Fig. 3: Simulation and real-world tasks. We evaluate a diverse set of tasks that elicit different types of corrections; avoiding collisions, improving placement precision, and correctly articulating objects.

active steering costs, allowing all triggered corrections to jointly influence action selection.

IV. Experiments

We aim to answer three questions with our experiments:

Q1. Can OopsieRecover learn from a small number of human corrections and generalize to new scenarios?

Simulation Experiments: We first evaluate the zero-shot generalization of OopsieRecover in simulation using the RoboCasa benchmark [42]. We consider three tasks: `CoffeeMugSetup`, `BenderLid`, and `StartKettle`, as illustrated in Fig. 3. For all experiments, we use a $\pi_{0.5}$ [1] VLA policy finetuned on the full RoboCasa dataset [43]. For each task, we evaluate all methods under three levels of scene and object-pose variation to assess generalization: $\mathcal{L}_0$ uses the same reset state in which the correction was collected; $\mathcal{L}_1$ uses different initial object poses within the same scene; and $\mathcal{L}_2$ uses different initial object poses in unseen scenes. We collect 5–8 corrections per task sequentially, one per round, with policy execution in each round steered by all corrections collected in previous rounds. All correction-based baselines are trained on this same data. The baselines are:

- $\pi_{0.5}$: The base π_0 policy finetuned on the full RoboCasa dataset, without inference-time steering.
- `HG-DAgger`*: A variant of human-gated DAgger [6] that uses the same human intervention data collected for OopsieRecover. We LoRA-finetune $\pi_{0.5}$ on the full set of collected interventions for 20K iterations.
- `OLAF`: A re-implementation of OLAF [11], where $\pi_{0.5}$ is finetuned on intervention data in which the pre-intervention actions are relabeled by a VLM.
- `VLS` [40]: An inference-time steering method that extracts a differentiable reward function. Unlike OopsieRecover, it relies solely on the task description to generate the reward function, whereas we use human corrections to learn correction programs.

Results. Fig. 4 shows zero-shot generalization across the three simulation tasks. Statistical significance is determined using the STEP test [44], with compact letter displays (CLD) [45] denoting significantly different groups at a global confidence level of 95%. Each method is evaluated over 50 episodes, with 5–8 corrections collected per task. The base policy, $\pi_{0.5}$, performs poorly across. Finetuning with the collected interventions (`HG-DAgger`$*$) improves $\mathcal{L}_0$ success by 40%, but fails to generalize to $\mathcal{L}_1$ and $\mathcal{L}_2$, where the initial state differs from the correction scenarios. `OLAF` generally underperforms `HG-DAgger`$*$, as the raw actions selected by the VLM are often inaccurate. Similarly, `VLS` fails to generalize, as the task rewards provide insufficient signal to effectively steer the policy, resulting in failures such as collisions. These failures highlight where additional correction supervision is needed to learn more informative costs. In contrast, OopsieRecover achieves strong performance in $\mathcal{L}_0$ while also generalizing to $\mathcal{L}_1$ and $\mathcal{L}_2$ zero-shot. The CLD analysis confirms this generalization, with OopsieRecover forming its own statistically separated top-performing group at both difficulty levels. This demonstrates that OopsieRecover learns transferable corrective behaviors rather than overfitting to the correction scenarios.

Real World Experiments: We use the DROID hardware setup [46] and the official DROID checkpoint of OpenPI ($\pi_0.5$). We consider three tasks, `Put Croffle`, `Load Coffee Pod`, and `Shelve item` (Fig. 3). Each of these tasks introduces diverse failure modes like misalignment on grasps and placements, being unable to close the lid, toppling fragile objects, etc. We use $3-6$ semantic corrections for each task. We again evaluate on the three levels of $\mathcal{L}_0$, $\mathcal{L}_1$ and $\mathcal{L}_2$. Each evaluation is done for 20 rollouts. We consider two baselines: $\pi_{0.5}$ and `HG-DAgger`*. We report task progress rather than only binary task success.

Fig. 4 shows the real-world task-progress results. Since real-world evaluation uses partial task-progress (fraction of stages completed) rather than binary success, statistical significance is instead determined based on the NSCORE sequential test [47], again reported via CLD at a global confidence level of 95%. We find that OopsieRecover outperforms the next-best baseline by at least 60% on task progress at every difficulty level tested and is statistically separated as the sole best method at $\mathcal{L}_0$ and $\mathcal{L}_2$, tying with $\pi_0.5$ at $\mathcal{L}_1$.

Q2. How does the performance of OopsieRecover scale with the number of corrections provided? We conduct the experiment over multiple rounds, following the standard iterative setup of DAgger on the `CoffeeMug` task. In each round, OopsieRecover receives *only* one semantically annotated correction, while `HG-DAgger`* receives ten. We report OopsieRecover after each round up to round 10, followed by two additional batches of ten corrections, for a total of 20 and 30 corrections. This enables comparison with `HG-DAgger`* at matched correction counts of 10, 20, and 30. At every checkpoint, OopsieRecover updates its behavior by steering with the accumulated corrections, whereas `HG-DAgger`* is fine-tuned according to the procedure outlined in the previous section. We collect corrections in randomly sampled scenes from $\mathcal{L}_1$ and evaluate both methods on a held-out test set of 25 scenes with

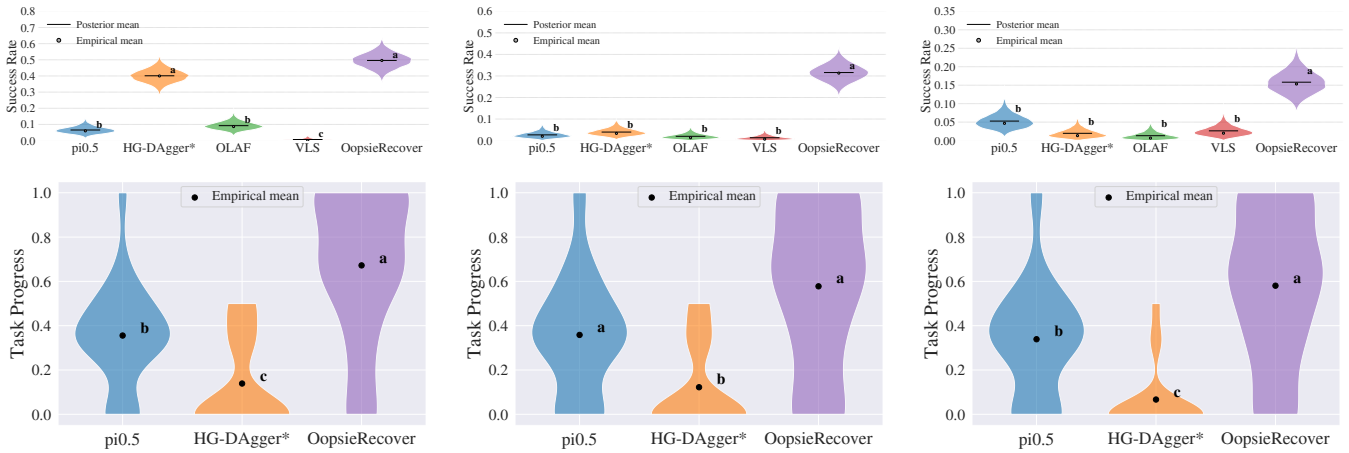


Fig. 4: Distribution of success/task-progress across difficulty levels $\mathcal{L}_0$–$\mathcal{L}_2$ (left–right), pooled over three tasks. **Top (Sim)**: zero-shot success rates; CLD letters (STEP, 95% confidence) mark statistical separation, shared letters indicate no significant difference; OOPSIERECOVER leads or ties for best at every level. **Bottom (Real)**: real-world task-progress scores; OOPSIERECOVER outperforms the next-best baseline by at least 60% and is statistically separated at $\mathcal{L}_1$ and $\mathcal{L}_2$.

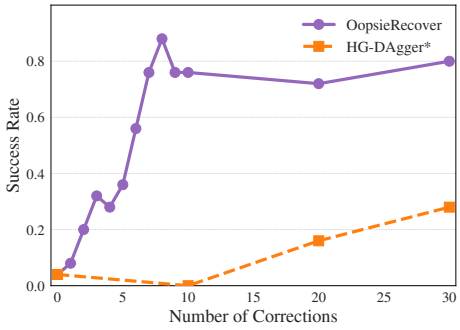


Fig. 5: Scaling with number of corrections. On `CoffeeMug`, OOPSIERECOVER rapidly reaches an 84% success rate with *as few as 10 corrections*, while `HG-DAgger`* achieves only 28% despite using three times as many corrections.

$\mathcal{L}_1$ randomization. We find that OOPSIERECOVER rapidly improves performance, reaching 84% success with just 8 corrections before saturating without degradation as additional corrections are added. In contrast, `HG-DAgger`* improves much more slowly, reaching only 28% success with the same number of corrections.

Q3. Can learned correction programs generalize across scenes to detect and prevent the same failure mode?

To evaluate generalization of learned correction programs, we conduct a controlled experiment using the correction type *"avoid colliding with the steam handle"* from the `CoffeeMug` task. We learn the corrective program from a single correction episode and test its performance across 10 novel scenes. In each trial, we induce the targeted failure by teleoperating the robot toward the steam handle, and then evaluate whether (a) the correction detector triggers, and (b) whether correction-guided steering prevents the failure over 100 environment steps. In all 10 trials, the correction detector successfully triggers, and correction-guided steering prevents the targeted failure, resulting in a 100% failure-prevention rate. These results demonstrate that the learned correction program generalizes to novel scenes and enables effective failure prevention beyond the training environment.

V. LIMITATIONS

OOPSIERECOVER targets corrections expressible as spatial conditions over semantically grounded point sets; its correction representation does not explicitly encode contact forces or unobserved physical properties. Its effectiveness depends on reliable scene grounding, point-set tracking, and motion prediction, so perception and dynamics errors can affect both when steering is triggered and which actions are selected. However, our results with HG-DAgger indicates a promising avenue: distill the corrective behaviors from OOPSIERECOVER into a image-to-action policy refinement. Steering selects among a finite set of sampled action chunks and cannot guarantee recovery when this set lacks an effective corrective behavior. Finally, the ranking and trigger checks assess consistency with recorded interventions rather than guaranteeing that the inferred program captures the intended correction in every situation.

VI. CONCLUSION

We presented OOPSIERECOVER, a framework that learns transferable corrections for frozen VLAs from a handful of semantically annotated human interventions. It uses abduction to synthesize context-gated costs over semantically grounded 3D point sets, which determine when steering is needed and rank candidate actions by their predicted consequences. Experiments in simulation and the real world show improved execution and transfer across object-pose and scene variations without additional corrective data. Compared to `HG-DAgger`*, OOPSIERECOVER achieves higher success with fewer interventions, supporting reusable corrective intent as a data-efficient alternative to directly imitating corrective actions.

REFERENCES

[1] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai et al., "$\pi_{0.5}$: A vision-language-action model with open-world generalization," 2025.

[2] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang et al., "Gr00t n1: An open foundation model for generalist humanoid robots," 2025.

[3] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi et al., "Openvla: An open-source vision-language-action model," 2024.

[4] A. Balaji, A. Bahety, S. Ambatipudi, D. Lam, J. Xu, and R. Martín-Martín, "OopsieVerse: A safety benchmark with damage-aware simulation for robot manipulation," in RSS, 2026.

[5] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in AISTATS. JMLR, 2011.

[6] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in (ICRA). IEEE, 2019.

[7] H. Liu, S. Nasiriany, L. Zhang, Z. Bao, and Y. Zhu, "Robot learning on the job: Human-in-the-loop autonomy and learning during deployment," The International Journal of Robotics Research, 2025.

[8] M. Murray, D. Chen, S. Bagaria, D. Fortier, T. Hellebrekers, G. Mullins, H. Gajarla, O. Mees, M. Cakmak, and A. Kolobov, "Flowdagger: Human-in-the-loop adaptation of generative robot policies in latent space," 2026.

[9] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese, "Human-in-the-loop imitation learning using remote teleoperation," 2020.

[10] L. X. Shi, Z. Hu, T. Z. Zhao, A. Sharma, K. Pertsch, J. Luo, S. Levine, and C. Finn, "Yell at your robot: Improving on-the-fly from language corrections," 2024.

[11] H. Liu, A. Chen, Y. Zhu, A. Swaminathan, A. Kolobov, and C.-A. Cheng, "Interactive robot learning from verbal correction," 2023.

[12] X. Sun, S. Yang, Y. Chen, F. Fan, Y. Liang, and D. Rakita, "Dynamic rank adjustment in diffusion policies for efficient and flexible training," 2025.

[13] M. Zhao, H. Admoni, R. Simmons, A. Ramdas, and A. Bajcsy, "Conformalized interactive imitation learning: Handling expert shift and intermittent feedback," in ICLR, vol. 2025, 2025, pp. 9238–9251.

[14] Q. Gu, Y. Ju, S. Sun, I. Gilitschenski, H. Nishimura, M. Itkina, and F. Shkurti, "Safe: Multitask failure detection for vision-language-action models," NeurIPS, 2026.

[15] Z. Lin, J. Duan, H. Fang, D. Fox, R. Krishna, C. Tan, and B. Wen, "Failsafe: Reasoning and recovery from failures in vision-language-action models," 2025.

[16] H. Zhang, Y. Lu, B. Wang, X. Kang, Y.-L. Kuo, Z. Cheng, M. Wang, and O. C. Jenkins, "Foresight: Failure detection for long-horizon robotic manipulation with action-conditioned world model latents," 2026.

[17] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley et al., "Robots that ask for help: Uncertainty alignment for large language model planners," 2023.

[18] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone, "Real-time anomaly detection and reactive planning with large language models," 2024.

[19] H. Liu, S. Dass, R. Martín-Martín, and Y. Zhu, "Model-based runtime monitoring with interactive imitation learning," in ICRA. IEEE, 2024.

[20] H. Liu, Y. Zhang, E. Zhang, J. Liu, C. Ding, and Y. Zhu, "Multi-task interactive robot fleet learning with visual world models," 2024.

[21] C. Xu, T. K. Nguyen, E. Dixon, C. Rodriguez, P. Miller, R. Lee, P. Shah, R. Ambrus, H. Nishimura, and M. Itkina, "Can we detect failures without failure data? Uncertainty-aware runtime failure detection for imitation learning policies," in RSS, 2025.

[22] J. Diquigiovanni, M. Fontana, and S. Vantini, "The importance of being a band: Finite-sample exact distribution-free prediction sets for functional data," 2021.

[23] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning from physical human corrections, one feature at a time," in HRI, 2018, pp. 141–149.

[24] D. Ramachandran, E. Amir et al., "Bayesian inverse reinforcement learning." in IJCAI, vol. 7, 2007, pp. 2586–2591.

[25] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," Advances in neural information processing systems, vol. 30, 2017.

[26] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, J. Fan et al., "Eureka: Human-level reward design via coding large language models," in ICLR, vol. 2024, 2024, pp. 26 516–26 560.

[27] Y. J. Ma, W. Liang, H. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "DrEureka: Language model guided sim-to-real transfer," RSS, 2024.

[28] K. Baumli, S. Baveja, F. Behbahani, H. Chan, G. Comanici, S. Flennerhag, M. Gazeau, K. Holsheimer, D. Horgan, M. Laskin et al., "Vision-language models as a source of rewards," 2023.

[29] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. Erickson, "RL-VLM-f: Reinforcement learning from vision language foundation model feedback," in ICML, 2024, pp. 51 484–51 501.

[30] J. Tremblay, T. To, A. Molchanov, S. Tyree, J. Kautz, and S. Birchfield, "Synthetically trained neural networks for learning human-readable plans from real-world demonstrations," in ICRA, 2018.

[31] J. Xin, L. Zheng, K. Rahmani, J. Wei, J. Holtz, I. Dillig, and J. Biswas, "Programmatic imitation learning from unlabeled and noisy demonstrations," IEEE RAL, 2024.

[32] N. Patton, K. Rahmani, M. Missula, J. Biswas, and I. Dillig, "Programming-by-demonstration for long-horizon robot tasks," Proceedings of the ACM on Programming Languages, 2024.

[33] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in ICRA. IEEE, 2023, pp. 9493–9500.

[34] A. Wagenmaker, M. Nakamoto, Y. Zhang, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine, "Steering your diffusion policy with latent space reinforcement learning," 2025.

[35] M. M. Hong, J. Zhang, A. Nagabandi, and A. Gupta, "Tmrl: Diffusion timestep-modulated pretraining enables exploration for efficient policy finetuning," 2026.

[36] A. Tang, W. Chen, A. Wagenmaker, C. Finn, and S. Levine, "Improving robotic generalist policies via flow reversal steering," 2026.

[37] S. Vats, D. K. Jha, M. Likhachev, O. Kroemer, and D. Romeres, "Recoverychaining: Learning local recovery policies for robust manipulation," IROS, 2025.

[38] Y. Wu, R. Tian, G. Swamy, and A. Bajcsy, "From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment," RSS, 2025.

[39] Y. Wu, A. Li, T. Hermans, F. Ramos, A. Bajcsy, and C. PÃŠrez-D'Arpino, "Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification," ICRA, 2026.

[40] S. Liu, I. S. Singh, Y. Xu, J. Duan, and R. Krishna, "Vls: Steering pretrained robot policies via vision-language models," CoRL, 2026.

[41] N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu, D. Suris Coll-Vinent, C. Ryali, K. V. Alwala, H. Khedr, A. Huang et al., "Sam 3: Segment anything with concepts," in ICLR, vol. 2026, 2026.

[42] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," in RSS, 2024.

[43] S. Nasiriany, S. Nasiriany, A. Maddukuri, and Y. Zhu, "Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots," in ICLR, vol. 2026, 2026, pp. 98 643–98 667.

[44] D. Snyder, A. J. Hancock, A. Badithela, E. Dixon, P. Miller, R. A. Ambrus, A. Majumdar, M. Itkina, and H. Nishimura, "Is your imitation learning policy better than mine? policy comparison with near-optimal stopping," 2025.

[45] H.-P. Piepho, "An algorithm for a letter-based representation of all-pairwise comparisons," Journal of Computational and Graphical Statistics, vol. 13, no. 2, pp. 456–466, 2004.

[46] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina et al., "DROID: A large-scale in-the-wild robot manipulation dataset," in RSS, 2024.

[47] D. Snyder, A. Badithela, N. Matni, G. Pappas, A. Majumdar, M. Itkina, and H. Nishimura, "Beyond binary success: Sample-efficient and statistically rigorous robot policy comparison," 2026.