D&A: Decoupling Description and Assessment Enables VLMs as Zero-Shot Robotic Reward Models

Marius Memmel[1,2], Shawn Yin[1,*], Urvi Rutia[1,*], Angad Wadhwa[1,*], Eldon Li[1], Gavin Angeloff[1]
Ankit Goyal[2], Jesse Zhang[1], Dieter Fox[1], Fabio Ramos[2], Anqi Li[2,†], Abhishek Gupta[1,†]
[1]University of Washington, [2]NVIDIA, *Equal Contribution, †Equal Advising

***Abstract*— Vision-Language Models (VLMs) hold promise for zero-shot video task-progress estimation in robot learning, yet existing frameworks frequently suffer from confirmation bias, hallucinating progress in failed executions due to implicit success assumptions or relative temporal ranking designs. Naïve prompting of frontier models collapses into a binary success detector that fails to capture nuanced, non-linear progress and failures. While post-training approaches can mitigate this through explicit training, they demand expensive real-world robotics data and require computationally intensive post-training for every individual base model. To resolve these limitations, we introduce D&A (Describe & Assess): our approach decouples progress estimation into an objective, task-grounded textual description of scene dynamics followed by an assessment phase using explicit rubrics. By compelling the VLM to synthesize an explicit, text-based definition of absolute success before scoring, the framework anchors evaluations to an external, global criterion rather than the video's internal timeline. Empirical results demonstrate that our method calibrates zero-shot estimates and mitigates confirmation bias, providing fine-grained, partial progress metrics that accurately penalize failed executions while correctly interpolating successful runs. Being purely prompt-based, D&A generalizes across tasks, embodiments, and benchmarks—matching or surpassing post-trained reward models—and its reward estimates enable efficient downstream reinforcement learning.**

Website at https://weirdlabuw.github.io/dna

I. Introduction

First impressions are notoriously difficult to overcome [1]. When evaluating complex situations, humans often exhibit confirmation bias [2], seeking and interpreting evidence that aligns with their initial hypotheses and introducing systematic errors into judgment. Similar biases appear in language models, which, trained to appear competent to human evaluators, distort answers to agree with users or take "shortcuts" to appear to solve given tasks [3, 4, 5, 6, 7, 8]. Crucially, the psychology literature shows such biases can be mitigated through structured procedures that mandate a comprehensive *description* of the available data before any task *assessment* [9], [10].

We build on this intuition to design better robotics *reward models* for downstream applications, e.g., real-world reinforcement learning and failure explanation. When naïvely querying Vision-Language Models (VLMs) for zero-shot video reward estimation, we observe these same confirmation-bias shortcuts: the models frequently hallucinate progress even when little to no meaningful advancement has occurred (see Fig. 1). Current zero-shot robotics

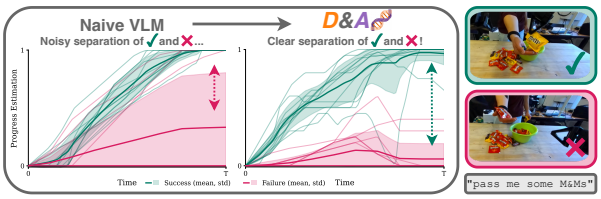


Fig. 1: **Confirmation Bias in Naïve VLM Reward Estimation:** Zero-shot querying causes models to hallucinate progress even on failed tasks, resulting in high prediction variance and no clear difference in reward estimates for successful and failed trajectories.

reward models fall victim to the same issue—frameworks like GVL [11] and TOPReward [12] estimate progress by ranking shuffled video frames or tracking token logits—our experiments show that these approaches bias towards task success. For downstream reinforcement or imitation learning algorithms, these erroneous evaluations become highly misleading reward signals that severely impede policy learning.

To bypass the limitations of zero-shot foundation models, recent approaches post-train models on task-specific robotic datasets [13], [14], [15], [16]. Yet diverse real-world robotics data, with labeled failures, is hard to obtain [17], limiting the generalization capabilities of the resulting models. Post-training is further limited to smaller, less capable open-source models and must be repeated for each individual base VLM, creating a cost, compute, and capability barrier to truly generalist robotics reward models.

Instead, we propose D&A (**D**escribe **&** **A**ssess), a zero-shot reward model that avoids the bias issues of prior zero-shot methods while achieving better visual and semantic generalization than post-trained reward models. Being zero-shot is central to its effectiveness: requiring no robotics-specific training, the same prompt-based recipe drops directly onto any promptable VLM and *improves for free* as the underlying models advance. D&A decouples evaluation into two distinct stages, *description* and *assessment*. It first forces the VLM to generate an explicit, textual *description* of what the robot actually did, inducing an objective foundation for progress measurement. It then *assesses* progress by reasoning about what task success means semantically and anchoring its reward estimate to the success criterion. Conditioned on the trajectory description and this success reasoning, the VLM

predicts progress over key frames in $[0, 1]$ as dense rewards for policy learning. We further show that this paradigm can be easily adapted to generate rich natural language feedback that goes beyond binary reward signals and provides targeted corrective context for policy improvement.

In summary, D&A is a reward model paradigm that decouples video description from progress assessment, enabling zero-shot robotics reward modeling with pre-trained VLMs. Because it is purely prompt-based, D&A is model-agnostic and requires no robotics data or fine-tuning; it generalizes across tasks, embodiments, and language, matching or outperforming post-trained models on reward benchmarks. Our scaling experiments show D&A *improves with VLM capability* across four model families and 28 open- and closed-source models. On real-world online and offline reinforcement learning with a DROID Franka and two bimanual YAM arm tasks, D&A improves success rate over both zero-shot and post-trained reward models by $+95\%$, $+15\%$, and $+45\%$, respectively. Finally, scaling up D&A to GPT-6 Astra achieves SOTA performance on `D&A-Bench`, `RBM-EVAL-OOD`, and `RoboRewardBenchmark`, surpassing all seven pre-trained and zero-shot baselines.

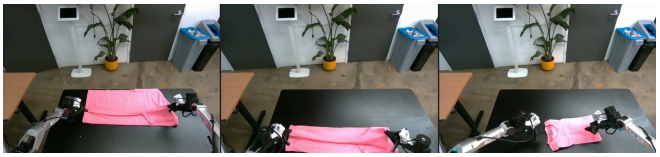
II. Related Work

Foundation models as reward models. LLMs and VLMs have been widely used to specify rewards, e.g., by generating executable reward code or shaping terms from natural language [18, 19, 20, 21], but such methods typically assume access to privileged low-dimensional state that is unavailable in real-world deployment. A more deployable alternative uses pre-trained VLMs as zero-shot progress or success estimators directly from pixels [11, 12, 22, 23, 24, 25]. D&A belongs to this zero-shot family, but whereas prior methods extract a scalar signal directly—e.g., ranking shuffled frames in GVL [11] or reading the logit of a "True" token in TOPReward [12]—we route reward estimation through explicit description and assess phases, which we find essential for countering the success bias of direct prompting.

Post-trained reward models. To improve reliability, a growing line of work fine-tunes VLMs and VLAs on robotics data with progress-prediction objectives [13, 14, 15, 16, 26, 27, 28] or language reasoning [29], [30]. RoboReward [15] predicts discretized progress labels obtained via counterfactual instruction relabeling, while Robometer [16] jointly predicts progress, success, and trajectory preferences from a large curated dataset. These models inherit the cost of collecting diverse, high-quality robot data [17], are restricted to open-source backbones, and must be retrained as stronger base models appear. D&A instead remains fully zero-shot, making it model-agnostic and improving with frontier VLM improvements, while matching or exceeding post-trained models on out-of-distribution benchmarks.

III. Demonstrating Bias in Zero-Shot VLM Prompting

We begin with a qualitative example showing that directly prompting a zero-shot VLM [31] for per-frame progress—



```
rewards = []
prev_frame = None
for frame in sample_frames(video):
    if prev_frame is None:
        rewards.append(0.0)
    else:
        reward = mean_diff(frame, prev_frame)
        rewards.append(reward)
    prev_frame = frame
rewards = normalize_0_to_1(rewards)
```

Fig. 2: Example reasoning trace generated by GPT-5.3 (queried April 11, 2026) for the *opposite instruction* "unfold the shirt."

conditioned on a robot video and its task instruction—fails to produce accurate rewards. Given an *expert demonstration* paired with the correct instruction, VLMs generally produce reasonable per-frame $[0, 1]$ rewards that end at 1.0. However, when the robot performs the task incorrectly or suboptimally, they often "shortcut" by assuming success—precisely the case a reward model must detect to correct a deployed policy.

For example, in Fig. 2 we prompt GPT-5.3 with "provide per-timestep rewards for this video of a robot 'unfolding the shirt'." However, the video (from the π_0-FAST blog [32]) in fact shows a π_0 policy *folding* a shirt. Rather than reasoning about task completion, the model writes a reward function that scores normalized *mean pixel differences* between consecutive frames, rewarding motion rather than progress. This reflects a confirmation bias arising from fine-tuning on human feedback, which makes models unlikely to challenge user assumptions [3, 4, 5, 6, 7, 8].

While prompt engineering can mitigate specific cases, our experiments (Sec. V) show that existing zero-shot reward methods—built on both closed- and open-source VLMs with custom prompting and reward-extraction schemes—still perform poorly across benchmarks, real-world tasks, and embodiments. Next, we detail our solution: decoupled *description* and *assessment* phases.

IV. D&A: Decoupling Description and Assessment

Problem Formulation. Given an image sequence (video) $o_{1:T} \coloneqq \{o_1, o_2, \ldots, o_T\}$ of length T of a robot performing a task specified by natural language instruction l, our goal is to predict a corresponding sequence of reward values $r_{1:T} \coloneqq \{r_1, r_2, \ldots, r_T\}$. Each scalar value $r_t \in [0, 1]$ represents the estimated reward at timestep t, where $r_t = 0$ indicates no progress and $r_t = 1$ corresponds to task completion. We construct a mapping function $R : (o_{1:T}, l) \mapsto r_{1:T}$ that estimates rewards from the multi-modal inputs $o_{1:T}$ and l. Prior works either post-train a VLM to learn R [13, 14, 15, 16], directly query a VLM with a custom prompt and

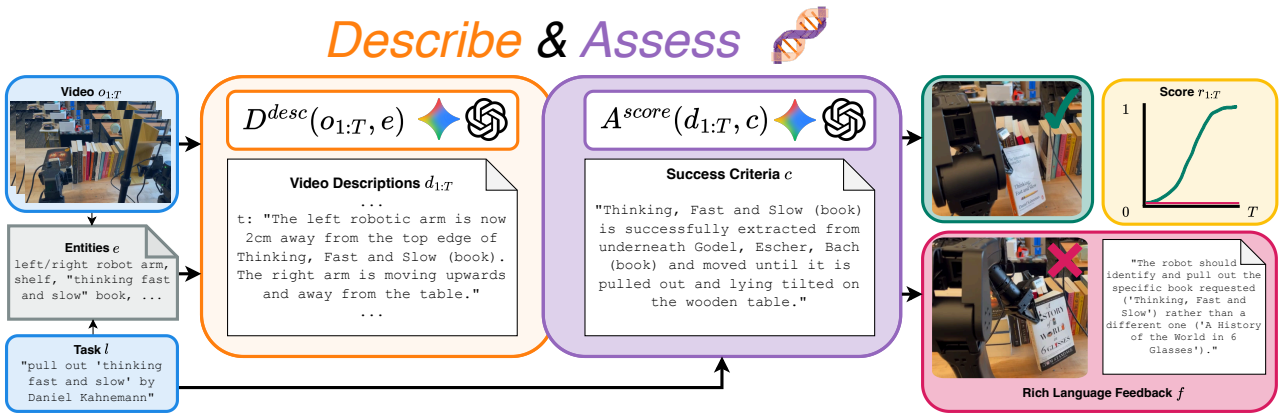


Fig. 3: **Overview of D&A.** Given a video $o_{1:T}$ and instruction l, D&A decomposes reward prediction $R(o_{1:T}, l) = A\big(l,\, D(o_{1:T})\big)$ into a *Describer* D and an *Assessor* A. D narrates the trajectory *blind to* l to avoid confirmation bias, producing a temporally consistent description $d_{1:T}$; A then scores each timestep of $d_{1:T}$ against an explicit success criterion for l, yielding dense per-frame rewards $r_{1:T} \in [0, 1]$ as well as rich language feedback.

shuffled image inputs [11, 33], or produce $r_{1:T}$ by inspecting the logits of a specific token after processing $o_{1:T}, l$ [12].

Method Overview. We would like to directly use pre-trained VLMs, zero-shot, for reward prediction; doing so allows us to leverage the generalization capabilities and learned reasoning chains of ever-improving frontier VLMs trained on internet-scale data, all without the tedium of robotics-specific post-training. But in practice, as demonstrated in Sec. III, we find that naïvely instantiating $R(o_{1:T}, l)$ as a VLM suffers from confirmation bias, where the model predicts increasing rewards regardless of whether the observation-language pairing represents a success or failure. To address these limitations, we propose Describe and Assess (D&A), which decomposes R into two components with an explicit natural-language interface:

$$\begin{aligned} &\text{Describer} \quad d_{1:T} = D(o_{1:T}), && \triangleright \text{ Describes observed behavior in language} \\ &\text{Assessor} \quad r_{1:T} = A(l, d_{1:T}). && \triangleright \text{ Scores behavior against description} \end{aligned} \tag{1}$$

Both functions can use the same VLM (or another function with the same specification) but are specified by different inputs and prompts and have no shared context. Each function has a specific task to help prevent inherent VLM biases that can contaminate reward generation. The full reward function is the composition $R(o_{1:T}, l) = A\big(l,\, D(o_{1:T})\big)$. While we use $o_{1:T}$ notation here, in practice, we linearly downsample the T frames to H frames with $H \ll T$, e.g., $H = 8$, and then linearly interpolate the rewards to length T to reduce inference cost and time. We detail the describer next in Sec. IV-A, followed by the assessor in Sec. IV-B. We provide a visual overview of D&A in Fig. 3.

A. Describing the Robot Trajectory Video

The role of the describer $D(o_{1:T})$ is to describe what the robot has accomplished in $o_{1:T}$ *unconditioned* on the task instruction l. Withholding l avoids confirmation bias [2, 3]: a describer that knows the intended task tends to narrate behavior as if that task were being accomplished, regardless of what occurred. See Fig. 3 for an example description.

Describing the full trajectory at once. In contrast to prior work [13, 14, 15, 16, 26] that predicts a single reward $r_t = R(o_{1:t}, l)$ causally and then iterates over t to label dense rewards, D&A consumes the entire trajectory $o_{1:T}$ in a single pass. By describing the trajectory holistically, D can place each timestep in the context of what happens before *and after* it, yielding descriptions that are temporally consistent.

Grounding the description in scene entities. Withholding l avoids confirmation bias but introduces a new failure mode: without any knowledge of the task, the describer may omit details that turn out to be critical for downstream assessment. Take the example in Fig. 3, where an entity extractor may describe the target book as "Thinking, Fast and Slow by Daniel Kahnemann, "the book with an orange spine", "the book with a pencil on the white cover", etc. The entity should reflect the granularity in the instruction to avoid ambiguity in the description. To resolve this without leaking l into the describer, we decompose D into two stages: an *entity extractor* E that lists *all* scene entities e, both task-relevant and task-irrelevant, from the instruction and video, and a *grounded describer* D^{desc} that narrates the trajectory anchored to those entities: $d_{1:T} = D^{\text{desc}}\big(o_{1:T},\, E(o_{1:T}, l)\big)$. Since D^{desc} does not know which entities are task-relevant, the full describer D is effectively blind to l. Next, we describe how the description is used by the assessor.

B. Success Criteria Generation and Reward Assessment

Given the grounded description d and instruction l, the assessor's role is to produce the per-timestep reward sequence $r_{1:T}$. At this stage, one might again be tempted to naïvely query an LLM/VLM directly by giving it (l, d) and asking it to score progress. In practice, however, we find this naïve version fails in a characteristic way: the assessor frequently collapses to outputting all zeros (when it cannot tell whether meaningful progress has been made) or becomes a binary success detector and simply interpolates from 0 to 1 as shown in Fig. 1. However, to generate rewards

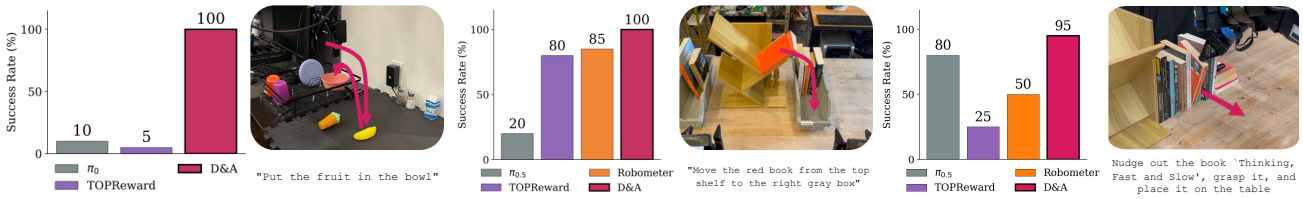


Fig. 4: **Real-world RL Results.** D&A improves real-world online (*left*) and offline reinforcement learning (*middle, right*) while other zero-shot and post-trained models struggle. **Left:** in the online setting, TOPReward fails to distinguish between success and failure, predicting high reward for both picking up the ice cream cone and the banana. **Middle, right:**Behavior cloning on suboptimal data hurts performance, while offline RL with D&A leads to a higher success rate and faster completion time. TOPReward and Robometer lack generalization and assign similar reward to optimal and suboptimal action chunks.

useful for downstream applications, we would want them to encode much more granular progress information, e.g., partial progress towards task or subtask completions. Such granular rewards would provide more useful, dense signals for policy learning, e.g., by improving the sample efficiency of reinforcement learning policies or by helping to filter out poor-quality actions in imitation learning [34].

Grounding the assessor in an explicit success criterion. This failure mode suggests the assessor lacks a clear *target* for progress: l specifies what to do, but not necessarily what describes task *success*. We address this by having the assessor first produce task completion explicitly. Concretely, given $(l, d_{1:T})$, the assessor internally generates a natural-language success criterion c—a description of the observable state of the world when l has been fully accomplished—and then scores each timestep of the description against c rather than against l alone: $r_{1:T} = A(l,\, d_{1:T}) = A^{\text{score}}\big(d_{1:T},\, C(l,\, d_{1:T})\big)$, where C generates the success criterion c and A^{score} scores the description against it. Anchoring assessment to c reframes "estimate progress toward l" as the better-posed question "how close is d to satisfying success criteria c?", helping (1) admit a meaningful notion of partial progress and (2) resist both reward collapse modes. In practice, criterion generation and scoring occur within the single assessor call A via system prompting: the model is instructed to first articulate the success condition for l, then score each timestep of d relative to it—forcing an implicit rubric [35] design through CoT reasoning [36]. See Fig. 3 for an example.

C. From Progress to Rich Language Feedback

Scalar reward signals execution failure without offering actionable context for policy correction. By evaluating trajectory progress relative to defined success criteria, D&A constructs an implicit semantic rubric over time. Tracking criteria fulfillment across timesteps enables D&A to localize execution deviations, demonstrating an implicit understanding of *how* and *why* a trajectory failed. To convert these diagnostic capabilities into downstream guidance, we augment the assessor A with a feedback generator $F(l, d_{1:T})$ that outputs a text string f containing a failure analysis and corrective instructions. This updates the assessor formulation to: $A(l, d_{1:T}) = A^{\text{score}}\big(d_{1:T}, C(l, d_{1:T}), F(l, d_{1:T})\big)$. See Fig. 3 for feedback examples.

D. Using D&A for Downstream Policy Improvement

The dense progress rewards and rich language feedback produced by D&A can be readily leveraged across various downstream policy improvement paradigms. Here we outline how D&A accelerates the paradigms used in our experiments in Sec. V, online and offline reinforcement learning (RL) and Code-as-Policies (CaP) [37]. Though we specifically detail those, D&A is a general-purpose reward and rich feedback framework and can be paired with other applications.

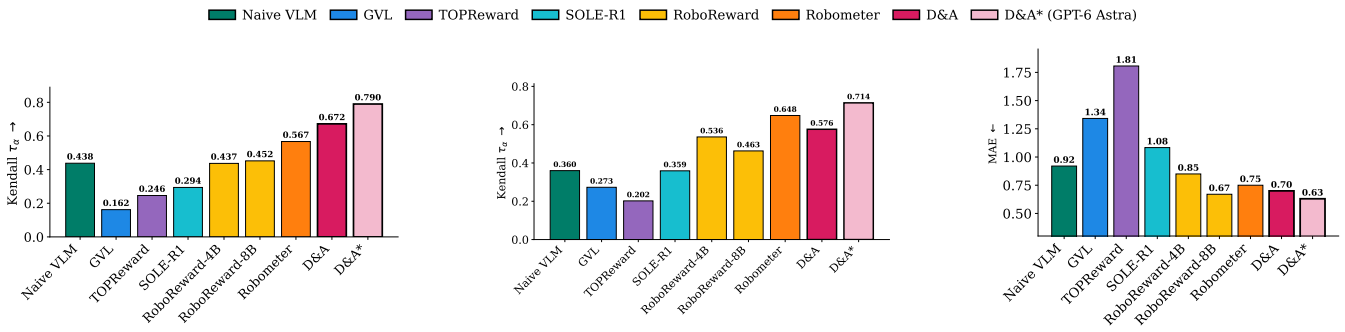
Online RL. In online RL, the agent improves its behavior through active interaction with the environment. Here, the absolute progress score $r_{1:T} = R(o_{1:T}, l)$ from D&A serves directly as the dense environment reward and can be optimized by any online RL algorithm. In our experiments, we fine-tune a frozen base policy with latent-space RL [38].

Offline RL. In offline RL, we are given a dataset $\mathcal{D} = \{\tau_n\}_{n=1}^{N}$ of trajectories *without* ground-truth reward labels, and use D&A to supply the missing reward signal. Since D&A's progress scores are bounded within $[0, 1]$ and reflect global task completion, we treat them as a *potential function* [39], from which we derive per-timestep potential-based dense rewards r_t and, in turn, offline advantages: $A_t = r_{min(t+C,T)} - r_t$ with action chunk size C. We use these advantages for filtering the training data or conditioning a policy on them (cf. RECAP and CFGRL [27, 40]); at test time, conditioning on a positive advantage biases generation towards optimal trajectories, maximizing downstream performance using D&A's zero-shot rewards.

Code-as-Policies. Recent observations in robot failure analysis [41] and feedback-driven optimization [42] suggest that rich language feedback can improve policy performance. We evaluate D&A as a between-attempt feedback source for an open-loop CaP planner. At every attempt, the planner observes the initial scene state and generates a code plan which is then executed open-loop in the environment. Across attempts, the planner keeps track of previous plans, and feedback generated by D&A which identifies the failure modes and proposes corrections.

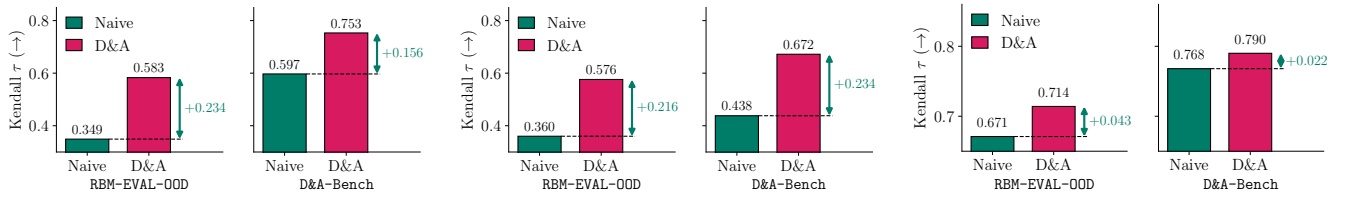
V. Experimental Evaluation

Our evaluation aims to address the following questions:



(a) D&A-Bench (↑) (b) RBM-EVAL-OOD (↑) (c) RoboRewardBench (↓)

Fig. 5: **Results on D&A-Bench, RBM-EVAL-OOD [16], and RoboRewardBenchmark [15].** D&A (Gemini 3 Flash) emerges as the strongest zero-shot method, outperforming (SOLE-R1) and matching the performance (Robometer/ RoboReward) of post-trained models. Scaling up the base model, D&A* (GPT-6 Astra), tops every benchmark.



(a) Qwen3.6-Plus (b) Gemini 3 Flash (c) GPT-6 Astra

Fig. 6: **Relative gains of D&A over Naive VLM across various base models on RBM-EVAL-OOD and D&A-Bench.**

(Q1) *Effectiveness*: Is D&A an effective reward model for real-world offline and online RL?
(Q2) *Generalization*: How well does D&A generalize to different tasks and embodiments?
(Q3) *Scaling*: How does D&A scale with the underlying VLM?
(Q4) *Ablation and analysis*: What components contribute to D&A's performance?
(Q5) *Rich feedback*: Can rich feedback generated by D&A improve over scalar rewards?

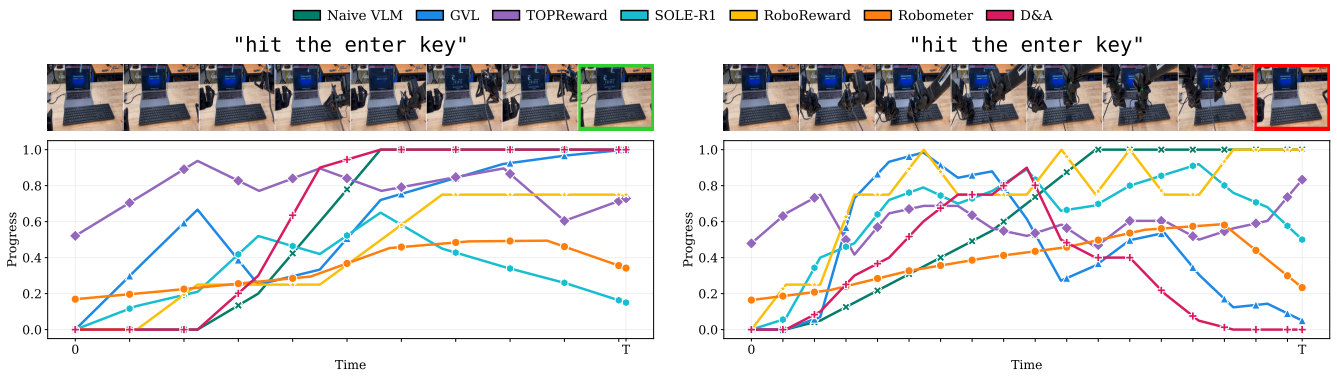
We evaluate D&A against the six strongest-performing approaches, spanning both explicitly post-trained and zero-shot reward models to provide a comprehensive analysis of the current landscape. If not denoted otherwise, we choose Gemini 3 Flash as a base model for D&A to balance cost and performance, but also report benchmark numbers for GPT-6 Astra (D&A*) to demonstrate capability scaling.

Post-trained Baselines are explicitly trained on robotics datasets to predict progress. The following approaches all use Qwen-3-VL [43] as the base model. RoboReward [15] fine-tunes to predict discrete end-of-episode scores from 1 to 5, trained on large-scale robotic data [44], [45] augmented with counterfactual language instructions that simulate failed or mismatched trajectories. Robometer [16] fine-tunes on RBM-1M (optimal and suboptimal data) with a multi-objective framework that jointly predicts continuous progress in $[0, 1]$, binary success, and pairwise trajectory preferences. SOLE-R1 [30] fine-tunes on CoT [36] and continuous progress estimates and then uses reinforcement learning from verifiable rewards (RLVR) to further tune the model.

Zero-shot Baselines leverage the world knowledge of off-the-shelf VLMs without any task- or robotics-specific fine-tuning. Naive VLM models the mapping $R : (o_{1:T}, l) \mapsto r_{1:T}$ with a single VLM call, predicting progress without decoupling or success-criteria reasoning. GVL [11] prompts a VLM with temporally shuffled frames, forcing it to judge each frame's intrinsic progress independently of chronological order to counter the VLM's bias toward time continuity. TOPReward [12] prompts a VLM with the full video and instruction and, rather than emitting progress as text, extracts it from the token logit of the "True" response to whether the instruction matches the execution. For fair comparison, we select comparable models for TOPReward (Qwen-VL 8B) and GVL (Gemini 3 Flash), and report their scaling performance on better models in Fig. 8 for **(Q3)**.

***(Q1)** Is D&A an effective reward model for real-world offline and online RL?*

Online RL. We apply DSRL [38] to the π_0 [46] policy fine-tuned on DROID [47], using the D&A progress score directly as the dense reward. We evaluate on "put the fruit in the bowl" in a DROID kitchen setup (Fig. 4, left), where the π_0 base model keeps grasping the ice cream cone and initially solves the task only 10% of the time; learning the correct behavior via RL requires rewards that discriminate good from bad state transitions. D&A provides this signal, assigning zero reward to ice-cream-cone trajectories and high, dense reward for manipulating the banana, raising the success rate to $\mathbf{100}\%$ in $\sim$90 minutes of wallclock time. In contrast,



(a) Success: the robot presses enter.

(b) Failure: the robot pressed a different key.

Fig. 7: **Qualitative `D&A-Bench` predictions for "hit the enter key":** Completion (an on-screen change) is not visible in the final frame (green: success, red: failure). D&A separates success ($r_T \to 1$) from the visually similar failure ($r_T \to 0$), while Robometer scores both alike, RoboReward/SOLE-R1 score the failure as a success, and GVL/TOPReward score both high. In the failure case, D&A captures visual nuance of the gripper close to the correct key without pressing it.

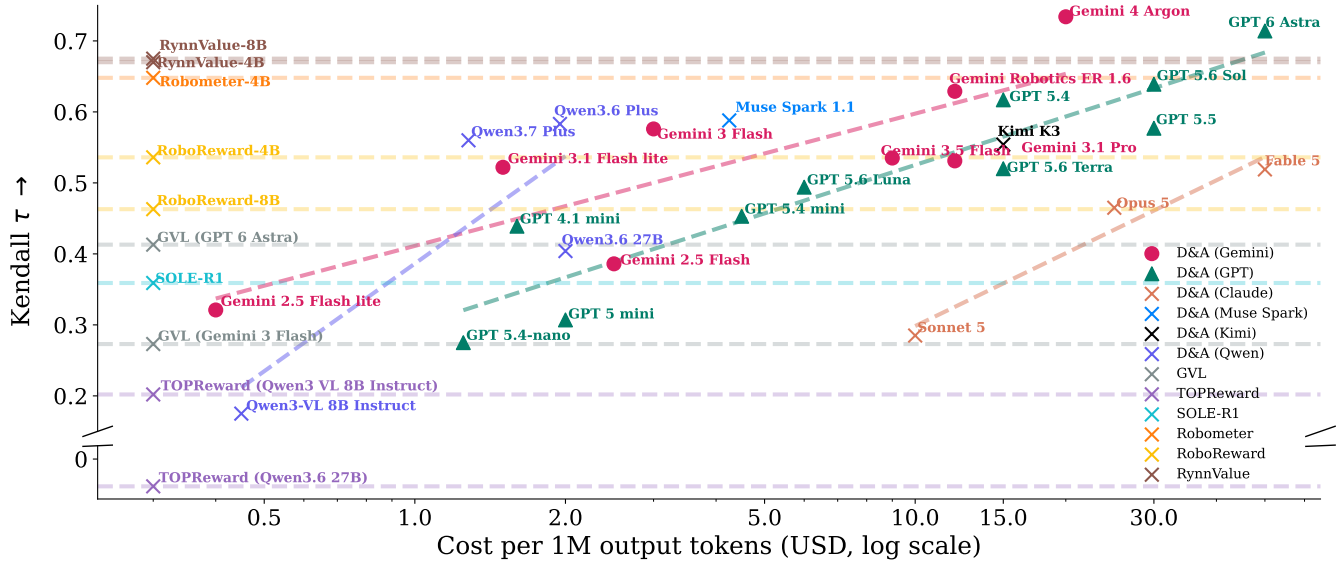


Fig. 8: **D&A scales with model capability across four open- and closed-source model families.** GVL scales much worse than D&A, while TOPReward and RoboReward show an inverse correlation between performance and model capability.

TOPReward assigns high reward to both successful and failed trajectories, leaving success at only 5%.

Offline RL. We condition $\pi_{0.5}$ [48] on advantages computed from D&A rewards via CFGRL/RECAP [27, 40], centering returns against a global value baseline estimated over the full dataset so that advantages rank state-action pairs against the dataset-wide distribution of outcomes. We collect 50 optimal, 50 suboptimal, and 50 failure demonstrations on a bimanual YAM (Fig. 4) for the task "move the red book from the top shelf to the gray box", where failures include using the wrong box or arm, or dropping the book mid-transfer. We LoRA-fine-tune $\pi_{0.5}$ for 50k steps. The baseline $\pi_{0.5}$ fine-tuned on the successful and suboptimal data only achieves 20% success, reproducing many suboptimal behaviors. TOPReward and Robometer leverage the suboptimal data better, but their noisy rewards fail to fully filter out suboptimality; D&A instead filters these behaviors, improving BC by **+80**% and outperforming all other models.

Advantage-filtered Behavior Cloning. We collect 50 optimal and 100 suboptimal demonstrations for "nudge out the book 'Thinking, Fast and Slow', grasp it, and place it on the table" on a bimanual YAM. Suboptimal demonstrations include nudging or pulling out different books but eventually complete the task. We rank all chunks in the dataset by their advantage A_t with a single dataset-wide threshold τ, so that chunks are compared against the full distribution of outcomes rather than within their own episode. We LoRA-fine-tune $\pi_{0.5}$ for 25k steps on the top 35th percentile of chunks ($A_t \geq \tau$). Robometer and TOPReward both perform poorly, struggling with the long horizon of the task

by assigning similar scores to optimal and suboptimal action chunks. Trained on the full dataset, $\pi_{0.5}$ is competitive but frequently performs suboptimal actions, leading to competitive performance but slow execution. On success, D&A completes the task faster than the baseline fine-tuned $\pi_{0.5}$, with $\mathbf{18.8 \pm 1.8}$ and 28.5 ± 9.2 seconds on average, and with markedly lower variance across rollouts. This shows that D&A filters out suboptimal and retains optimal action chunks.

(Q2) *How well does D&A generalize to different tasks and embodiments?*

To evaluate D&A across tasks, embodiments, and language, we evaluate on three offline benchmarks: D&A-Bench, a *generalization* benchmark we collected manually that is guaranteed to be out-of-distribution to all models to stress-test fine-grained scene and language understanding, and two external benchmarks RBM-EVAL-OOD [16] and RoboRewardBenchmark [15]. We propose D&A-Bench to evaluate nuanced scene understanding (e.g., "dose exactly 16 g of coffee beans"), counting ("build a 5 block tower"), and semantic reasoning ("assemble the car"). The benchmark comprises 54 (27 success, 27 failure) trajectories over 14 YAM tasks each annotated with 10 VLM-generated paraphrases of the instruction to test language robustness. We report the Kendall τ_α coefficient [49] for D&A-Bench and RBM-EVAL-OOD, measuring how well a reward model's final-step prediction align with the ground-truth ordering of optimal, sub-optimal, and failure trajectories ($\in [-1, +1]$; higher is better). RoboRewardBenchmark uses discrete end-of-episode progress labels reports Mean Absolute Error (MAE; lower is better).

Across all three benchmarks (Fig. 5), D&A closes the gap to post-trained models, outperforming RoboReward on RBM-EVAL-OOD, Robometer on RoboRewardBenchmark, and SOLE-R1 on both. On D&A-Bench, D&A leverages the generalization and reasoning capabilities of its base VLM, scoring highest (0.67) and beating the best post-trained models Robometer (0.56) and RoboReward (0.43). Zero-shot TOPReward (0.24) and GVL (0.27) fail to generalize to those challenging tasks. D&A improves +0.23 over Naive VLM, which here is a surprisingly strong baseline: because the benchmark rewards scene and language generalization, even unstructured prompting of a strong base VLM generalizes far better than GVL and TOPReward, which cannot reliably compare trajectories. Sec. V illustrates this on "hit the enter key," whose completion is only seen by a subtle on-screen change: D&A tracks the full trajectory and cleanly separates the success from a visually similar failure, whereas Robometer assigns the two nearly the same reward, RoboReward/SOLE-R1 score them oppositely, and zero-shot Naive VLM/GVL/TOPReward score both high. More qualitative examples in Sec. XIV. Finally, we demonstrate the scalability of D&A by using GPT-6 Astra as a base model (D&A*), achieving SOTA results on all three benchmarks.

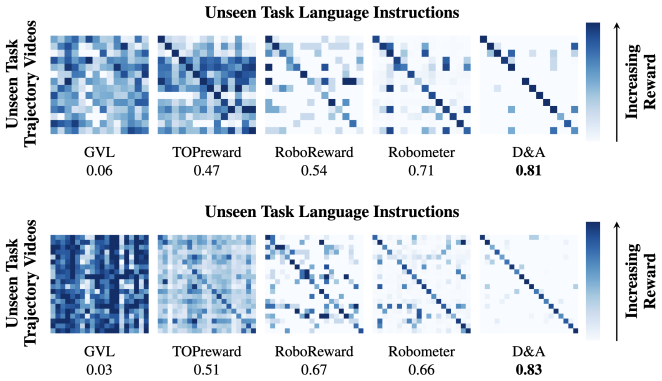


Fig. 9: **Video–language reward confusion matrices** over all video–instruction pairings from D&A-Bench (top) RBM-EVAL-OOD (bottom). The reported value is the column-normalized diagonal mean (fraction of reward on correct matches).

To evaluate semantic understanding, we plot the confusion matrices between sets of unseen task trajectory videos (rows) and unseen task language instructions (columns) in Fig. 9, where ■ means the model assigned high and ■ low reward to the combination. A reward model should discriminate mismatched video–instruction pairs while predicting high reward on the diagonal (correct matches). D&A consistently assigns zero progress to mismatched pairs while accurately scoring correct matches; post-trained models also separate the pairs, whereas zero-shot models cannot, reflecting their bias toward non-zero progress.

(Q3) *How does D&A scale with the underlying VLM?*

We investigate how D&A's ability to predict progress from video scales with base model capability. In Fig. 8, we plot trajectory ranking performance (Kendall τ_α on RBM-EVAL-OOD) against the cost per million tokens, using cost as a proxy for model capability. Baselines are shown as dashed horizontal lines. Across all base model families, we observe a clear upward trend: as D&A's base model capability increases, so does the ability to discriminate trajectory optimality. GVL also scales with the underlying model but at a much lower rate. TOPReward faces a major barrier on closed-source models through its reliance on token logprobs—which are unavailable for Gemini 3—and degrading performance under mandatory chat templates enforced by proprietary platforms. Our results further follow the negative scaling trend of TOPReward observed in the original publication [12]. D&A with Gemini 4 Argon and GPT-6 Astra as a base model achieves SOTA performance on the benchmark with 0.734 and 0.714, validating the observed trend and efficacy of our approach.

To disentangle the base model's capability from D&A's gains, we compare Naive VLM and D&A on different base models in Sec. V. On smaller models (Gemini 3 Flash, Qwen3.6-Plus), we observe the biggest improvements through D&A, outperforming RoboReward on both benchmarks and Robometer on D&A-Bench. Consequently, D&A enables competitive reward modeling in small open-

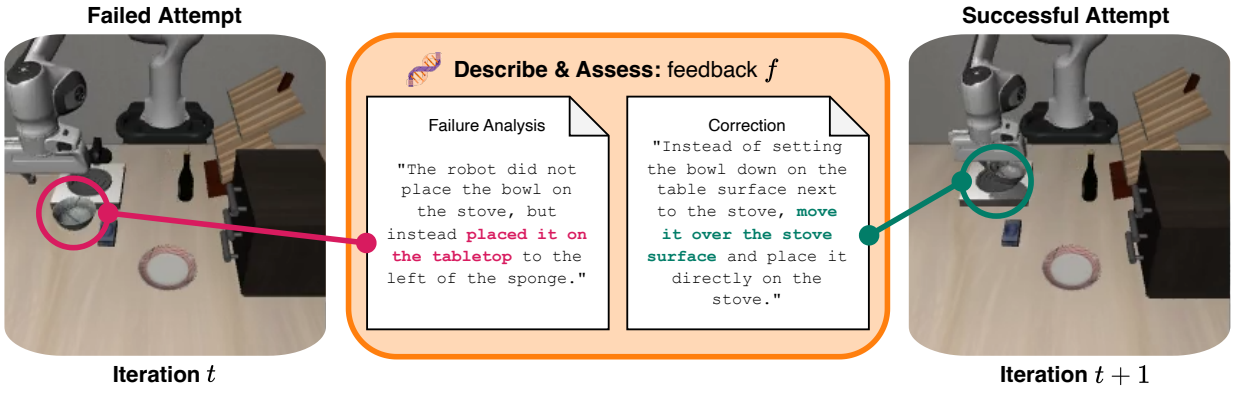


Fig. 10: **CaP with rich feedback.** Example of iterative improvement with D&A feedback in LIBERO.

Success Criteria (Sec. IV-B)	Decoupling (Sec. IV-A)	Grounding (Sec. IV-A)	Kendall $\tau_\alpha \uparrow$
✗	✗	✗	0.486
✓	✗	✗	0.561
✗	✓	✗	0.343
✗	✓	✓	0.546
✓	✓	✗	0.457
✓	✓	✓	**0.626**

TABLE I: **Ablations on `RBM-EVAL-OOD`.**

source and closed-source models. D&A applied to GPT-6 Astra leads to smaller but still substantial gains in reward modeling capabilities. Qualitative examples in Sec. X.

(Q4) *What components contribute to D&A's performance?*

To understand how each design decision contributes to the performance of D&A, we perform a series of ablations in Tab. I using Gemini 3.7 Flash. Introducing an explicit success criterion to `Naive VLM` improves performance by $+0.18$ and closes the gap between zero-shot and post-trained methods. Combining the success criteria with decoupling the description and assessment stages, negatively impacts performance due to task-relevant objects not being identifiable or being missing completely from the language description. When grounding the description in the objects present in the video, the performance improves by $+0.26$ over the `Naive VLM`, outperforming post-trained models in some cases.

(Q5) *Can rich feedback generated by D&A improve over scalar rewards?*

We evaluate D&A as a source of rich feedback to improve Code-as-Policies (CaP) [37] on the LIBERO benchmark (goal suite) [50]. The CaP planner takes multiple attempts at solving the task, generating code and executing it in the environment. We compare a history of code and binary success to extending it with rich feedback generated by D&A. Allowing CaP to iterate over its solutions by giving it history improves performance from 11.4% (no history) to 40.6% (history). Adding D&A's rich language feedback achieves 51.4% ($p = 0.00076$ over history baseline), showcasing how rich semantic feedback provides a more effective signal than scalar rewards for downstream refinement. See Sec. V for an example of feedback improving the behavior.

VI. Limitations and Discussion

In this work, we introduce D&A, a methodology for leveraging frontier models in reward modeling and rich feedback. We counter confirmation bias and address hallucinated success by decoupling description and assessment. We demonstrate the efficacy of D&A on several real-world robotic manipulation tasks. While effective, our approach introduces several limitations. First, it inherits the systemic constraints of off-the-shelf frontier models, including high inference latency, token cost, and restricted context windows. Because these models are typically accessed via closed-source APIs, they introduce stochasticity and limited granular control. They also remain susceptible to occasional semantic hallucinations, which can degrade description fidelity and subsequent reward assessment. Second, the generated reward is bounded by the granularity of natural language descriptions. This poses challenges in domains requiring highly nuanced or continuous reward structures that are difficult to verbalize, such as precise spatial relationships or absolute geometric distances. Despite these challenges, D&A marks a pivotal shift from fine-tuning to zero-shot robotics reward modeling that scales with rapid advances in frontier VLMs.

Acknowledgments

We thank Marc Toussaint and the Learning and Intelligent Systems lab at TU Berlin for the initial discussions that eventually led to this project. Part of this work was supported by TRI U3.0, Ai2, and the Cross-Pacific AI Initiative (X-PAI), which is supported by Amazon, NVIDIA, the University of Washington, and the University of Tsukuba.

REFERENCES

[1] A. P. Gregg, B. Seibt, and M. R. Banaji, "Easier done than undone: Asymmetry in the malleability of implicit preferences.," *Journal of personality and social psychology*, vol. 90, no. 1, p. 1, 2006.

[2] R. S. Nickerson, "Confirmation bias: A ubiquitous phenomenon in many guises," *Review of general psychology*, vol. 2, no. 2, pp. 175–220, 1998.

[3] S. Kim and D. Khashabi, "Challenging the evaluator: Llm sycophancy under user rebuttal," *arXiv preprint arXiv:2509.16533*, 2025.

[4] Z. Zhong, A. Raghunathan, and N. Carlini, "Impossiblebench: Measuring LLMs' propensity of exploiting test cases," in *The Fourteenth International Conference on Learning Representations*, 2026.

[5] Z. Feng, Z. Chen, J. Ma, Y. T. Po, E. Chersoni, and B. Li, "Good arguments against the people pleasers: How reasoning mitigates (yet masks) llm sycophancy," *arXiv preprint arXiv:2603.16643*, 2026.

[6] I. Petrov, J. Dekoninck, and M. Vechev, "Brokenmath: A benchmark for sycophancy in theorem proving with llms," *arXiv preprint arXiv:2510.04721*, 2025.

[7] Y. Wan, X. Jia, and X. L. Li, "Unveiling confirmation bias in chain-of-thought reasoning," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025, pp. 3788–3804.

[8] M. Cheng, S. Yu, C. Lee, P. Khadpe, L. Ibrahim, and D. Jurafsky, "ELEPHANT: Measuring and understanding social sycophancy in LLMs," in *The Fourteenth International Conference on Learning Representations*, 2026.

[9] P. Croskerry, "Cognitive forcing strategies in clinical decisionmaking," *Annals of emergency medicine*, vol. 41, no. 1, pp. 110–120, 2003.

[10] R. P. Larrick, "Debiasing," *Blackwell handbook of judgment and decision making*, pp. 316–338, 2004.

[11] Y. J. Ma et al., "Vision language models are in-context value learners," in *The Thirteenth International Conference on Learning Representations*, 2024.

[12] S. Chen et al., "Topreward: Token probabilities as hidden zero-shot rewards for robotics," *arXiv preprint arXiv:2602.19313*, 2026.

[13] S. Zhai et al., "A vision-language-action-critic model for robotic real-world reinforcement learning," *arXiv preprint arXiv:2509.15937*, 2025.

[14] H. Tan et al., "Robo-dopamine: General process reward modeling for high-precision robotic manipulation," *arXiv preprint arXiv:2512.23703*, 2025.

[15] T. Lee, A. Wagenmaker, K. Pertsch, P. Liang, S. Levine, and C. Finn, "Roboreward: General-purpose vision-language reward models for robotics," *arXiv preprint arXiv:2601.00675*, 2026.

[16] A. Liang et al., "Robometer: Scaling general-purpose robotic reward models via trajectory comparisons," *arXiv preprint arXiv:2603.02115*, 2026.

[17] R. Tian, Y. Wu, and A. Bacjsy, *Position: Good embodied reward models need bad behavior data*, 2026.

[18] Y. J. Ma et al., "Eureka: Human-level reward design via coding large language models," in *International Conference on Learning Representations (ICLR)*, 2024.

[19] W. Yu et al., "Language to rewards for robotic skill synthesis," in *Conference on Robot Learning (CoRL)*, 2023.

[20] T. Xie et al., "Text2reward: Reward shaping with language models for reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2024.

[21] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[22] S. A. Sontakke et al., "Roboclip: One demonstration is enough to learn robot policies," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[23] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang, "Vip: Towards universal visual reward and representation via value-implicit pre-training," in *International Conference on Learning Representations (ICLR)*, 2023.

[24] J. Rocamonde, V. Montesinos, E. Nava, E. Perez, and D. Lindner, "Vision-language models are zero-shot reward models for reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2024.

[25] P. Yang et al., "Rarm: Confidence-gated progress reward modeling for rl in manipulation," *arXiv preprint arXiv:2606.22027*, 2026.

[26] J. Zhang et al., "Rewind: Language-guided rewards teach robot policies without new demonstrations," *arXiv preprint arXiv:2505.10911*, 2025.

[27] P. Intelligence et al., "$\pi^*_{0.6}$: A vla that learns from experience," *arXiv preprint arXiv:2511.14759*, 2025.

[28] D. Huang et al., "Rynnvalue: Scaling robotic value foundation models with temporal distance," *arXiv preprint arXiv:2608.09853*, 2026.

[29] J. Zhang et al., "Progresslm: Towards progress reasoning in vision-language models," *arXiv preprint arXiv:2601.15224*, 2026.

[30] P. Schroeder, T. Weng, K. Schmeckpeper, E. Rosen, S. Hart, and O. Biza, "Sole-r1: Video-language reasoning as the sole reward for on-robot reinforcement learning," *arXiv preprint arXiv:2603.28730*, 2026.

[31] A. Singh et al., "Openai gpt-5 system card," *arXiv preprint arXiv:2601.03267*, 2025.

[32] K. Pertsch et al., "Fast: Efficient action tokenization for vision-language-action models," *arXiv preprint arXiv:2501.09747*, 2025.

[33] P. Budzianowski et al., "Opengvl–benchmarking visual temporal progress for data curation," *arXiv preprint arXiv:2509.17321*, 2025.

[34] A. Gupta, A. Pacchiano, Y. Zhai, S. Kakade, and S. Levine, "Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 281–15 295, 2022.

[35] A. Gunjal et al., "Rubrics as rewards: Reinforcement learning beyond verifiable domains," *arXiv preprint arXiv:2507.17746*, 2025.

[36] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[37] J. Liang et al., "Code as policies: Language model programs for embodied control, 2023," *URL https://arxiv. org/abs/2209.07753*, vol. 3, 2022.

[38] A. Wagenmaker et al., "Steering your diffusion policy with latent space reinforcement learning," *arXiv preprint arXiv:2506.15799*, 2025.

[39] A. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, 1999.

[40] K. Frans, S. Park, P. Abbeel, and S. Levine, "Diffusion guidance is a controllable policy improvement operator," *arXiv preprint arXiv:2505.23458*, 2025.

[41] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *arXiv preprint arXiv:2306.15724*, 2023.

[42] Y. Lee, J. Boen, and C. Finn, "Feedback descent: Open-ended text optimization via pairwise comparison," *arXiv preprint arXiv:2511.07919*, 2025.

[43] S. Bai et al., "Qwen3-vl technical report," *arXiv preprint arXiv:2511.21631*, 2025.

[44] O.-E. Collaboration et al., "Open x-embodiment: Robotic learning datasets and rt-x models," *arXiv preprint arXiv:2310.08864*, vol. 1, no. 2, 2023.

[45] P. Atreya et al., "Roboarena: Distributed real-world evaluation of generalist robot policies," *arXiv preprint arXiv:2506.18123*, 2025.

[46] K. Black et al., "π_0: A vision-language-action flow model for general robot control," *arXiv preprint arXiv:2410.24164*, 2024.

[47] A. Khazatsky et al., "DROID: A large-scale in-the-wild robot manipulation dataset," *arXiv preprint arXiv:2403.12945*, 2024.

[48] P. Intelligence et al., "$\pi_{0.5}$: A vision-language-action model with open-world generalization," *arXiv preprint arXiv:2504.16054*, 2025.

[49] C. Muslimani, K. Johnstonbaugh, S. Chandramouli, S. Booth, W. B. Knox, and M. E. Taylor, "Towards improving reward design in rl: A reward alignment metric for rl practitioners," *arXiv preprint arXiv:2503.05996*, 2025.

[50] B. Liu et al., "Libero: Benchmarking knowledge transfer for lifelong robot learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 44 776–44 791, 2023.

VII. Offline RL via Advantage Conditioning: Full Derivation

We provide the full derivation of the advantage-conditioned offline RL pipeline summarized in Sec. IV-D. Unlike standard behavior cloning (BC) or filtered BC, which are strictly constrained by the average performance of the training data, RL can yield a better-than-demonstrator policy.

Suppose we are provided an offline dataset $\mathcal{D} = \{\tau_n\}_{n=1}^{N}$ containing N trajectories. Each trajectory τ_α is a sequence of observations and actions, $\tau = \{(o_1, a_1), (o_2, a_2), \ldots, (o_T, a_T)\}$. Recall that we treat the progress scores $r_t = R(o_{1:T}, l)$ as a potential function and define the step-wise potential-based reward as the temporal difference $\tilde{r}_t = r_t - r_{t-1}$ with $r_0 = 0$, which effectively captures the instantaneous rate of task improvement at each step.

Because we operate in an offline setting, we can compute empirical state values and advantages entirely offline without the stability risks or compounding errors of bootstrapping. For any timestep t within a trajectory τ_α, the discounted Monte Carlo return is calculated as $G_t = \sum_{k=t}^{T} \gamma^{k-t} \tilde{r}_k$, where $\gamma \in [0, 1)$ is the discount factor. To isolate how much better a specific sequence of actions is relative to the entire dataset, we establish a global value baseline V as the empirical mean of the discounted Monte Carlo returns across all timesteps and all trajectories in $\mathcal{D}$:

$$V = \frac{1}{\sum_{n=1}^{N} T_n} \sum_{n=1}^{N} \sum_{t=1}^{T_n} G_{t,n}.$$

Computing V over the full dataset (rather than per-trajectory or per-task) lets the resulting advantages rank state-action pairs against the dataset-wide distribution of outcomes, which is what CFGRL conditions on. Using this global baseline, the offline advantage $A(s_t, a_t)$ for each state-action pair is computed as the centered return: $A(s_t, a_t) = G_t - V$.

These computed advantages can be directly used for downstream offline RL algorithms. We leverage the CFGRL framework [40] to scale actions or action tokens by training a conditional policy $\pi_\theta(a_t \mid s_t, l, g)$ alongside an unconditional or baseline variant, utilizing the advantage as an explicit conditioning variable g during optimization. Following RECAP [27], we adapt this advantage-guided framework directly to a $\pi_{0.5}$ [48] and optimize the model via a standard supervised log-likelihood objective: $\mathcal{L}(\theta) = -\mathbb{E}_{(s_t, a_t, l) \in \mathcal{D}} [\log \pi_\theta(a_t \mid s_t, l, g)]$.

VIII. Online RL via Latent Space Noise Steering: Full Derivation

We provide the full optimization details for the online DSRL pipeline, summarized in Sec. IV-D. Unlike the offline setting, which benefits from static, dataset-wide baseline calculations, online policy learning requires the agent to continually explore and optimize its behavior through active environment interaction. DSRL [38] learns a policy $\pi_\theta(w \mid s, l)$ that generates a lower-dimensional latent steering vector or noise variable w, which is then mapped through the frozen base model to produce the final robot actions, rather than optimizing actions directly in the high-dimensional action space.

Under this paradigm, the agent interacts with the environment online, and the absolute task progress score $r_t = R_{1:T}(o_{1:T}, l)$ generated by D&A is used directly as the dense environmental reward. To optimize the policy over latent space interactions, we maintain a soft action-value function $Q_\phi(s, w, l)$ and a target state-value network $V_{\bar{\psi}}(s, l)$. The critic parameters ϕ are optimized by minimizing the standard temporal-difference (TD) error:

$$\mathcal{L}(\phi) = \mathbb{E}_{(s_t, w_t, r_t, s_{t+1}) \sim \mathcal{B}} \left[\left(Q_\phi(s_t, w_t, l) - \left(r_t + \gamma V_{\bar{\psi}}(s_{t+1}, l) \right) \right)^2 \right]. \tag{2}$$

Concurrently, the policy parameters θ are updated to maximize the expected future return from the critic alongside an entropy regularization term:

$$\mathcal{L}(\theta) = -\mathbb{E}_{s_t \sim \mathcal{B},\, w_t \sim \pi_\theta} \left[Q_\phi(s_t, w_t, l) - \alpha \log \pi_\theta(w_t \mid s_t, l) \right]. \tag{3}$$

By mapping this online optimization loop directly onto the latent noise space of a frozen expert, this framework mitigates catastrophic forgetting and policy collapse, enabling sample-efficient online behavior optimization powered by D&A rewards.

IX. Additional Results

Benchmark	Model	Score
RoboRewardBenchmark	D&A (GPT-6-Astra)	**0.63**
	Naive VLM (GPT-5.5 (2026-04-23) [15])*	0.66
	Naive VLM (GPT-5.5 xhigh reasoning) [15]	**0.63**
D&A-Bench	D&A (GPT-6-Astra)	**0.790**
	D&A (Gemini 3 Flash)	0.672
RBM-EVAL-OOD	D&A (GPT-6-Astra)	**0.714**
	Robometer	0.648

TABLE II: **D&A (GPT-6 Astra) vs. best reported on benchmark.**

Naive VLM	GVL	TOPReward	RoboReward-4B	Robometer	D&A
0.92	1.34	1.80	0.85	0.75	0.70

TABLE III: RoboRewardBenchmark (MAE ↓)

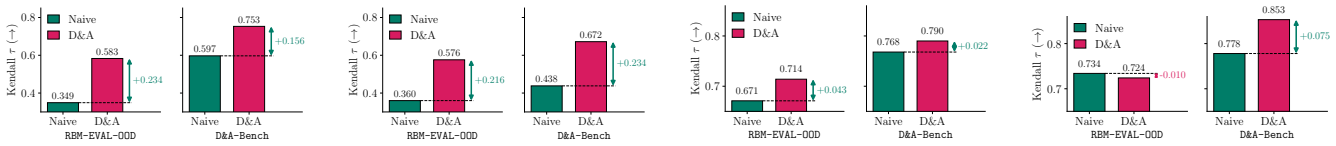
	Naive VLM	GVL	TOPReward	RoboReward-4B	Robometer	D&A
RBM-EVAL-OOD	0.950	0.855	0.796	0.842	0.934	0.885
D&A-Bench	0.962	0.014	0.490	0.780	0.945	0.914

TABLE IV: **Additional results** (VOC ↑)

X. Gains of D&A over Base Model Capabilities - Frontier Models

On smaller models (Gemini 3 Flash, Qwen3.6-Plus), we observe the biggest improvements through D&A, outperforming RoboReward on both benchmarks and Robometer on D&A-Bench (Fig. 11a and Fig. 11b). On more recent VLMs, D&A improves reward modeling capabilities, leading to SOTA results on both benchmarks (Fig. 11c and Fig. 11d). The only exception is Gemini 4 Argon on RBM-EVAL-OOD, where D&A is on par with Naive VLM. Given D&A's significant improvement on D&A-Bench, we hypothesize that Gemini 4 Argon was trained on the RBM-EVAL-OOD dataset (Google specifically advertises Gemini 4 Argon as a value function[1]), diluting the benchmark results and preventing D&A from improving.

Qualitatively, Fig. 12a and Fig. 13a both show the bias toward task success of Naive VLM even when the task is not completed successfully, while Fig. 12b and Fig. 13b show D&A's fine-grained reward assignment.



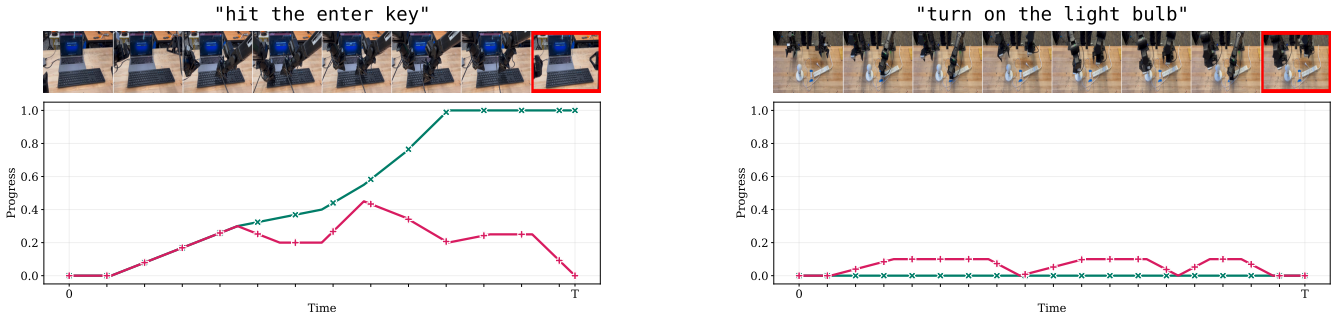
(a) Qwen3.6-Plus (b) Gemini 3 Flash (c) GPT-6 Astra (d) Gemini 4 Argon

Fig. 11: **D&A gains on frontier models**

XI. Gains of D&A over Base Model Capabilities - Gemini Model Family

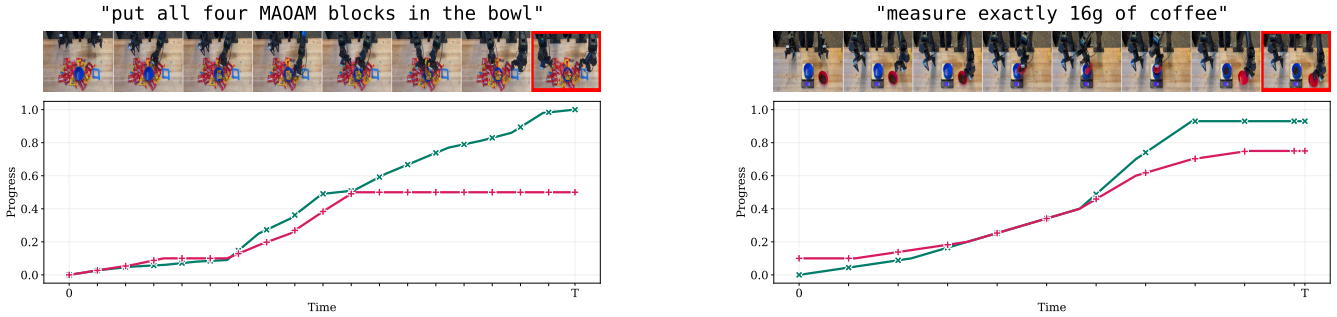
The benefits of D&A persist as the base VLM becomes stronger, achieving consistent gains when improving the underlying Gemini models (Tab. V). Crucially, these gains are not driven by increased test-time compute or reasoning, as reducing test-time compute by switching the thinking level to "low" matches results and can even improve performance (Tab. V). Furthermore, baselines are evaluated on their strongest compatible backbones—such as Qwen-VL 8B Instruct for TOPReward and Gemini 3 Flash for GVL —rather than artificially weaker models. Existing baselines face major technical barriers on closed-source models: TOPReward relies on token logprobs unavailable on proprietary platforms, GVL performs poorly across closed models, and fine-tuning methods (RoboReward, Robometer) cannot be applied at all. As a result, D&A is the only approach that successfully scales to powerful closed-source models.

[1] https://developers.googleblog.com/gemini-4-argon-for-robotics/



(a) Enter task (b) Light task

Fig. 12: **D&A qualitative gains on GPT-6 Astra**



(a) Maoam task (b) Coffee task

Fig. 13: **D&A qualitative gains on Gemini 4 Argon**

RBM-EVAL-OOD	Naive VLM	D&A	D&A
Thinking Level	medium	medium	low
Gemini 2.5 flash	0.280	0.379	0.418
Gemini 2.5 pro	0.278	0.407	0.478
Gemini 3.5 flash	0.499	0.517	0.609
Gemini 3.1 pro	0.515	0.620	0.591

TABLE V: **Backbone and compute** (Kendall $\tau \uparrow$)

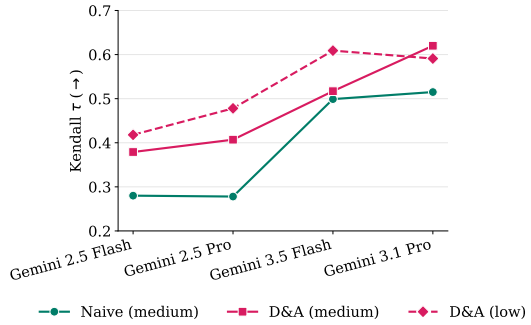


Fig. 14: **D&A gains persist across base models**

XII. D&A Cost, Wall Clock Time, and Token Usage

Model	Cost ($)	Wall Clock Time (s)	Tokens
Gemini 3.1 Pro	0.035	23.7	7,358
Gemini 3.5 Flash	0.039	25.9	8,842
Gemini 2.5 Pro	0.060	58.0	10,843
Gemini 2.5 Flash	0.015	36.9	11,177

TABLE VI: Average compute, latency, and financial cost per trajectory evaluated across 567 trajectories on RBM-EVAL-OOD.

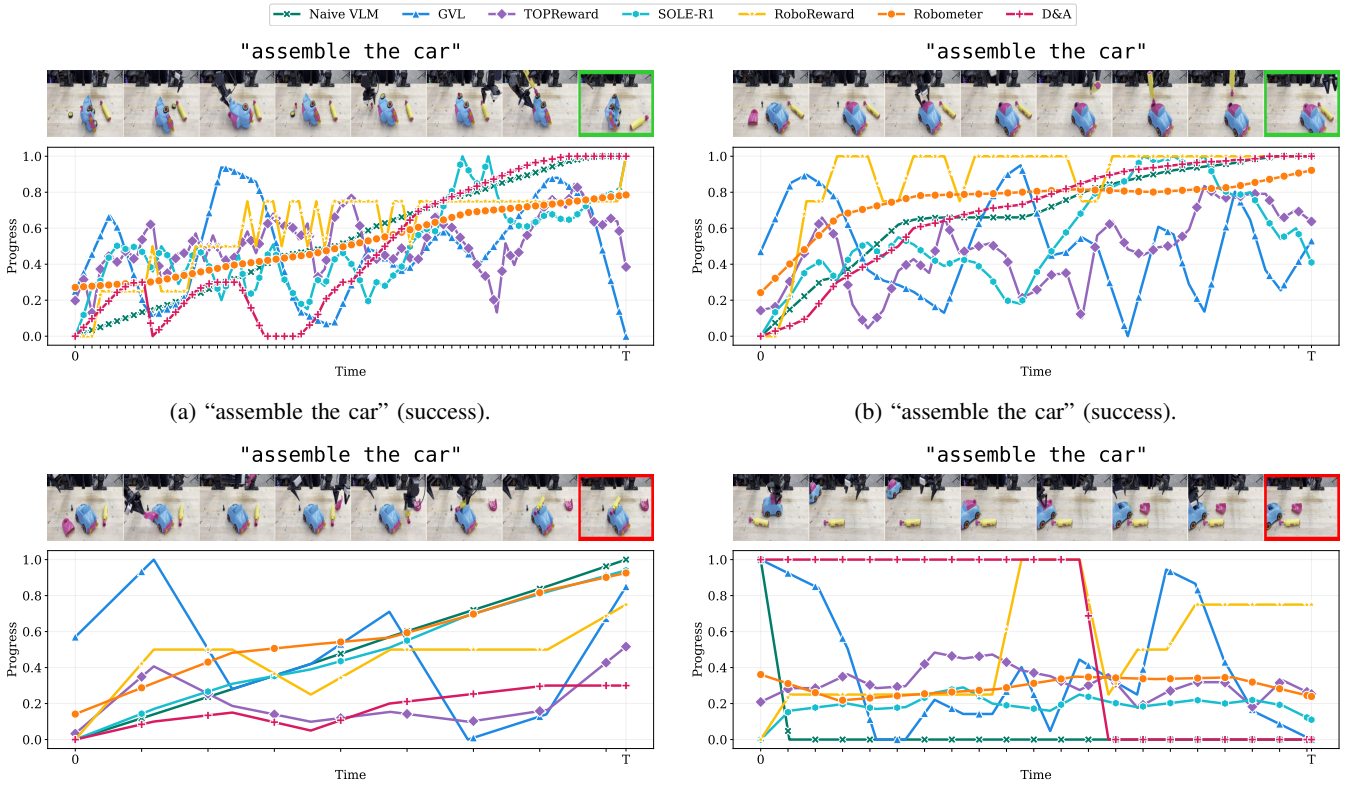
XIII. CaP Experiment Details

We evaluate D&A as a between-attempt feedback source for an open-loop code-as-policies planner on the LIBERO libero_goal suite [50]. The planner (gemini-3.7-flash, reasoning effort MEDIUM) observes the initial scene state and generates a code plan of absolute-waypoint primitives (move_to [x,y,z(,g)], gripper, done, ≤ 40 actions) which is then executed open-loop. Each of the 500 initial conditions (10 tasks, 50 initial states each) is attempted 25 times. If required,

D&A scores each video (gemini-3.7-flash, reasoning effort MEDIUM, `agentview` rendered at 448×448, downsampled to 16 frames), and its output is appended to the planner's accumulated history for the next attempt. Let $s_i(t) \in \{0, 1\}$ denote ground-truth task success of initial condition i on attempt t, for $i = 1, \ldots, N$ ($N = 500$) and $t = 1, \ldots, 25$. We report Success@k, cumulative unique initial conditions solved at attempt k: $M(k) = \frac{1}{N} \sum_{i=1}^{N} \max_{1 \le t \le k} s_i(t)$

XIV. D&A-Bench Benchmark and Qualitative Results

Dataset: 14 real-world manipulation tasks manually collected on a bimanual YAM robot platform. Each task recorded as 2-4 iPhone-captured episodes (2 success, 2 failure; book task only 1 each), resized to 1280x720. Every task has one handwritten instruction plus 9 AI-generated paraphrases (Gemini 3 Flash, shown first frame of episode + original instruction, prompted for M paraphrases preserving meaning but varying phrasing/specificity/vocabulary), giving 10 instructions/task. Full grid: 14 tasks × 4 episodes × 10 instructions = 560 (episode, instruction) pairs scored per reward model. Below shows predictions of all model for the vanilla instruction. While `RBM-OOD-EVAL` subsamples the videos to 8 frames [16], we subsample the videos by a factor of 60 (min 8 frames) to capture visual subtleties in the examples below. We make the dataset publicly available at: https://huggingface.co/datasets/memmelma/dnabench



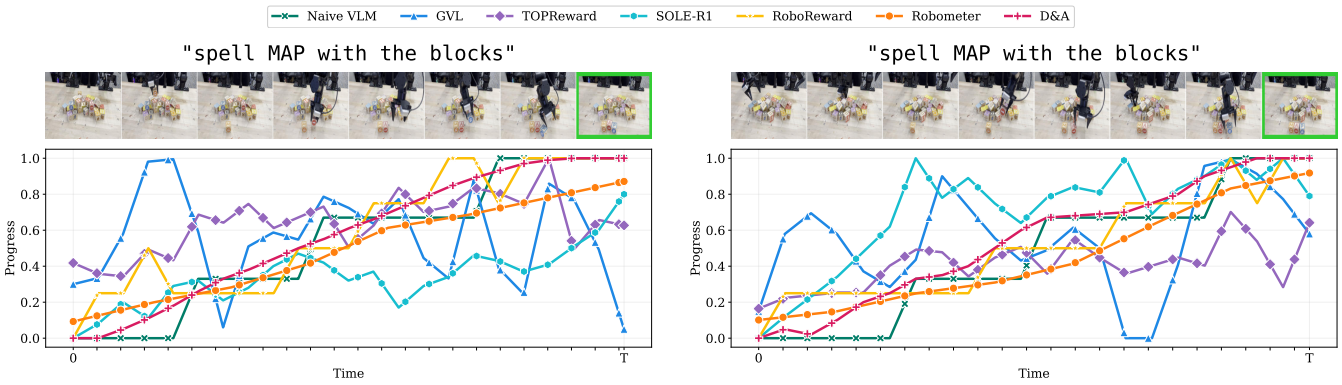
(a) "assemble the car" (success).

(b) "assemble the car" (success).

(c) "assemble the car" (failure).

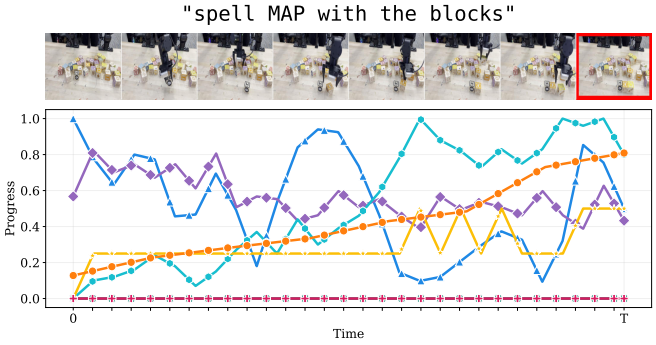
(d) "assemble the car" (failure).

Fig. 15: **Qualitative per-trajectory predictions for "assemble the car".**

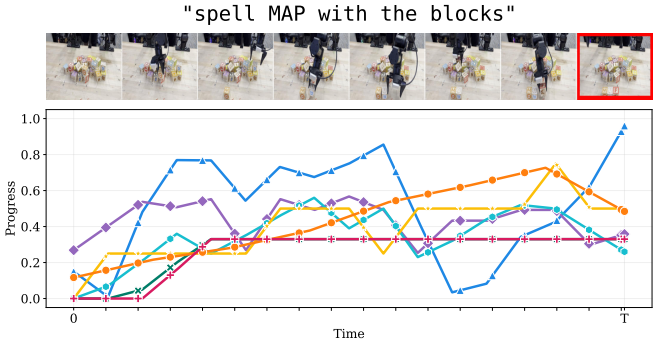


(a) "spell MAP with the blocks" (success).

(b) "spell MAP with the blocks" (success).

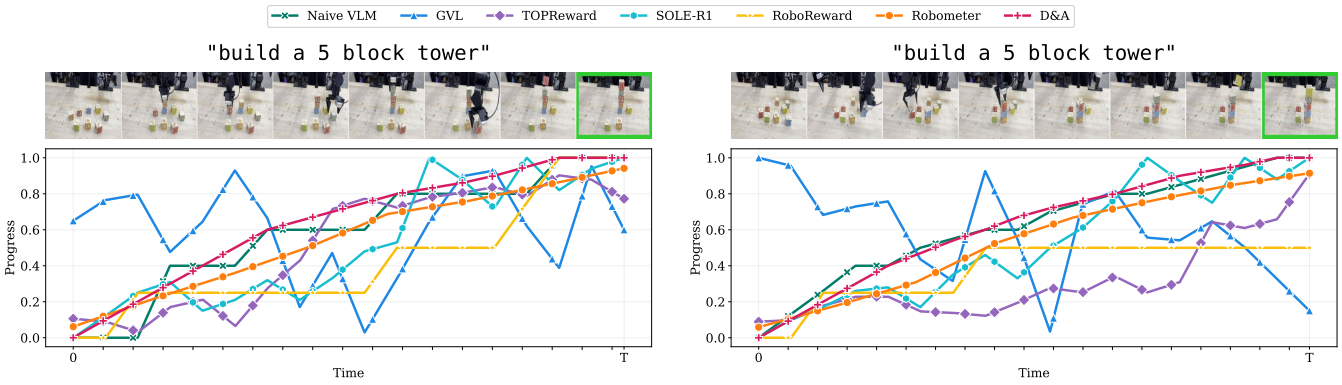


(c) "spell MAP with the blocks" (failure).



(d) "spell MAP with the blocks" (failure).

Fig. 16: **Qualitative per-trajectory predictions for "spell MAP with the blocks".**



(a) "build a 5 block tower" (success).

(b) "build a 5 block tower" (success).

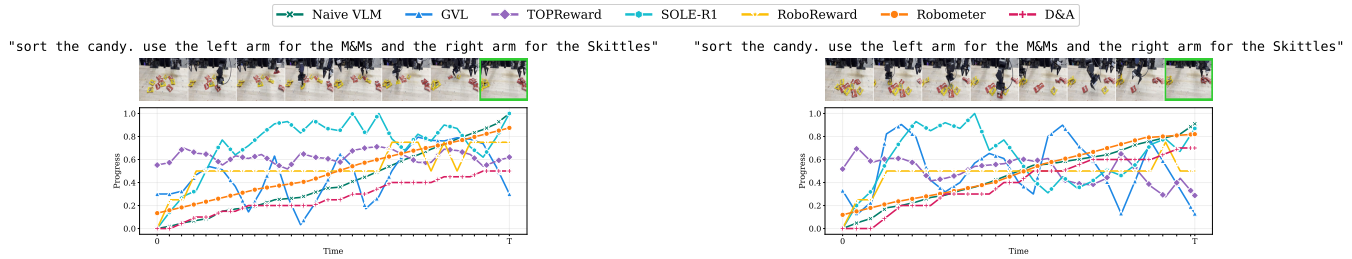


(c) "build a 5 block tower" (failure).



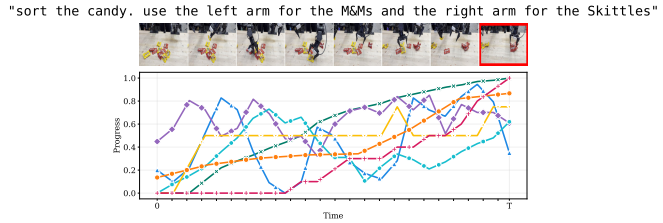
(d) "build a 5 block tower" (failure).

Fig. 17: **Qualitative per-trajectory predictions for "build a 5 block tower".**

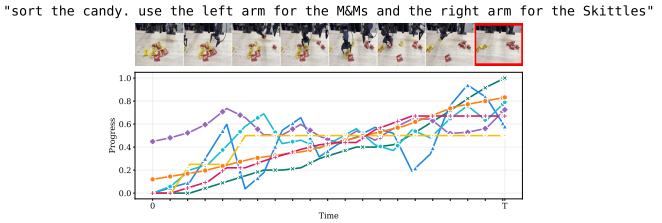


(a) "sort the candy. use the left arm for the M&Ms and the right arm for the Skittles" (success).

(b) "sort the candy. use the left arm for the M&Ms and the right arm for the Skittles" (success).

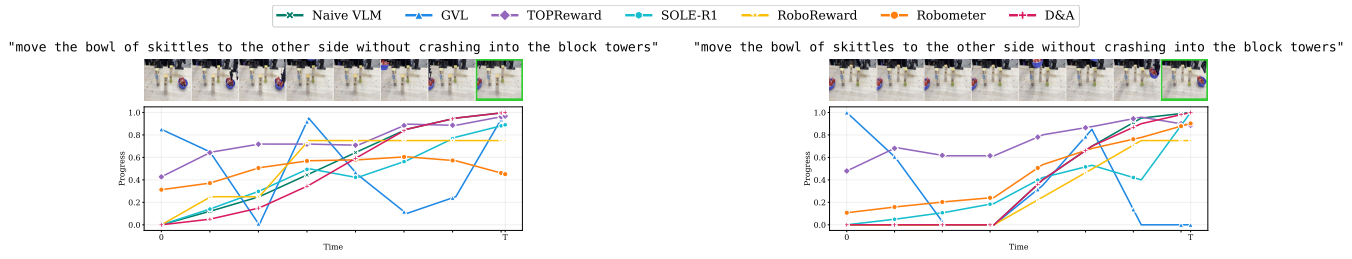


(c) "sort the candy. use the left arm for the M&Ms and the right arm for the Skittles" (failure).



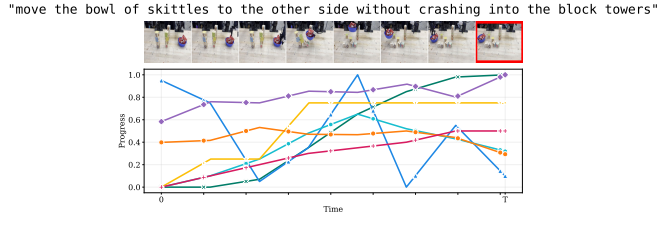
(d) "sort the candy. use the left arm for the M&Ms and the right arm for the Skittles" (failure).

Fig. 18: **Qualitative per-trajectory predictions for "sort the candy".**

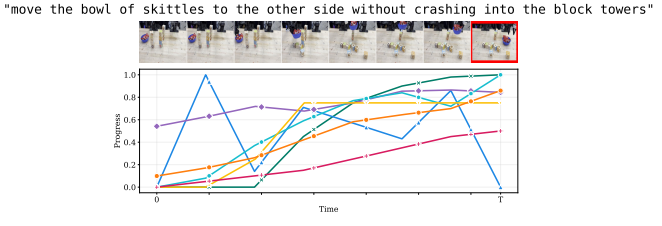


(a) "move the bowl of skittles to the other side without crashing into the block towers" (success).

(b) "move the bowl of skittles to the other side without crashing into the block towers" (success).

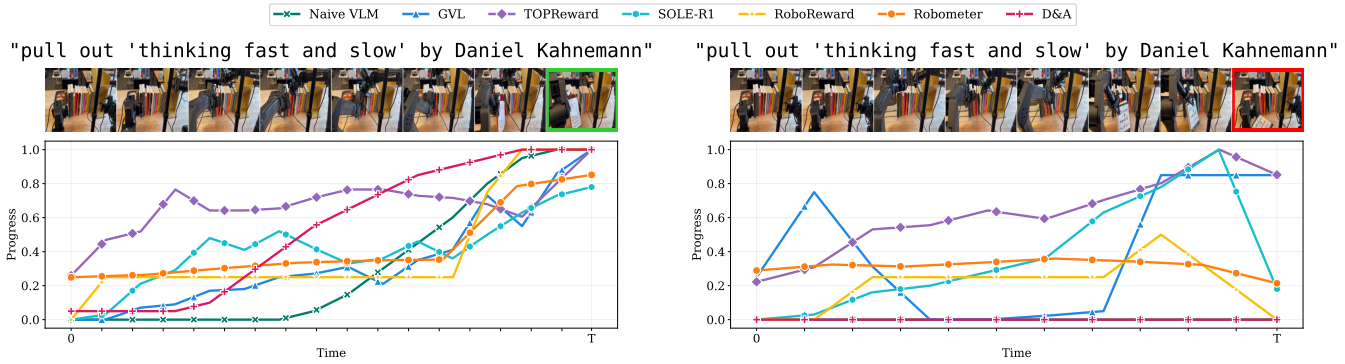


(c) "move the bowl of skittles to the other side without crashing into the block towers" (failure).



(d) "move the bowl of skittles to the other side without crashing into the block towers" (failure).

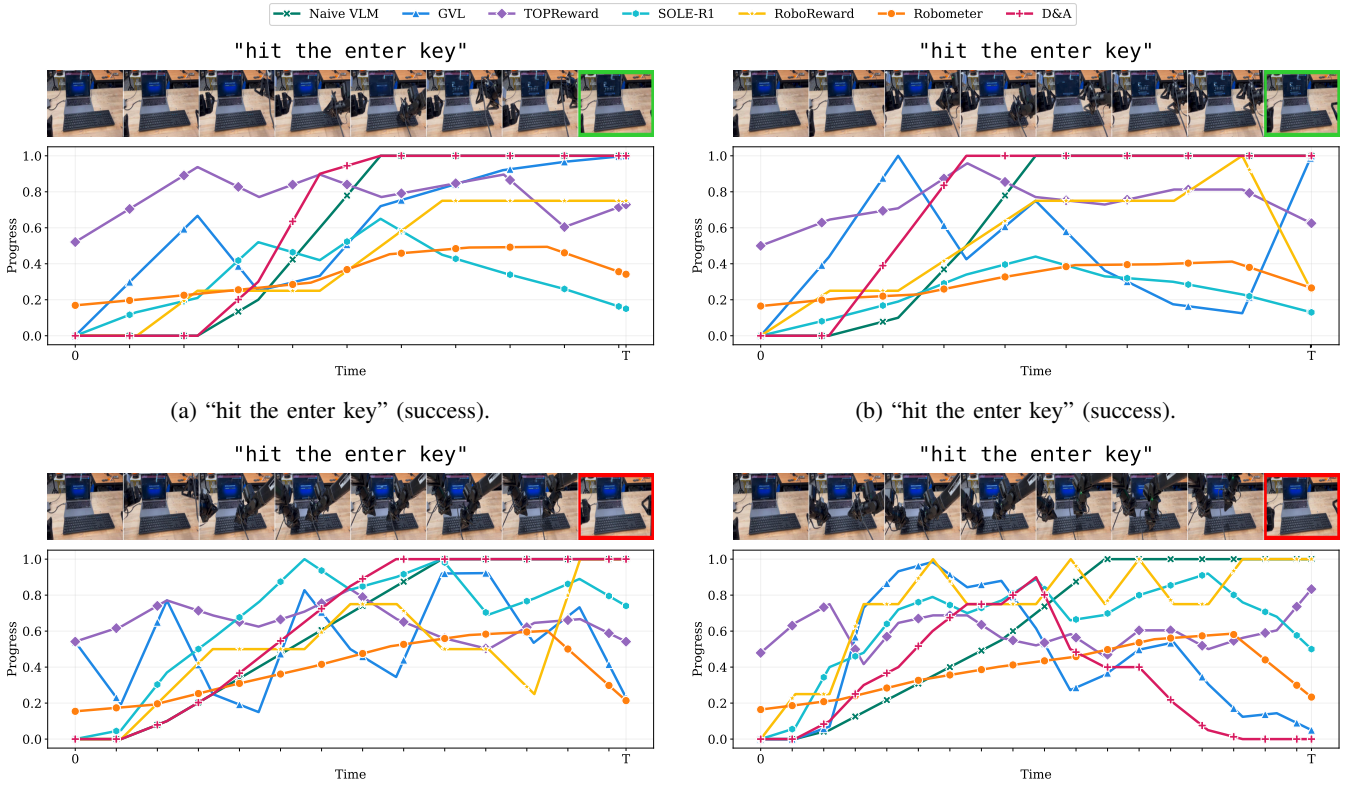
Fig. 19: **Qualitative per-trajectory predictions for "move the bowl of skittles without crashing".**



(a) "pull out 'Thinking, Fast and Slow' by Daniel Kahnemann" (success).

(b) "pull out 'Thinking, Fast and Slow' by Daniel Kahnemann" (failure).

Fig. 20: **Qualitative per-trajectory predictions for "pull out 'Thinking, Fast and Slow'".**



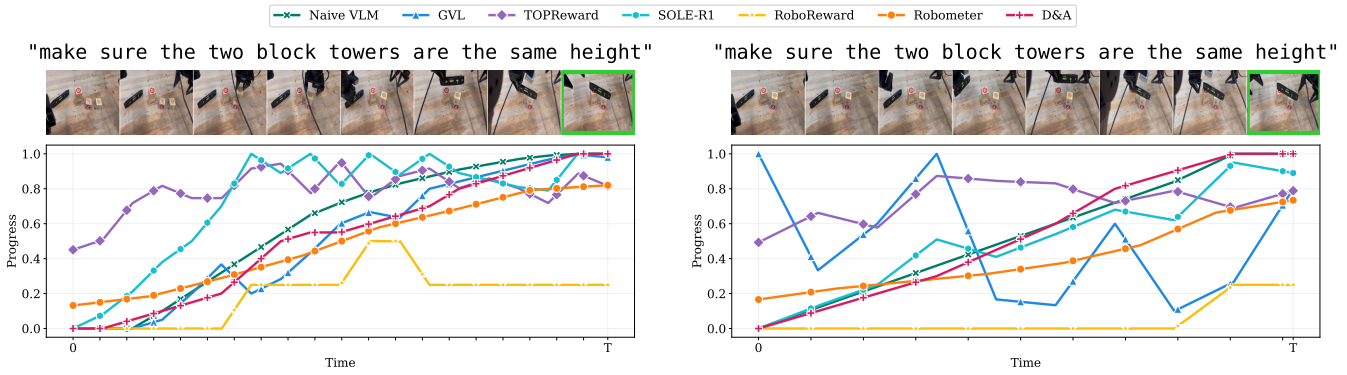
(a) "hit the enter key" (success).

(b) "hit the enter key" (success).

(c) "hit the enter key" (failure).

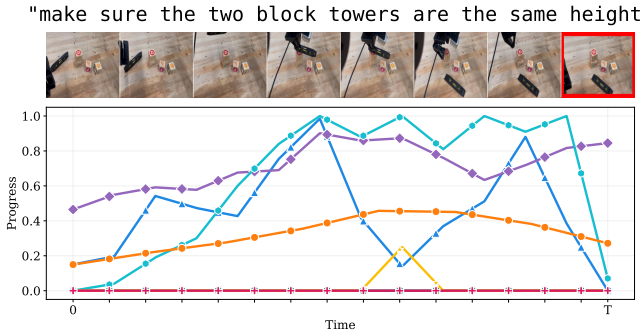
(d) "hit the enter key" (failure).

Fig. 21: **Qualitative per-trajectory predictions for "hit the enter key".**

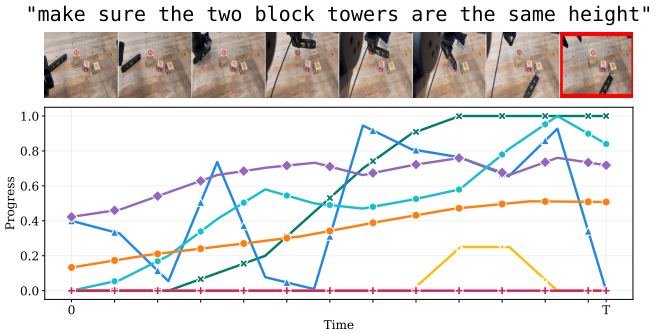


(a) "make sure the two block towers are the same height" (success).

(b) "make sure the two block towers are the same height" (success).

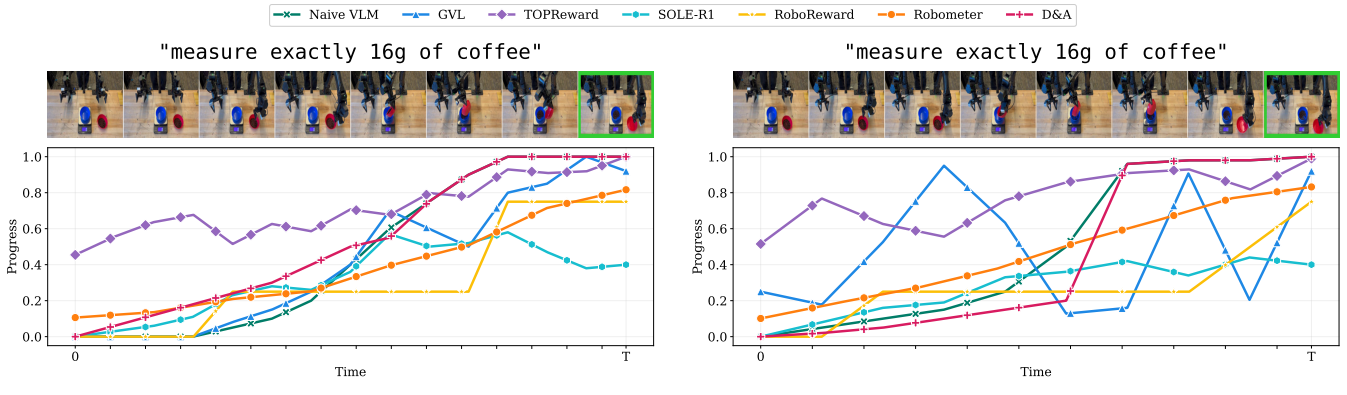


(c) "make sure the two block towers are the same height" (failure).



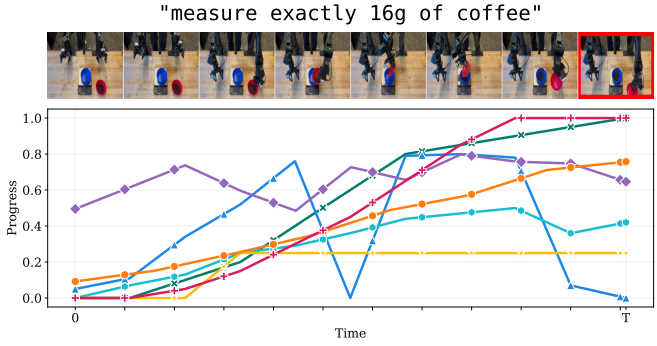
(d) "make sure the two block towers are the same height" (failure).

Fig. 22: **Qualitative per-trajectory predictions for "make sure the two block towers are the same height".**

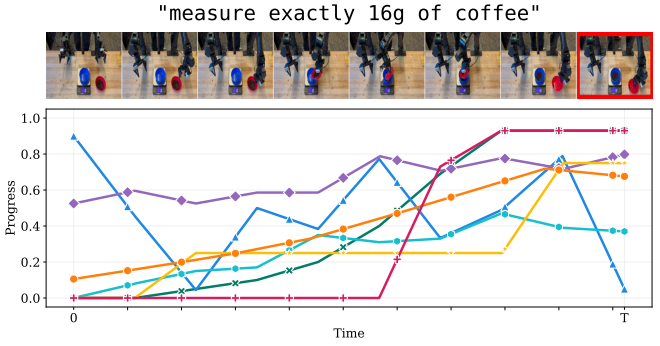


(a) "measure exactly 16g of coffee" (success).

(b) "measure exactly 16g of coffee" (success).

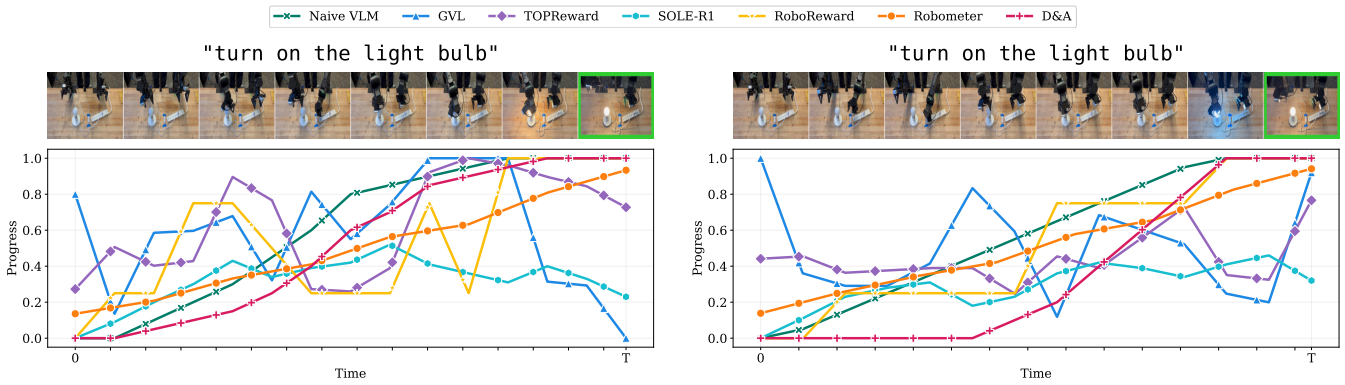


(c) "measure exactly 16g of coffee" (failure).



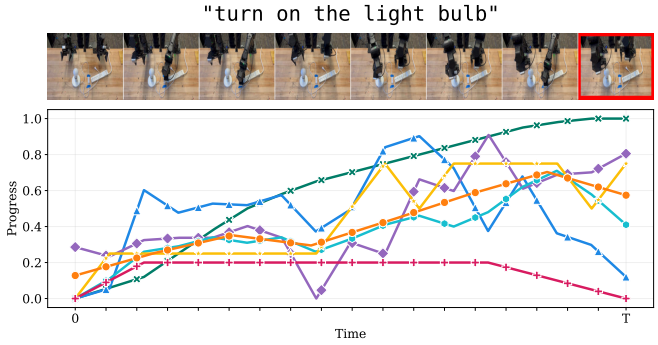
(d) "measure exactly 16g of coffee" (failure).

Fig. 23: **Qualitative per-trajectory predictions for "measure exactly 16g of coffee".**

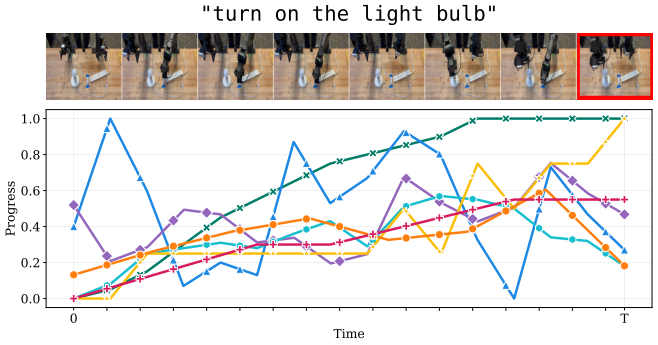


(a) "turn on the light bulb" (success).

(b) "turn on the light bulb" (success).

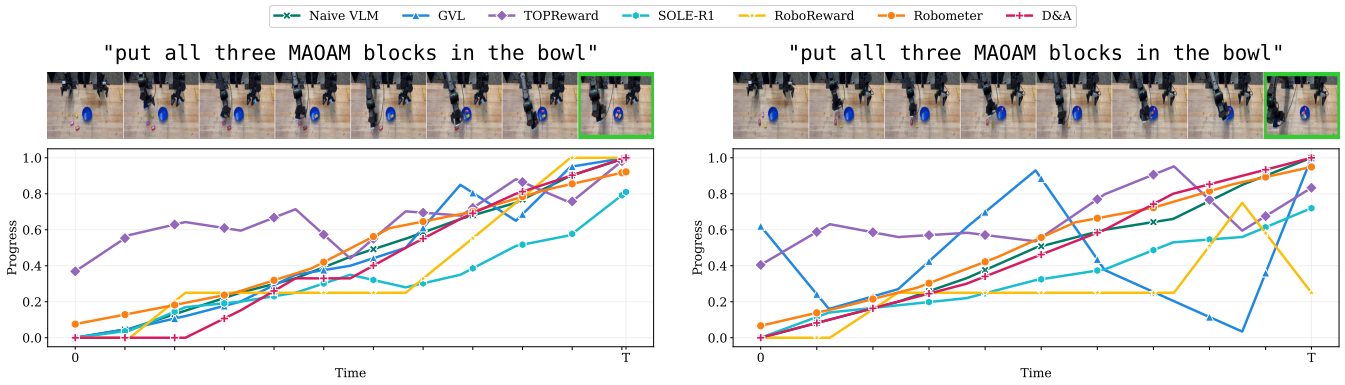


(c) "turn on the light bulb" (failure).



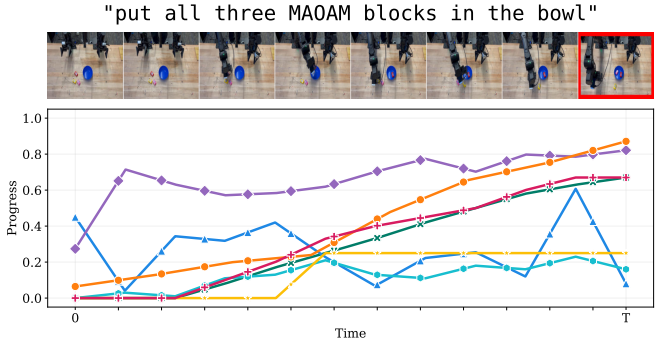
(d) "turn on the light bulb" (failure).

Fig. 24: **Qualitative per-trajectory predictions for "turn on the light bulb".**

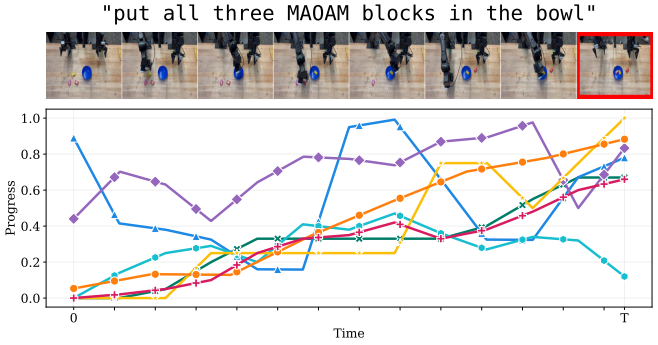


(a) "put all three MAOAM blocks in the bowl" (success).

(b) "put all three MAOAM blocks in the bowl" (success).

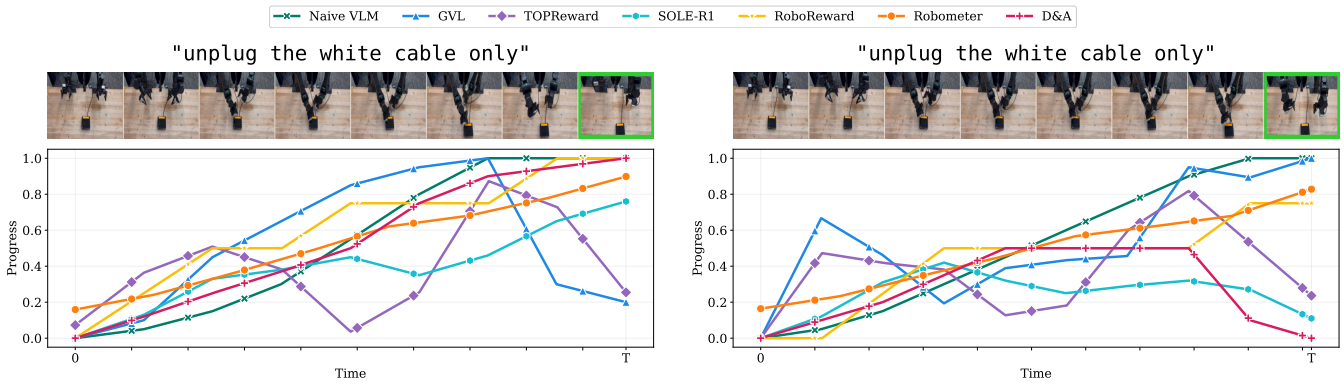


(c) "put all three MAOAM blocks in the bowl" (failure).



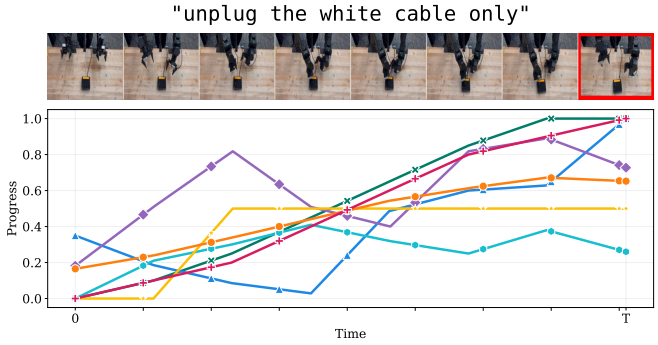
(d) "put all three MAOAM blocks in the bowl" (failure).

Fig. 25: **Qualitative per-trajectory predictions for "put all three MAOAM blocks in the bowl".**

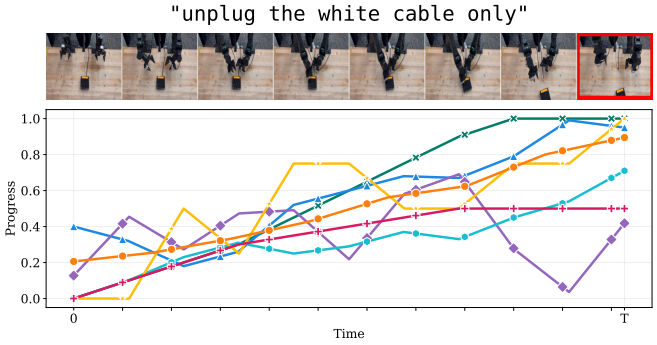


(a) "unplug the white cable only" (success).

(b) "unplug the white cable only" (success).

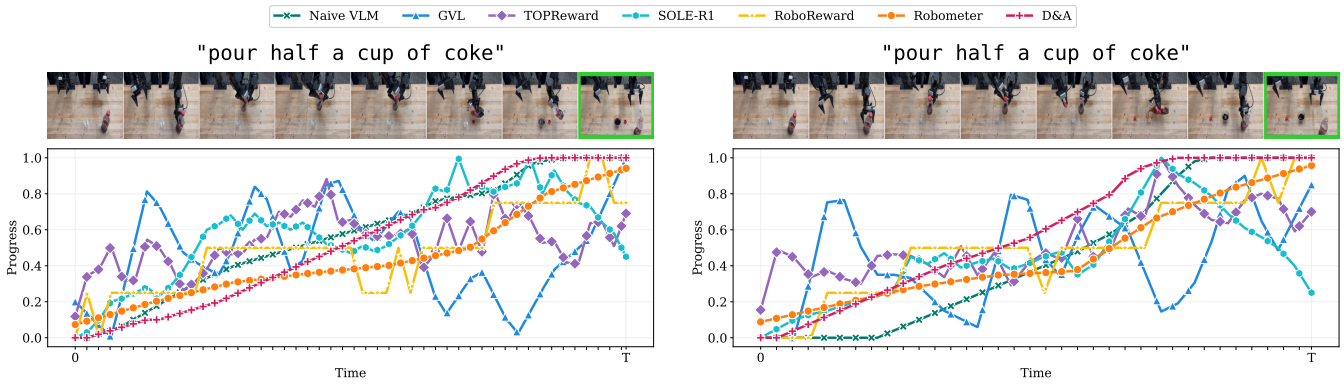


(c) "unplug the white cable only" (failure).



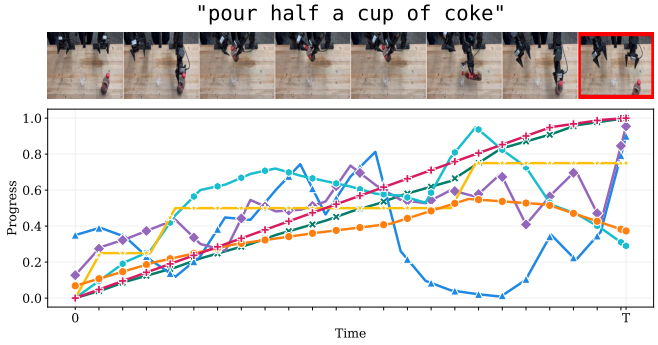
(d) "unplug the white cable only" (failure).

Fig. 26: **Qualitative per-trajectory predictions for "unplug the white cable only".**

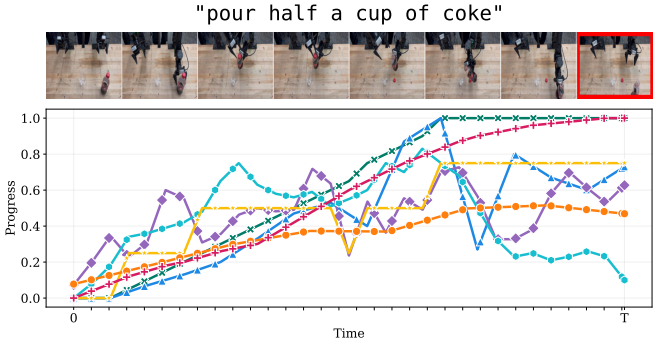


(a) "pour half a cup of coke" (success).

(b) "pour half a cup of coke" (success).

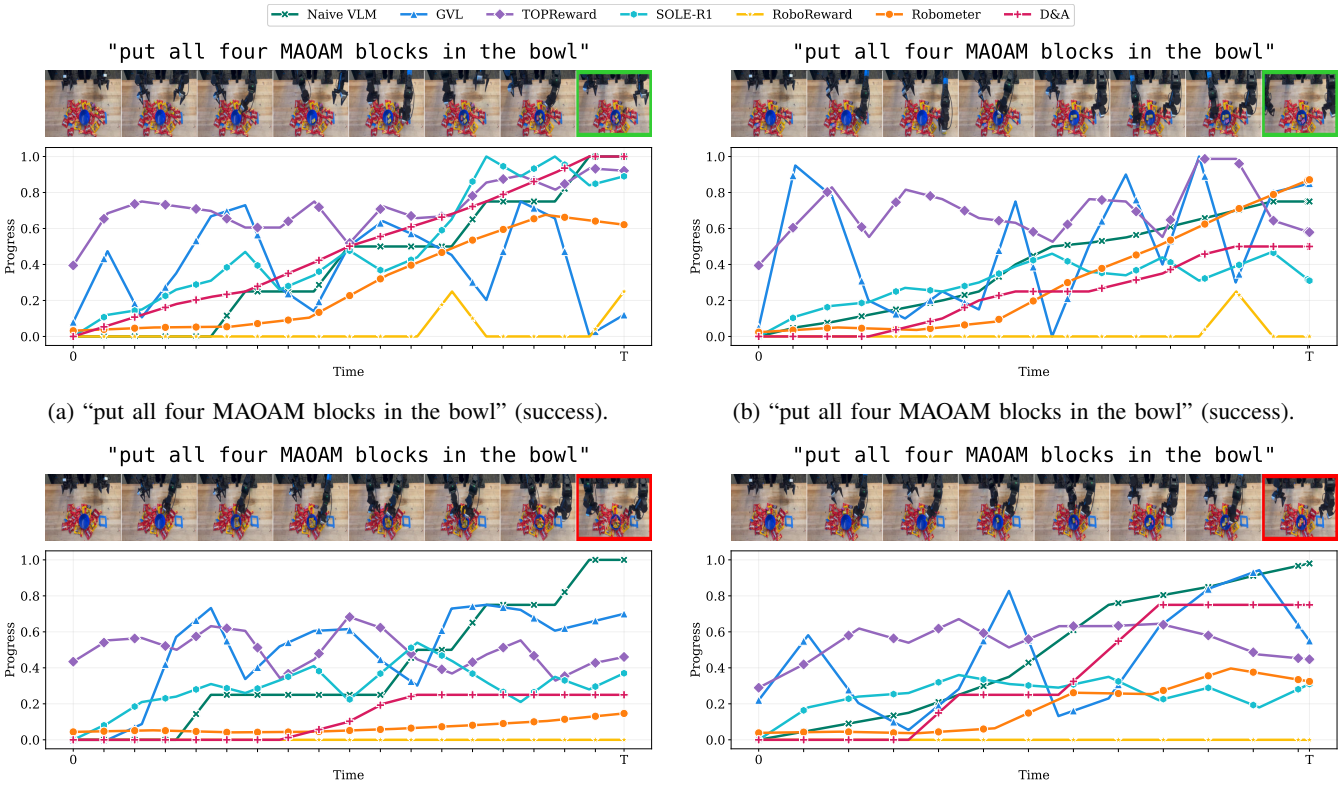


(c) "pour half a cup of coke" (failure).



(d) "pour half a cup of coke" (failure).

Fig. 27: **Qualitative per-trajectory predictions for "pour half a cup of coke".**



(a) "put all four MAOAM blocks in the bowl" (success).

(b) "put all four MAOAM blocks in the bowl" (success).

(c) "put all four MAOAM blocks in the bowl" (failure).

(d) "put all four MAOAM blocks in the bowl" (failure).

Fig. 28: **Qualitative per-trajectory predictions for "put all four MAOAM blocks in the bowl".**

XV. D&A Prompts

A. Naive VLM Baseline

```
Task:
"{{task}}"

1. The task text fixes which objects count (category, color, shape, and wording). The video frames are the only
     ↪ evidence of what is in the environment. Do not "ground" the task by renaming visible objects to match the task.
2. For each object the task requires, it may be tied only to objects visible in the video that refer to that same
     ↪ entity without contradiction. Do not substitute a different object because it could serve the same role. If the
     ↪ visual appearance of an object conflicts with the task's description of that object, or no clear referent is
     ↪ visible, treat that requirement as absent.
3. You may track the same physical object across frames only when it is plainly the same entity already visible in the
     ↪ video--not to map task nouns onto other objects.
4. For each timestep, reason about the % progress towards that fully completed state (0-100%).

Respond with a list of progress values (ordered by frame) in the following JSON format:
[
    {{
        "progress": [int, int, ...]
    }}
]
```

B. D&A

1) Grounding:

```
<role>
You are an expert in object recognition.
</role>

<task>
1. all_objects: list objects that are in the video.
2. are some objects relevant to the following task? Task: "{task}" If yes, list them in task_objects. If no, return an
     ↪ empty list for task_objects.
</task>

<output_format>
Respond with a list of objects in the following JSON format:
[
    {
        "all_objects": [str, str, ...],
        "task_objects": [str, str, ...],
    }
]
</output_format>
```

2) Grounded Video Description:

```
<role>
You are an expert in scene understanding. Provide highly detailed descriptions including the robot and state of the
     ↪ following objects {objects} at each timestep. Describe robot motion and distance between robot and objects.
Without looking at the rest of the video, describe the scene in the first and last frame to ground the descriptions.
</role>

<constraints>
- The video may be captured from either a third-person or wrist-mounted (robot POV) camera.
- Do not make any judgement about what the robot is trying to accomplish. The robot is imperfect and might do things
     ↪ that don't make any sense to you.
- Describe the scene as objectively as possible.
</constraints>

<output_format>
Respond with exactly {N} descriptions, where {N} is the number of video frames. There is only one string per frame. in
     ↪ the following JSON format:
[
    {
        "description_first_frame": str,
        "description_last_frame": str,
        "descriptions": [str, str, ...],
    }
]
</output_format>
```

3) Success Criteria and Assessment:

```
Task:
"{{task}}"

Scene descriptions:
{{description}}
```

```
1. The task text fixes which objects count (category, color, shape, and wording). Scene descriptions are the only
     ↪ evidence of what is in the environment. Do not "ground" the task by renaming scene objects to match the task.
2. For each object the task requires, it may be tied only to scene mentions that refer to that same entity without
     ↪ contradiction. Do not substitute a different object because it could serve the same role. If attributes in the
     ↪ scene conflict with the task's description of that object, or no clear referent exists, treat that requirement
     ↪ as absent.
3. You may align short paraphrases across timesteps only when they plainly denote the same physical object already
     ↪ present in the scene--not to map task nouns onto other objects.
4. If ALL required objects are absent by these rules, the task cannot be completed in this scene: use all 0% progress
     ↪ and explain in "completion state". Otherwise define precisely what the environment would look like if the task
     ↪ were FULLY and COMPLETELY finished (not started, not halfway--entirely done). Reward partial progress if the
     ↪ task is only partially completed and some required objects are present.
5. For each timestep, reason about the % progress towards that fully completed state.
6. Progress is defined as % progress towards the fully completed state. Progress does not have to be monotonic, i.e.,
     ↪ the robot can reverse progress.

Respond with a list of progress values (ordered by frame) in the following JSON format:
[
    {{
        "completion state": str,
        "progress": [int, int, ...]
    }}
]

Before you respond, reflect whether the task is actually fully completed, otherwise adjust the progress values!
```

4) Success Criteria and Assessment - Modified for RoboRewardBenchmark:

```
Task:
"{{task}}"

Scene descriptions:
{{description}}

Given the task instruction and a description of the robot's rollout video (or the final frame observation), assign a
     ↪ discrete progress score reward (1, 2, 3, 4, 5) for the robot's performance. Judge only the final state without
     ↪ considering time limits.

Rubric for End-of-Episode Progress:
* 1 - No Success: The final state shows no goal-relevant change for the command.
* 2 - Minimal Progress: The final state shows a small but insufficient change toward the goal.
* 3 - Partial Completion: The final state shows good progress toward the goal but violates more than one requirement or
     ↪ a major requirement.
* 4 - Near Completion: The final state is correct in region and intent but misses a single minor requirement.
* 5 - Perfect Completion: The final state visibly satisfies all requirements.

Hints & Clarifications for Scoring (Particularly 3 vs. 4):
To correctly distinguish between a Partial Completion (3) and a Near Completion (4), you must differentiate between
     ↪ MAJOR and MINOR task requirements based on the following rules:
* Treat a requirement as MAJOR when the PRIMARY object identity or the PRIMARY spatial relation to the reference object
     ↪ is not satisfied. If a major requirement is missed, the score must be 3 or lower.
* Treat a requirement as MINOR when the PRIMARY object identity and PRIMARY spatial relation to the reference object
     ↪ are satisfied, but an AUXILIARY constraint is missed (e.g., the orientation of the object is slightly off, or
     ↪ the object is placed a bit too far forward/backward but still in the correct general region). Missing a single
     ↪ minor requirement results in a score of 4.

Provide a brief step-by-step reasoning analyzing the observation against the rubric and hints. Conclude with your final
     ↪ discrete score in the following JSON format:
[
    {{
        "score": int
    }}
]
```

5) Success Criteria and Assessment - Modified for Failure Analysis and Feedback Prediction:

```
Task:
"{{task}}"

Scene descriptions:
{{description}}

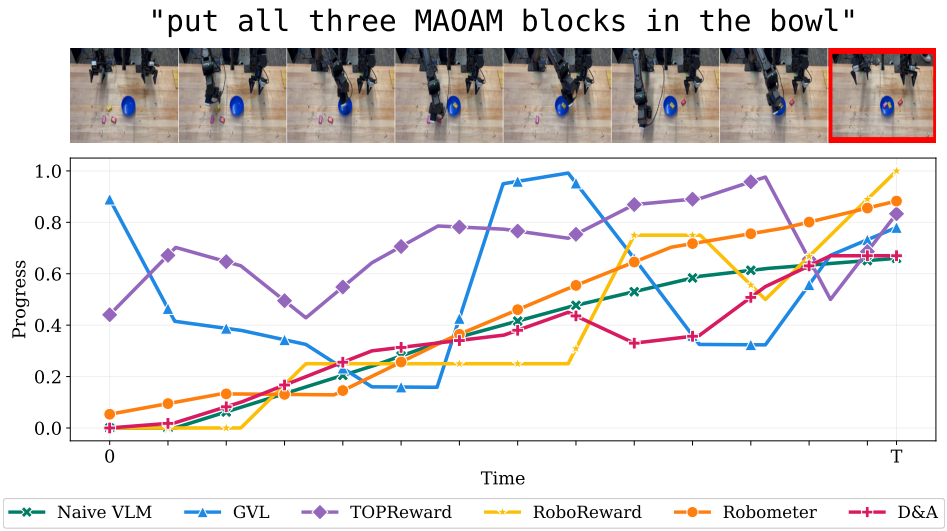
1. The task text fixes which objects count (category, color, shape, and wording). Scene descriptions are the only
     ↪ evidence of what is in the environment. Do not "ground" the task by renaming scene objects to match the task.
2. For each object the task requires, it may be tied only to scene mentions that refer to that same entity without
     ↪ contradiction. Do not substitute a different object because it could serve the same role. If attributes in the
     ↪ scene conflict with the task's description of that object, or no clear referent exists, treat that requirement
     ↪ as absent.
3. You may align short paraphrases across timesteps only when they plainly denote the same physical object already
     ↪ present in the scene-not to map task nouns onto other objects.
```

```
4. If ALL required objects are absent by these rules, the task cannot be completed in this scene: use all 0% progress
     ↪ and explain in "completion state". Otherwise define precisely what the environment would look like if the task
     ↪ were FULLY and COMPLETELY finished (not started, not halfway-entirely done). Reward partial progress if the
     ↪ task is only partially completed and some required objects are present.
5. For each timestep, reason about the % progress towards that fully completed state.
6. Progress is defined as % progress towards the fully completed state. Progress does not have to be monotonic, i.e.,
     ↪ the robot can reverse progress.
7. If the task was not fully completed, describe the failure.
8. If the task was not fully completed, give feedback on what the robot should have done differently.

Respond with a list of progress values (ordered by frame) in the following JSON format:
[
    {{
        "failure": str,
        "feedback": str,
        "completion state": str,
        "progress": [int, int, ...]
    }}
]

Before you respond, reflect whether the task is actually fully completed, otherwise adjust the progress values!
```

XVI. Failure Analysis and Feedback Prediction Examples

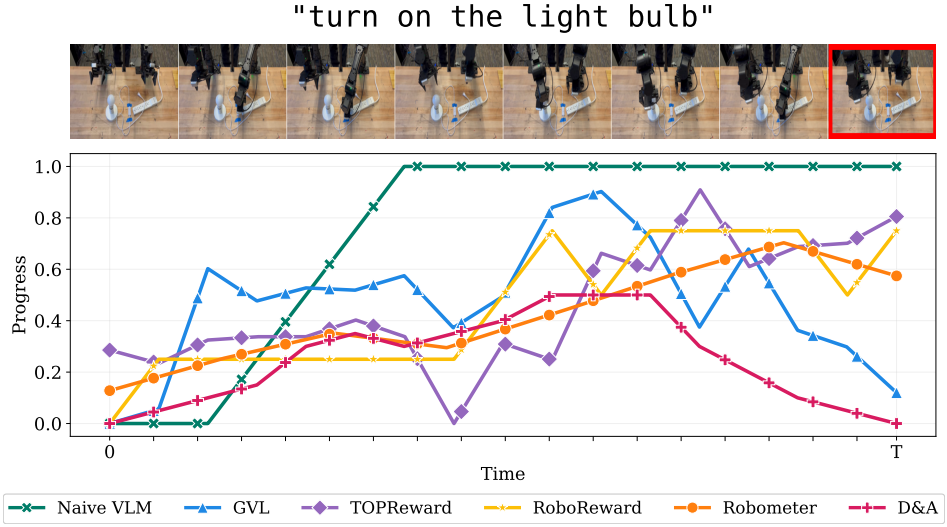


Grounded Video Description:

```
[
    {'description': 'Robotic arms idle over a wooden table featuring a central blue bowl and three candy blocks.'},
    {'description': 'The left robotic arm descends toward the yellow MAOAM block; the right arm remains stationary.'},
    {'description': 'The left arm reaches and grasps the yellow block on the wooden table.'},
    {'description': 'The left robotic arm lifts the yellow block, increasing distance from the table surface.'},
    {'description': 'The left arm transports the yellow block toward the blue bowl.'},
    {'description': 'The left arm releases the yellow block into the blue bowl.'},
    {'description': 'The left robotic arm moves from the bowl toward the red MAOAM block.'},
    {'description': 'The left arm descends and secures the red MAOAM block.'},
    {'description': 'The left arm carries the red block toward the right side of the wooden table.'},
    {'description': 'The red block is dropped onto the table surface near the right robotic arm.'},
    {'description': 'The left arm maneuvers across the table toward the pink MAOAM block.'},
    {'description': 'The left robotic arm grasps and lifts the pink block.'},
    {'description': 'The left arm drops the pink block into the blue bowl and retracts.'}
]
```

Failure Analysis, Feedback Prediction, Success Criteria, and Score:

```
[
  {
    "failure": "The robot only placed the yellow and pink MAOAM blocks into the bowl; the red block was moved to the
    ↪ right side of the table and dropped there instead of being placed in the bowl.",
    "feedback": "To complete the task, the robot should have moved the red block to the blue bowl after picking it up,
    ↪ rather than dropping it on the right side of the table.",
    "completion state": "All three MAOAM blocks (yellow, red, and pink) are contained within the blue bowl on the
    ↪ wooden table.",
    "progress": [0, 0, 0, 0, 0, 33, 33, 33, 33, 33, 33, 33, 67]
  }
]
```

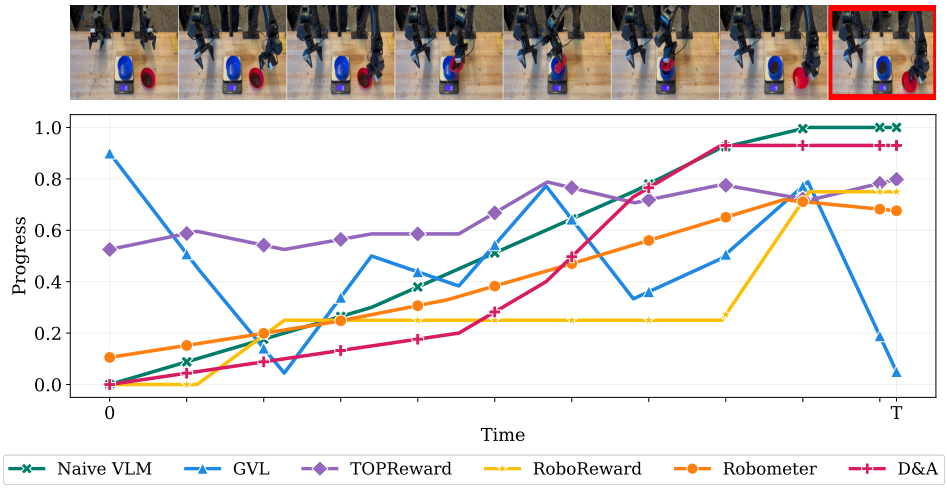


Grounded Video Description:

```
[
    {'description': 'In the initial frame, two robot arms are retracted above a light-colored wooden table. On the
     ↪ table, a white light bulb in a base is on the left, a power strip is on the right, and a white cord with a
     ↪ switch and a plug connects the two. The switch is taped to the table with two strips of blue tape.'},
    {'description': 'The right robot arm begins to lower its gripper towards the white switch located in the middle of
     ↪ the table, while the left robot arm remains stationary in its elevated position.'},
    {'description': "The right robot arm's gripper reaches the switch, closing the distance to just a few centimeters
     ↪ as it prepares to interact with the device."},
    {'description': "The right robot arm's gripper makes contact with the white switch, attempting to manipulate it.
     ↪ The light bulb remains unlit. The left arm stays in its high position."},
    {'description': 'The right robot arm continues to hold its gripper against the white switch, maintaining direct
     ↪ contact with the component secured by blue tape.'},
    {'description': "The right robot arm's gripper remains in contact with the switch, while the left arm is still idle
     ↪ at the top of the frame."},
    {'description': 'The right robot arm is still positioned with its gripper touching the switch, showing minimal
     ↪ movement from the previous frame.'},
    {'description': 'The right robot arm has retracted to its original height, while the left robot arm begins its
     ↪ descent toward the white light bulb on the left side of the table.'},
    {'description': "The left robot arm's gripper is now directly over the white light bulb, roughly 10 centimeters
     ↪ away, while the right arm is fully retracted."},
    {'description': 'The left robot arm lowers its gripper until it makes contact with the top surface of the white
     ↪ light bulb.'},
    {'description': 'The left robot arm maintains its position with the gripper touching the light bulb. The right
     ↪ robot arm remains inactive and elevated.'},
    {'description': 'The left robot arm continues to keep its gripper in contact with the light bulb. The distance
     ↪ between the gripper and the bulb remains at zero.'},
    {'description': 'The left robot arm begins to lift its gripper away from the light bulb, creating a vertical gap of
     ↪ several centimeters.'},
    {'description': 'As the left robot arm retracts further, the right robot arm begins a second descent, moving its
     ↪ gripper back towards the center of the table near the power strip and plug.'},
    {'description': "The right robot arm's gripper is now close to the white plug and power strip on the right side of
     ↪ the wooden table, approximately 5 centimeters away."},
    {'description': 'The right robot arm starts to move upwards again, increasing the distance between its gripper and
     ↪ the power strip on the table surface.'},
    {'description': 'In the final frame, both robot arms have returned to their high, retracted positions. The light
     ↪ bulb, power strip, switch, and cord remain on the table in their original configuration.'}
]
```

Failure Analysis, Feedback Prediction, Success Criteria, and Score:

```
[
    {
        "failure": "The robot failed to turn on the light bulb. It successfully reached the switch and the plug area
     ↪ but did not actuate the switch or complete the electrical connection required to light the bulb, eventually
     ↪ returning to its starting position with the bulb still off.",
        "feedback": "The robot should have applied the necessary force or correct motion to flip the switch to the 'on'
     ↪ position. It also should have ensured the plug was fully inserted into the power strip. Touching the bulb
     ↪ itself was not necessary for this task.",
        "completion state": "The light bulb is illuminated, which requires the power cord to be plugged into the power
     ↪ strip and the switch to be flipped to the 'on' position.",
        "progress": [0, 10, 20, 30, 30, 30, 30, 15, 10, 10, 10, 10, 5, 15, 25, 10, 0]
    }
]
```



Grounded Video Description:

```
[
    {'description': 'Robotic arms hover ~20cm above the table. Blue bowl is on the scale; red bowl with coffee beans is
     ↪ stationary.'},
    {'description': 'The right robotic arm descends toward the red bowl, closing the distance to ~10cm.'},
    {'description': "Right arm gripper reaches the red bowl; left arm remains static. Scale still reads '0.0'."},
    {'description': "Right arm grasps the red bowl's rim. Distance between gripper and bowl is zero."},
    {'description': 'Right arm lifts the red bowl ~15cm and moves it horizontally toward the blue bowl on the scale.'},
    {'description': 'Right arm tilts the red bowl over the blue bowl. Coffee beans begin falling into the blue bowl.'},
    {'description': 'Right arm continues tilting; beans transfer to the blue bowl. The digital scale display
     ↪ fluctuates.'},
    {'description': 'Right arm levels the red bowl and begins moving it back to the right side of the table.'},
    {'description': "Right arm lowers the red bowl onto the wooden table. The scale display stabilizes at '14.8'."},
    {'description': 'Right arm releases the red bowl and retracts upward. The coffee beans remain in the blue bowl on
     ↪ the scale.'}
]
```

Failure Analysis, Feedback Prediction, Success Criteria, and Score:

```
[
  {
    "failure": "The robot stopped pouring coffee beans prematurely, resulting in a final measurement of 14.8g instead
     ↪ of the required 16.0g.",
    "feedback": "The robot should have continued to tilt the red bowl and monitor the digital scale until the weight
     ↪ reached exactly 16.0g before leveling the bowl and moving it away.",
    "completion state": "The blue bowl sits on the digital scale, which displays a measurement of exactly 16.0g of
     ↪ coffee beans.",
    "progress": [
      0,
      5,
      10,
      15,
      25,
      40,
      70,
      92,
      92,
      92
    ]
  }
]
```