SARM2: MULTITASK STAGE AWARE REWARD MODELING FOR SELF IMPROVING ROBOTIC MANIPULATION

Anonymous authors
Paper under double-blind review

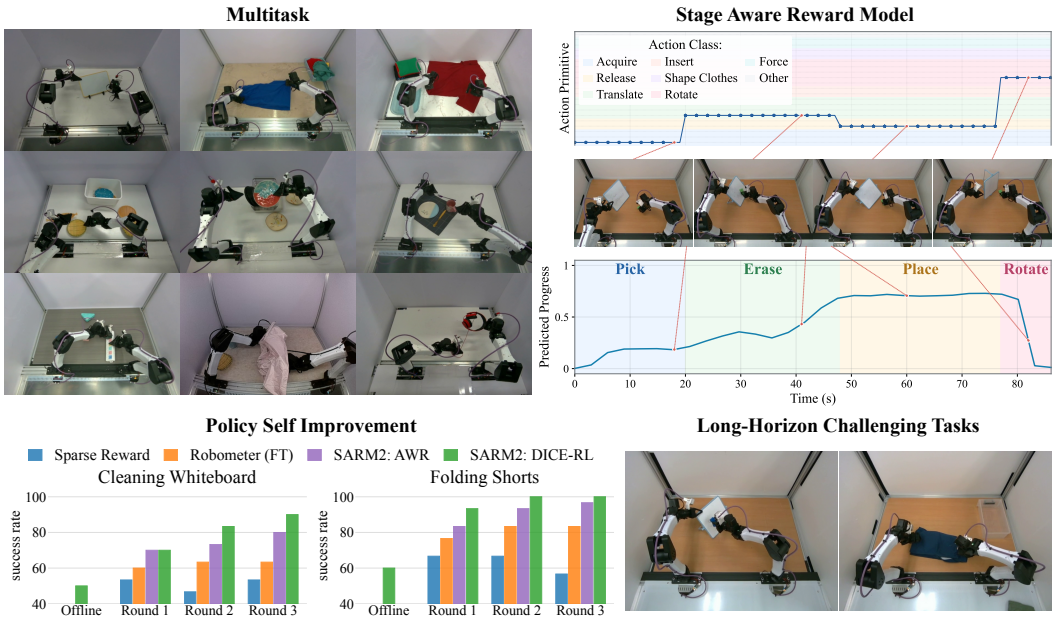


Figure 1: **Overview of SARM2.** SARM2 achieves multitask stage-aware reward modeling by combining a shared action-primitive stage estimator with a multi-gate Mixture-of-Experts value head. We use its dense rewards to drive policy self-improvement on challenging long-horizon tasks, including Cleaning Whiteboard and Folding Shorts.

ABSTRACT

Fine-tuning vision-language-action (VLA) policies for long-horizon manipulation still largely depends on behavior cloning (BC), which requires costly high-quality demonstrations and limits policies to the demonstration distribution. Reward models support learning beyond demonstrations through data reweighting and dense reward signals for reinforcement learning (RL). However, existing reward models struggle to combine multitask scalability and precision: task-specific stage-aware models require per-task annotation and retraining, while general vision-language reward models often miss fine-grained progress in long-horizon tasks. We introduce **SARM2**, a multitask stage-aware reward model combining a shared action-primitive stage estimator with a multi-gate Mixture-of-Experts (MMoE) value head for accurate dense rewards. We use its dense rewards in **SPIRAL**, an offline-to-online pipeline for policy self-improvement from demonstrations and autonomous rollouts. On ten tasks, SARM2 reduces overall value-estimation MSE by 33.3% versus multitask SARM and 53.5% versus the best evaluated general-purpose reward model baseline. Starting from offline RL policies, SPIRAL raises success rates from 60% to 100% on Folding Flattened Shorts, 33.3% to 66.7% on Folding Crumpled Shorts, and 50% to 90.0% on Cleaning Whiteboard, supporting a robot data flywheel driven by accurate dense rewards.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Vision-Language-Action (VLA) models (Zitkovich et al., 2023; Kim et al., 2024; Team et al., 2024; Black et al., 2024; Intelligence et al., 2025) have established end-to-end policy learning as a popular paradigm in robotic manipulation. Yet behavior cloning (BC) for long-horizon tasks requires extensive in-domain demonstrations and leaves policies prone to out-of-distribution (OOD) failures. These limitations motivate reinforcement learning (RL) adaptation that allows VLAs to improve from their own deployment experience beyond the human demonstrations.

A direct approach to RL adaptation uses online interaction data with sparse rewards (Guo et al., 2025; Sun & Song, 2026; Lei et al., 2025), but can struggle on long-horizon tasks. Intervention-based methods such as $\pi^*_{0.6}$ (Physical Intelligence et al., 2025) and ConRFT (Chen et al., 2025c) continually refine the policy via on-policy DAgger, but require costly human-in-the-loop supervision from operators proficient in both teleoperation and policy training. Progress reward models offer a different path with much less human effort. Their finer-grained reward signal supports offline data filtering (Chen et al., 2025b) as well as online policy self-improvement on some simple tasks (Liang et al., 2026; Tan et al., 2025). Despite these advances, reliably guiding policy self-improvement on challenging long-horizon tasks remains difficult: progress reward models must provide accurate local feedback through subtle stage transitions, execution errors, and recovery while handling the distribution shift from demonstrations to policy-generated rollouts.

Existing progress reward models have advanced both value estimation and task coverage. Task-specific methods (Ma et al., 2023; Alakuijala et al., 2024; Hung et al., 2024; Kim et al., 2025; Zhang et al., 2025a; Chen et al., 2025b; Yu et al., 2026) learn to estimate task progress from demonstrations, using normalized elapsed time as a proxy for progress. Among them, SARM introduces a stage-conditioned hierarchy that substantially improves long-horizon accuracy. In parallel, reward models based on multimodal large language models (MLLMs) (Ma et al., 2024; Tan et al., 2025; Chen et al., 2026; Lee et al., 2026; Liang et al., 2026) draw on broad pretrained priors to generalize across tasks. These advances highlight the value of stage structure and broad task coverage, but leave two limitations for long-horizon policy self-improvement:

(1) ***Limited scalability***: SARM relies on task-specific stage definitions and annotations, requiring its stage estimator to be retrained for each new task. This dependence makes extending stage-aware reward modeling to diverse tasks costly.
(2) ***Precision loss in general reward models***: MLLM-based reward models often provide coarse and noisy evaluations at the action-chunk level, making subtle progress, adjustments, and errors difficult to distinguish. This ambiguity undermines the accurate dense rewards required for long-horizon self-improvement.

Motivated by these limitations, we propose **SARM2**, a stage-aware reward model that scales to multiple tasks while maintaining high accuracy in fine-grained progress estimation. SARM2 combines a reusable stage estimator with a multitask value head. To make stage estimation reusable, SARM2 uses a shared action-primitive set and a unified stage estimator across tasks. To obtain precise local feedback, the predicted primitive selects a gate in a multi-gate Mixture-of-Experts (MMoE) value head, routing observations to specialized experts to achieve accurate reward estimation across multiple tasks. To translate SARM2's dense reward estimates into stable policy self-improvement, we further introduce **SPIRAL**, an offline-to-online pipeline that improves policies from demonstrations and then their own rollouts. Its lightweight one-time reward model calibration addresses the demonstration-to-rollout distribution shift, while replaceable RL algorithm modules drive policy updates. Our contributions are summarized as follows:

- We propose **SARM2**, a multitask stage-aware reward model that provides dense, accurate reward signals across diverse long-horizon manipulation tasks.
- We introduce **SPIRAL** to utilize SARM2's dense reward signals into stable downstream policy improvement. Its offline-to-online process supports learning from policy deployment experience and drives a robotics data flywheel.
- We extensively evaluate SARM2 on a ten-task benchmark spanning diverse bimanual manipulation tasks, where it outperforms previous multitask reward model baselines. We further instantiate SARM2 within SPIRAL on two challenging long-horizon real robot tasks, demonstrating that its dense rewards enable consistent policy self-improvement.

2 Related Works

Reward Models for Robotic Manipulation. Task-specific reward models learn from visual observations using mid-sized vision-language encoders such as CLIP (Radford et al., 2021) or SigLIP (Tschannen et al., 2025) (Ma et al., 2023; Alakuijala et al., 2024; Hung et al., 2024; Kim et al., 2025; Zhang et al., 2025a; Physical Intelligence et al., 2025). To handle long-horizon, contact-rich tasks, *stage-aware* variants (Mu et al., 2024; Kim et al., 2025; Chen et al., 2025b) decompose a task into sub-stages and produce stage-conditioned dense rewards. These methods are fundamentally single-task: each new task requires retraining, and SARM (Chen et al., 2025b) further requires dense per-frame stage annotations. We compare against ReWiND (Zhang et al., 2025a), the strongest single-task baseline without dense annotation. A complementary line builds task-agnostic reward models on top of pretrained VLMs (Ma et al., 2024; Lee et al., 2026; Tan et al., 2025; Chen et al., 2026; Liang et al., 2026); their out-of-the-box rewards are too coarse for dense per-step signals, and fine-tuning their large parameter count is expensive. We benchmark against TOPReward (Chen et al., 2026) and Robometer (Liang et al., 2026) in Section 4.2.

Real Robot Reinforcement Learning. RL on real robot data is hindered by two challenges: obtaining effective reward signals on long-horizon tasks, and backpropagation through iterative diffusion/flow sampling. End-to-end RL trains compact single-task policies from scratch (Huang et al., 2024; Luo et al., 2024; 2025; Zhao et al., 2025; Kalashnikov et al., 2018; Seo et al., 2024; Wu et al., 2022), sacrificing expressiveness. Offline-to-online methods initialize via BC then refine with RL: simple MLP heads (Hu et al., 2023; Yang et al., 2024; Chen et al., 2025c) are easy to fine-tune but limited; AWR-style updates on expressive policies (Wu et al., 2025; Physical Intelligence et al., 2025; Peng et al., 2019) depend on human DAgger interventions. DPPO (Ren et al., 2024), ReinFlow (Zhang et al., 2025b), FPO++ (Yi et al., 2026), and RL-100 (Lei et al., 2025) directly optimize iterative diffusion or flow policies with policy-gradient-style updates. Steering (Wagenmaker et al., 2025; Niu et al., 2024) and residual (Xiao et al., 2025; Ankile et al., 2025) methods instead modify base-policy outputs without updating all base-policy weights. DICE-RL (Sun & Song, 2026) unifies value-guided action selection with residual learning and serves as one of our inherited policy optimizers. We use DICE-RL and AWR as two policy-optimization backends while holding the SPIRAL offline-to-online data pipeline fixed. MoE in robotics has been explored for policy capacity scaling (Huang et al., 2024; Hao et al., 2026; Cheng et al., 2025; Zhang et al., 2026; Shen et al., 2025; Du et al., 2025; Yang et al., 2025; Huang et al., 2025; Seyde et al., 2022; Chen et al., 2025a), but to our knowledge **SARM2** is the first to apply MoE to a multitask robotics reward model.

3 Method

Task definition. Given a task instruction ℓ and an observation sequence $\mathbf{o}_t$ containing multi-view images and robot proprioceptive states, a progress reward model predicts a scalar measuring how far the robot has advanced toward completing the task. We train SARM2 with an MSE loss to predict negative normalized steps-to-go, $\hat{V}_t \in [-1, 0]$. For policy learning, we express this value as progress $\bar{V}_t = 1 + \hat{V}_t \in [0, 1]$, with 0 denoting the task start and 1 completion. For an action chunk $\mathbf{a}_t = (a_t, \ldots, a_{t+h-1})$ spanning h control steps, we define its dense reward as the change in predicted progress:

$$r_{t,h}^{\text{dense}} = \bar{V}_{t+h} - \bar{V}_t = \hat{V}_{t+h} - \hat{V}_t. \tag{1}$$

Progress therefore earns positive reward, little change yields near-zero reward, and regression incurs negative reward, providing local credit for policy learning. For a chunk truncated by episode termination at T, we use its executed length $\min(h, T - t)$ in place of h.

3.1 SARM2: Multitask Stage-Aware Reward Model

3.1.1 Action-Primitive Stage Estimator

Robot manipulation trajectories can be divided into stages defined by *action primitives*, such as picking, placing, and rotating. We use these reusable primitives as a task-agnostic representation for stage estimation. The primitive estimator is shared across tasks, while the value head is jointly trained on in-domain demonstrations. Two observations from an internal corpus of $10{,}000+$ hours of annotated manipulation data motivate this design: (1) demonstrations across nearly all tasks decompose into sequences of action primitives that recur across tasks, despite differences in task names, scenes, and objects; (2) the primitive vocabulary is comparatively small, and many primitives are

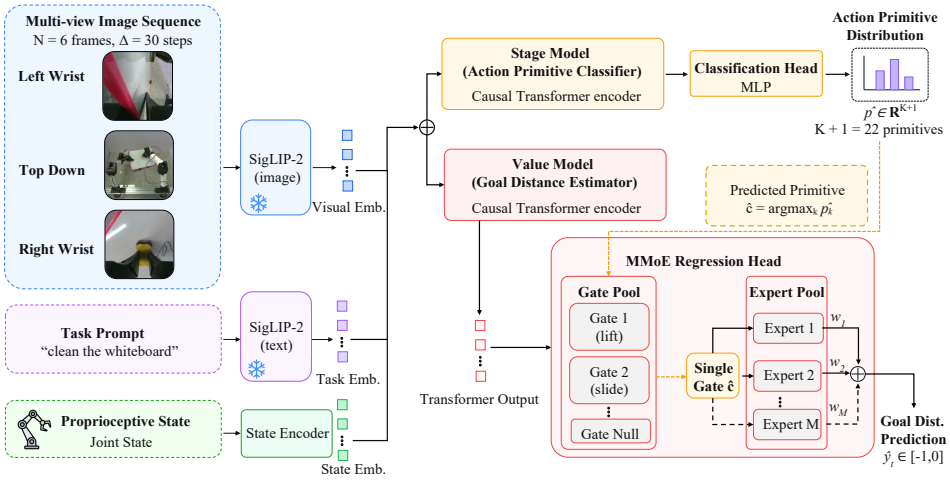


Figure 2: **Overview of SARM2.** Three camera views plus proprioceptive state are encoded by a shared frozen SigLIP-2 backbone, whose cached frame embeddings feed two *separately trained* causal Transformers: (i) a task-agnostic *stage estimator* that classifies the current segment over $K{+}1{=}22$ candidates ($K{=}21$ action primitives and a null class used as a fallback when the model is uncertain), and (ii) a *multi-gate MoE value decoder* whose gate is selected by the predicted primitive group and routes the fused token through top-k shared experts to produce a dense progress estimate.

either semantically similar (e.g., `pour`/`dump`) or form dual pairs rarely co-occurring within a task (e.g., `pull`/`push`), allowing further compression.

Action Primitive Dataset Construction. We curated 200 hours of real-world manipulation data spanning 100 tasks. After consolidation we identified $K = 21$ action primitives that cover $> 90\%$ of total task duration. We partitioned the source dataset along annotation boundaries and constructed a balanced dataset $\mathcal{D}_{\text{AP}}$ with (i) each primitive contributing $t_k = 3$ hours, and (ii) maximum source-task diversity per primitive, yielding $(K + 1) \cdot t_k = 66$ hours of segments. The additional null class is formed from all remaining long-tail samples, including reset behaviors and action primitives with low occurrence frequency. Construction details are in Appendix 6.4.

Model Architecture. We frame stage estimation as temporal classification over the primitive vocabulary plus a null class ($K + 1 = 22$ classes). At time t, the model receives $N = 6$ recent frames sampled at stride $\Delta = 30$ (≈ 6 s of context at 30 Hz). We use three views $v \in \{\text{wrist}_L, \text{wrist}_R, \text{top}\}$, denoted $\mathbf{I}_t^{(v)}$, plus proprioceptive state $\mathbf{s}_t$. Each frame is encoded by a frozen SigLIP-2 (Tschannen et al., 2025) encoder ϕ; $\mathbf{s}_t$ is projected by an MLP ψ. All tokens are processed by a 4-layer causal Transformer f_θ followed by a linear head g_θ producing class logits $\mathbf{p}_t = \text{softmax}(g_\theta(\mathbf{h}_t))$. The model is trained with cross-entropy on $\mathcal{D}_{\text{AP}}$; full details on hyperparameters are in Appendix 6.5.

3.1.2 MMoE Value Decoder

Building on the stage estimator, we design a multitask value model that estimates dense, stage-conditioned goal-distance. Different primitives exhibit markedly different visual dynamics and progress signatures, so a monolithic value head suffers from interference and tail underfitting. We address this with a *multi-gate Mixture-of-Experts (MMoE)* decoder that specializes along primitive groups while sharing a common expert pool.

The value model uses its own 6-layer causal Transformer backbone, trained separately from the stage estimator's 4-layer Transformer; the two share only the frozen SigLIP-2 encoder and its cached six-frame embeddings, which eliminates redundant visual encoding at inference. At step t, the value model receives: (1) three-camera frames; (2) proprioceptive state $\mathbf{s}_t$; (3) task-name text embedding $\mathbf{e}_{\text{task}}$; and (4) the predicted-primitive embedding $\mathbf{e}_{\text{prim}}$ of $\tilde{y}_t$ from the stage estimator. The input frames concatenates with the following sequence: the first episode frame (visual anchor), the six recent frames at stride $\Delta = 30$, and up to three rewinding frames as implicit negatives (Zhang et al., 2025a) that prevent the model from collapsing to a monotonic time-index predictor.

We partition the $K + 1$ primitives into $M + 1$ semantic groups by shared visual/motion patterns (see Appendix 6.4). Following MMoE (Ma et al., 2018; Lei et al., 2024), each group has a dedicated gate

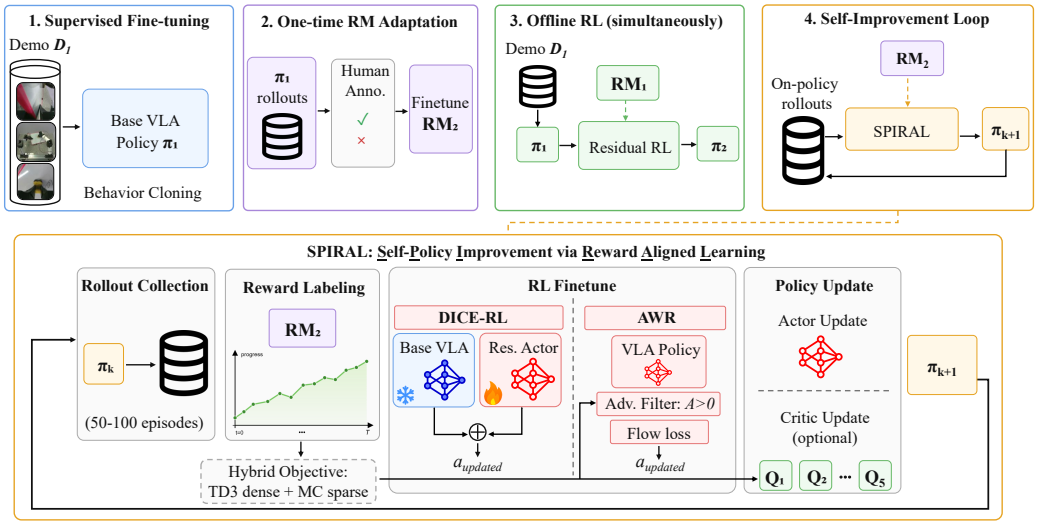


Figure 3: **SARM2-guided policy improvement with SPIRAL.** A standard behavior-cloning initialization produces the task policy and initial rollouts. A lightweight, one-time calibration aligns SARM2 to policy rollout states using coarse human labeling. In parallel, offline RL uses dense rewards from RM_1 to improve the prior policy and make fuller use of the demonstrations. The calibrated reward model is then frozen and supplies dense reward signals for iterative rollout relabeling, with replaceable RL algorithms performing the policy optimization.

G_m over a shared pool of E experts (each a 3-layer MLP). The active gate at step t is selected by $m(\tilde{y}_t)$, producing top-k routing weights and MoE output

$$\mathbf{o}_t = \sum_{e=1}^{E} g_{t,e}^{(m(\tilde{y}_t))} \cdot \mathcal{E}_e(\mathbf{h}_t), \quad \mathbf{g}_t^{(m)} = \mathrm{TopK}(\mathrm{softmax}(W_m \mathbf{h}_t)). \tag{2}$$

Group-conditioned routing provides an explicit inductive bias: primitives from different groups use separate gates, while shared experts remain accessible. To prevent router collapse, we add per-gate balance and entropy auxiliary losses ($\mathcal{L}_{\text{balance}}$, $\mathcal{L}_{\text{entropy}}$); full formulae and ablations are in Appendices 6.8 and 6.10, full details on hyperparameters are in Appendix 6.5.

Following the remaining-steps formulation of $\pi^*_{0.6}$ Physical Intelligence et al. (2025), the value model predicts negative normalized *remaining steps to completion*, with target $r_t^\star = -(T-t)/T \in [-1, 0]$, which we find easier to fine-tune on rollouts with variable horizons. Normalizing by trajectory length places demonstrations of different durations on a common endpoint scale. By incorporating action-primitive information into value prediction and expert selection, SARM2 improves overall value estimation accuracy. The full objective combines an MSE term with the auxiliary losses:

$$\mathcal{L} = \mathbb{E}_{\mathcal{D}}[(\hat{r}_t - r_t^\star)^2] + \lambda_{\text{bal}} \mathcal{L}_{\text{balance}} + \lambda_{\text{ent}} \mathcal{L}_{\text{entropy}}. \tag{3}$$

3.2 SARM2-Guided Policy Improvement with SPIRAL

To support stable policy improvement with SARM2's dense rewards, we introduce **SPIRAL** (*Self-Policy Improvement via Reward-Aligned Learning*), an offline-to-online pipeline. One-time calibration adapts SARM2 to rollout states, while offline warm-up on demonstrations strengthens the policy before online self-improvement. The policy then improves from the latest round's rollouts using replaceable RL modules, instantiated here with DICE-RL and AWR.

3.2.1 SPIRAL Offline-to-Online Process

(1) Policy initialization. We behavior-clone a pretrained VLA on task demonstrations $\mathcal{D}_{\text{demo}}$ to obtain π_1 and collect 50–100 initial rollouts $\mathcal{R}_1$. **(2) One-time calibration.** Human annotators coarsely annotate $\mathcal{R}_1$ at segment level. Then we briefly finetune RM_1 into RM_2, which remains fixed thereafter (see Appendix 6.14 for more annotation details). **(3) Offline warm-up.** In parallel with calibration, we label $\mathcal{D}_{\text{demo}}$ using RM_1 and update π_1 offline to obtain π_2, providing a stronger starting point and helping avoid rollout batches dominated by failures. **(4) Iterative improvement.** Starting from π_2, we repeatedly collect rollouts, auto-label them with RM_2, and update the policy

using only the latest round's rollouts with the chosen RL backend, without further progress annotation or corrective teleoperation. Later rollouts require safety monitoring, episode outcome recording, and environment resets.

3.2.2 Combining Dense and Sparse Rewards for Stable Learning

We combine the dense rewards in Equation 1 with sparse rewards from episode success or failure to regularize policy learning against errors in the learned value estimates. The human safety monitor records this episode outcome before the environment is reset for the next episode. DICE-RL uses dense rewards in a temporal-difference (TD) critic objective and sparse rewards in a Monte Carlo (MC) objective, while AWR uses the two signals to filter action chunks. The backend-specific formulations are detailed in Appendices 6.15 and 6.16.

3.2.3 Replaceable Policy RL Algorithms

We design SARM2 to provide dense reward signals for policy learning, allowing different RL algorithms to be plugged into SPIRAL as interchangeable policy-update modules.

DICE-RL backend. We inherit the distribution-contractive DICE-RL algorithm (Sun & Song, 2026). Within SPIRAL, its action-chunk critic learns from SARM2 rewards using the objective detailed in Appendix 6.15. We retain its action-chunk critic, multi-sample expectation training, and residual policy. The base flow policy remains fixed while a residual actor learns bounded action corrections, regularized toward the base policy. At inference, we execute one base-policy action chunk with its residual correction, without evaluating the critic. Implementation details are provided in Appendix 6.15.

AWR backend. We inherit advantage-weighted regression (Peng et al., 2019) and follow the advantage-based policy optimization formulation in $\pi^*_{0.6}$ (Physical Intelligence et al., 2025). In our implementation, we fine-tune only the VLA's action head while keeping its vision-language backbone frozen. We use SARM2 itself to supply value estimates at the start and end of each action chunk spanning h control steps. A binary positive-advantage indicator then gates action chunks for action-head fine-tuning with a flow-matching objective. Appendix 6.16 specifies the mixed advantage formulation and filtering update.

4 Experimental Results

We evaluate **SARM2** through three questions: **Q1:** Does SARM2 accurately estimate progress across tasks? **Q2:** Can SARM2's rewards drive policy self-improvement? **Q3:** How does reward quality affect self-improvement?

4.1 Experiment Setup

4.1.1 Reward Model Evaluation

We evaluate 10 tasks: five *classic* tasks ($\mathcal{S}_1$; e.g., pick-and-place, folding) common in reward model training and five *unconventional* tasks ($\mathcal{S}_2$; tool use or multi-stage composition; Appendix 6.2). Metrics are per-frame mean squared error (MSE) on 10 held-out demonstrations per task, rollout classification on Folding Shorts (T1) and Cleaning Whiteboard (T2), and MoE routing diagnostics (Appendix 6.6). Baselines are **ReWiND** (Zhang et al., 2025a) (**RW**), trained from scratch on these tasks without stages or MoE; multitask **SARM** (Chen et al., 2025b) (shared primitive estimator, dense head conditioned on primitive text); **TOPReward** (Chen et al., 2026) (**TR**, *Qwen3-VL-8B-Instruct*); and **Robometer** (Liang et al., 2026) (**RMT**). We also evaluate **RMT-FT**, adapted to these tasks with LoRA for 2000 steps following the official recipe. ReWiND, RMT-FT, and SARM2 receive identical inputs: three RGB camera views (two wrists and top) and proprioceptive states. RMT-FT is substantially larger than SARM2; its $2{,}000$ LoRA steps take approximately the same wall-clock time as SARM2's $10{,}000$ training steps, using a GPU with twice the memory capacity of the GPU used for SARM2. Ablations remove stage estimation (**w/o SE**), use one gate (**w/o MG**), or train only on $\mathcal{S}_1$ ($\mathcal{S}_1$ **Var.**). Appendices 6.8 and 6.9 detail compute matching and MMoE hyperparameter ablations.

4.1.2 Policy Learning

We evaluate policy improvement on two long-horizon bimanual tasks. **Folding Shorts** is a deformable-object task evaluated from both flat and crumpled initial states; its task policy is trained on 20 hours comprising 670 unfiltered demonstrations. **Cleaning Whiteboard** requires one arm to

stabilize a movable board while the other grasps an eraser, removes all letters, returns the objects, and terminates correctly; its task policy uses 10 hours comprising 530 demonstrations. We evaluate every reported task condition and checkpoint using 30 real robot trials and report success rates together with average progress for the Cleaning Whiteboard task. Full task definitions and scoring rules appear in Appendix 6.2 and Appendix 6.7.

Baselines and Ablations. Table 2 organizes our comparisons along three dimensions. **(1) Data source:** offline methods use demonstrations only, while rollout methods additionally learn from policy deployment data. **(2) Policy learning algorithm:** offline, we compare reward-aware BC (RA-BC) (Chen et al., 2025b), DICE-RL, and AWR as reward-based methods, with standard BC as the reference and offline RL's base policy. For online self-improvement, we also instantiate SPIRAL with AWR alongside DICE-RL to demonstrate its support for replaceable RL algorithms. **(3) Reward signal source:** we compare sparse terminal rewards or dense rewards from RMT-FT or SARM2 on the same DICE-RL backend. RMT-FT and SARM2 receive one-time calibration on the same approximately 100 rollouts per task, with identical human segment and final-progress annotations (Appendix 6.14). All reward-source variants start from the same offline DICE-RL checkpoint trained with SARM2 rewards; AWR uses its own offline checkpoint, and all methods collect 50 trajectories per round. We also ablate offline warm-up, one-time rollout calibration, and the auxiliary sparse MC term within the DICE-RL backend.

4.2 Q1: Does SARM2 Accurately Estimate Progress Across Tasks?

Table 1 shows that **SARM2** and its $\mathcal{S}_1$-only variant achieve the best results across all reported metrics.

(1) SARM2 provides more accurate progress estimates. Across ten tasks, **SARM2** reduces overall demonstration MSE by 53.5% relative to the best general-purpose reward model baseline evaluated, RMT-FT. LoRA fine-tuning improves Robometer's demo loss and rollout score, but its overall demo MSE remains higher than every from-scratch in-domain baseline despite its larger size ($\sim$ 4B vs. $\sim$ 200M parameters). We attribute this gap to LoRA's restricted parameter updates and MLLM pretraining focused on semantics and language generation, which can limit adaptation to fine-grained progress estimation within the training budget.

(2) Stage estimation and MMoE are individually beneficial and synergistic. Removing the stage estimator (**w/o SE**) raises demo loss from 0.020 to 0.034; removing the multi-gate design (**w/o MG**) raises it to 0.026. The full **SARM2** model improves substantially over both, showing that stage-awareness and multi-gate routing capture complementary structure: the stage estimator localizes the model in the task's primitive sequence, while the MMoE allocates capacity along primitive-group boundaries. The routing diagnostics provide a complementary explanation: the normalized concentration score falls from 0.87 without stage estimation and 0.42 without multiple gates to 0.10 for full SARM2, where lower scores indicate more balanced expert use (Equation 6).

(3) Training on a broader task distribution improves accuracy. Compared with the $\mathcal{S}_1$-only variant, **SARM2** trained on both subsets reduces $\mathcal{S}_1$ demonstration MSE by 40%, while their T1 rollout

Table 1: **Reward model evaluation on the 10-task benchmark. RW**, **TR**, **RMT**, and **RMT-FT** denote ReWiND, TOPReward, Robometer, and task-adapted Robometer. **Demo** $\mathcal{L}$: per-frame MSE on held-out demos for the classic ($\mathcal{S}_1$) and unconventional ($\mathcal{S}_2$) subsets and frame-level micro-average. **Rollout** $\rho \in [-1, 1]$, more details refer to Section 6.6 (T1: Folding Shorts, T2: Cleaning Whiteboard). **MoE Density**: normalized per-sample max expert utilization (values close to 1 indicate routing collapse, more details in Equation 6). Best per row in bold.

Metrics	RW	TR	RMT	RMT-FT	SARM	w/o SE	w/o MG	$\mathcal{S}_1$ Var.	SARM2
Demo $\mathcal{L} \downarrow$									
$\mathcal{S}_1$	0.026	0.090	0.067	0.032	0.019	0.025	0.010	0.010	**0.006**
$\mathcal{S}_2$	0.044	0.120	0.113	0.052	0.038	0.040	0.038	N/A	**0.031**
Overall	0.036	0.107	0.093	0.043	0.030	0.034	0.026	N/A	**0.020**
Rollout $\rho \uparrow$									
T1	0.167	-0.222	0.222	0.667	0.667	0.556	0.611	**0.889**	0.833
T2	0.222	-0.444	-0.111	0.000	0.611	0.556	0.333	N/A	**0.667**
MoE Density	N/A	N/A	N/A	N/A	N/A	0.87	0.42	0.23	0.10

scores remain close (0.833 vs. 0.889). This suggests that joint training captures shared progress patterns, improving accuracy on existing tasks while expanding task coverage.

4.3 Q2: Can SARM2's Rewards Drive Policy Self-Improvement?

Table 2 and Figure 4 show gains from demonstrations and subsequent rollouts, illustrated in Figure 5.

(1) SPIRAL supports offline policy improvement. Using demonstrations alone, SPIRAL with either DICE-RL or AWR improves success over BC by at least 26.7 percentage points across all evaluated task conditions. The **w/o offline warm-up** ablation shows the largest degradation on Folding Shorts, where BC rarely succeeds. Offline warm-up provides a stronger starting point and helps avoid rollout batches dominated by failures.

(2) Online self-improvement extends the offline gains. Starting from its offline DICE-RL checkpoint, SPIRAL raises success by 40.0, 33.3, and 40.0 percentage points on flat shorts, crumpled shorts, and whiteboard cleaning, respectively (Figure 4). Whiteboard progress also rises from 0.825 to 0.975. Rollouts expose the policy's own execution errors, bringing the training data closer to the distribution during deployment. SARM2's dense rewards distinguish productive actions from mistakes within these trajectories, so even unsuccessful episodes can supply useful learning signals.

(3) SARM2 supports replaceable RL algorithms. Within SPIRAL, SARM2 supports policy improvement with both DICE-RL and AWR, which achieve broadly similar final performance (Table 2). AWR's slightly lower success reflects the coarser guidance of binary filtering: it only selects action chunks for training, whereas DICE-RL's critic directly guides action refinement toward higher predicted returns (Appendices 6.15 and 6.16).

Table 2: **Policy evaluation and SPIRAL component ablations.** Offline methods use demonstrations only. Rollout methods follow SPIRAL's offline-to-online protocol. Ablations use DICE-RL with SARM2 rewards; the complete-pipeline reference repeats the rollout result. Each condition is evaluated over 30 trials; SR denotes success rate (%), and Prog. denotes mean task progress.

Data	Backend	Reward	Folding Shorts SR (%)↑		Whiteboard	
			Flat	Crumpled	SR (%)↑	Prog.↑
Offline	BC	—	6.7	0.0	23.3	0.483
	RA-BC	SARM2	50.0	**43.3**	43.3	0.750
	DICE-RL	Sparse	33.3	16.7	33.3	0.617
	DICE-RL	SARM2	60.0	33.3	**50.0**	**0.825**
	AWR	SARM2	**63.3**	26.7	**50.0**	0.800
Rollout	DICE-RL	Sparse	56.7	16.7	53.3	0.825
	DICE-RL	RMT-FT	83.3	40.0	63.3	0.883
	DICE-RL	SARM2	**100.0**	**66.7**	**90.0**	**0.975**
	AWR	SARM2	96.7	60.0	80.0	0.950
Component ablations (DICE-RL w/ SARM2)						
Complete SPIRAL			**100.0**	**66.7**	**90.0**	**0.975**
w/o offline warm-up			16.7	0.0	66.7	0.892
w/o rollout calibration (use RM_1)			86.7	43.3	73.3	0.908
w/o sparse reward regularization			90.0	53.3	80.0	0.933

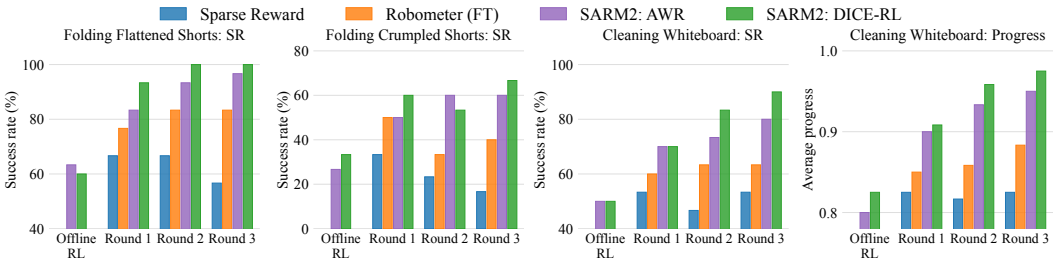


Figure 4: **SPIRAL self-improvement with DICE-RL and AWR.** Bars report success rate (SR) and mean whiteboard progress at offline RL initialization and after each rollout-update round. Sparse Reward and Robometer (FT) use DICE-RL and share the offline checkpoint shown as "SARM2: DICE-RL"; "SARM2: AWR" starts from its own offline checkpoint. Each success-rate percentage and mean progress value is computed from 30 real robot trials per task condition and checkpoint.

4.4 Q3: How Does Reward Quality Affect Self-Improvement?

We examine how reward sources, rollout calibration, and sparse reward regularization affect self-improvement (Table 2 and Figure 4).

(1) Reward quality is critical for long-horizon self-improvement. With the same DICE-RL backend, offline starting policy, and rollout budget, SARM2 outperforms both sparse rewards and RMT-FT on all tasks metrics. These comparisons highlight the need for accurate reward signals that distinguish progress, adjustment, and errors throughout challenging long-horizon execution, with more qualitative visualizations in Figures 9 and 10.

(2) Calibration and sparse reward regularization improve learning. Using uncalibrated RM_1 weakens self-improvement across both tasks, although it still yields substantial gains over BC. Removing sparse reward regularization also reduces success and whiteboard progress, with smaller losses than omitting calibration. These ablations support SPIRAL's design: rollout calibration aligns reward estimates with policy-generated states, while sparse reward regularization anchors value learning to episode outcomes, helping stabilize learning from imperfect dense rewards.

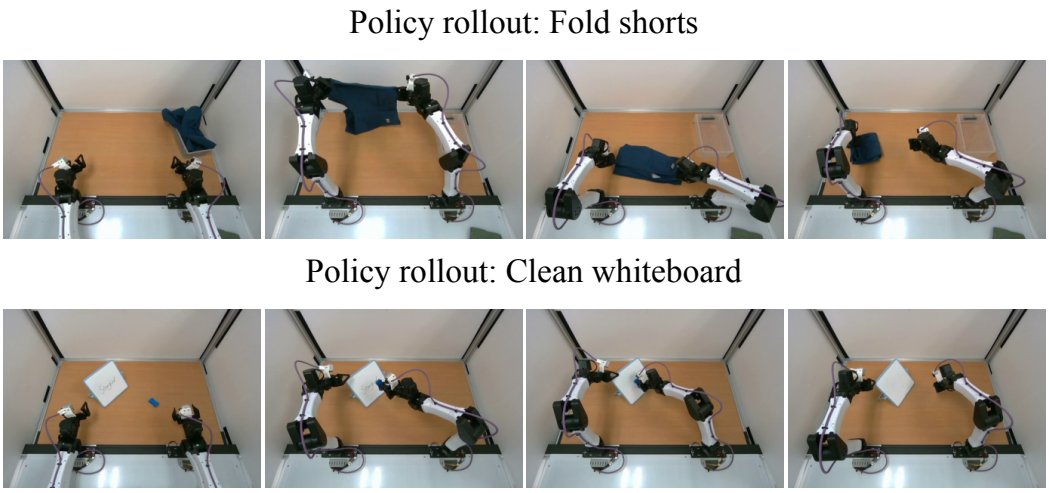


Figure 5: **Example policy rollouts.** Selected frames from Folding Shorts (top) and Cleaning Whiteboard (bottom), ordered from left to right.

5 Conclusion

We present **SARM2**, a multitask stage-aware reward model unifying an action-primitive stage estimator with a multi-gate MoE value decoder, producing dense, accurate rewards across long-horizon manipulation tasks. We evaluate SARM2 through SPIRAL, an offline-to-online self-improvement pipeline with replaceable RL modules. Across a 10-task benchmark, SARM2 outperforms task-specific and MLLM-based baselines on demo MSE and rollout classification; ablations show stage-awareness, multi-gate routing, and training-distribution diversity each contribute. Real robot experiments on two challenging long-horizon tasks show that SARM2's dense, accurate reward signals support both offline policy improvement and continued self-improvement from policy rollouts.

Limitations. Our approach has several limitations. *(1) Embodiment scope.* The action-primitive vocabulary is derived from a bimanual table-top dataset, so extending SARM2 to mobile manipulation or other embodiments requires rebuilding the primitive set and re-training the stage estimator. *(2) In-domain value training.* Although the primitive estimator transfers across tasks, the value head is trained on downstream task data; SARM2 is therefore a multitask reward model rather than a zero-shot general one. *(3) Human effort.* Though it is one-time effort, each downstream task requires a lightweight annotation of policy rollouts to cover failure and recovery states. Later rounds require no additional progress annotations, while safety monitoring, episode outcome recording, and environment resets continue.

AI USE STATEMENT

We used generative AI tools for text polishing and coding assistance. We did not use generative AI tools to generate experimental results, fabricate data, or make final scientific claims. All AI-assisted text, code, and suggestions were manually reviewed, revised, and verified by the authors. The authors take full responsibility for the final content, including all text, code, experimental results, and scientific claims.

REPRODUCIBILITY STATEMENT

We provide our code and real robot videos in the Supplementary Material to support reproducibility. Section 3 describes the SARM2 architecture, reward construction, and SPIRAL pipeline. Section 4 and the appendix detail the evaluation tasks, baselines, metrics, and protocols. Table 4 in Appendix 6.5 lists hyperparameters for SARM2 training, behavior cloning, and the DICE-RL and AWR backends.

REFERENCES

Minttu Alakuijala, Reginald McLean, Isaac Woungang, Nariman Farsad, Samuel Kaski, Pekka Marttinen, and Kai Yuan. Video-language critic: Transferable reward functions for language-conditioned robotics. *arXiv preprint arXiv:2405.19988*, 2024.

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual off-policy rl for finetuning behavior cloning policies. *arXiv preprint arXiv:2509.19301*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Qianzhong Chen, Naixiang Gao, Suning Huang, JunEn Low, Timothy Chen, Jiankai Sun, and Mac Schwager. Grad-nav++: Vision-language model enabled visual drone navigation with gaussian radiance fields and differentiable dynamics. *IEEE Robotics and Automation Letters*, 11(2):1418–1425, 2025a.

Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Yide Shentu, and Philipp Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. *arXiv preprint arXiv:2509.25358*, 2025b.

Shirui Chen, Cole Harrison, Ying-Chun Lee, Angela Jin Yang, Zhongzheng Ren, Lillian J Ratliff, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Topreward: Token probabilities as hidden zero-shot rewards for robotics. *arXiv preprint arXiv:2602.19313*, 2026.

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. Conrft: A reinforced fine-tuning method for vla models via consistency policy. *arXiv preprint arXiv:2502.05450*, 2025c.

Baiye Cheng, Tianhai Liang, Suning Huang, Maanping Shao, Feihong Zhang, Botian Xu, Zhengrong Xue, and Huazhe Xu. Moe-dp: An moe-enhanced diffusion policy for robust long-horizon robotic manipulation with skill decomposition and failure recovery. *arXiv preprint arXiv:2511.05007*, 2025.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024.

Zhiying Du, Bei Liu, Yaobo Liang, Yichao Shen, Haidong Cao, Xiangyu Zheng, Zhiyuan Feng, Zuxuan Wu, Jiaolong Yang, and Yu-Gang Jiang. Himoe-vla: Hierarchical mixture-of-experts for generalist vision-language-action policies. *arXiv preprint arXiv:2512.05693*, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. Improving vision-language-action model with online reinforcement learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15665–15672. IEEE, 2025.

Ce Hao, Xuanran Zhai, Yaohua Liu, and Harold Soh. Abstracting robot manipulation skills via mixture-of-experts diffusion policies. *arXiv preprint arXiv:2601.21251*, 2026.

Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

Runhan Huang, Shaoting Zhu, Yilun Du, and Hang Zhao. Moe-loco: Mixture of experts for multitask locomotion. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 14218–14225. IEEE, 2025.

Suning Huang, Zheyu Zhang, Tianhai Liang, Yihan Xu, Zhehao Kou, Chenhao Lu, Guowei Xu, Zhengrong Xue, and Huazhe Xu. Mentor: Mixture-of-experts network with task-oriented perturbation for visual reinforcement learning. *arXiv preprint arXiv:2410.14972*, 2024.

Kuo-Han Hung, Pang-Chi Lo, Jia-Fong Yeh, Han-Yuan Hsu, Yi-Ting Chen, and Winston H Hsu. Victor: Learning hierarchical vision-instruction correlation rewards for long-horizon manipulation. *arXiv preprint arXiv:2405.16545*, 2024.

I2RT-Robotics. Yam – 6-dof robotic arm. `https://i2rt.com/products/yam-6-dof-arm`, 2025.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pp. 651–673. PMLR, 2018.

Changyeon Kim, Minho Heo, Doohyun Lee, Jinwoo Shin, Honglak Lee, Joseph J Lim, and Kimin Lee. Subtask-aware visual reward learning from segmented demonstrations. *arXiv preprint arXiv:2502.20630*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

Hongyang Lei, Xiaolong Cheng, Qi Qin, Dan Wang, Kun Fan, Huazhen Huang, Qingqing Gu, Yetao Wu, Zhonglin Jiang, Yong Chen, et al. M3-jepa: Multimodal alignment via multi-gate moe based on the joint-embedding predictive architecture. *arXiv preprint arXiv:2409.05929*, 2024.

Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. Rl-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint arXiv:2510.14830*, 2025.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S Huang, Luke Zettlemoyer, Dieter Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16961–16969. IEEE, 2024.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1930–1939, 2018.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning*, pp. 23301–23320. PMLR, 2023.

Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, et al. Vision language models are in-context value learners. In *The Thirteenth International Conference on Learning Representations*, 2024.

Tongzhou Mu, Minghua Liu, and Hao Su. Drs: Learning reusable dense rewards for multi-stage tasks. *arXiv preprint arXiv:2404.16779*, 2024.

Haoyi Niu, Qimao Chen, Tenglong Liu, Jianxiong Li, Guyue Zhou, Yi Zhang, Jianming Hu, and Xianyuan Zhan. xted: Cross-domain adaptation via diffusion-based trajectory editing. *arXiv preprint arXiv:2409.08687*, 2024.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, et al. $\pi^*_{0.6}$: A vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

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

RealSense. Realsensedepth camera d405. `https://store.realsenseai.com/buy-intel-realsense-depth-camera-d405.html`, 2025.

Allen Z. Ren, Justin Lidard, Lars L. Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

Younggyo Seo, Jafar Uruç, and Stephen James. Continuous control with coarse-to-fine reinforcement learning. *arXiv preprint arXiv:2407.07787*, 2024.

Tim Seyde, Wilko Schwarting, Igor Gilitschenski, Markus Wulfmeier, and Daniela Rus. Strength through diversity: Robust behavior learning via mixture policies. In *Conference on Robot Learning*, pp. 1144–1155. PMLR, 2022.

Weijie Shen, Yitian Liu, Yuhao Wu, Zhixuan Liang, Sijia Gu, Dehui Wang, Tian Nian, Lei Xu, Yusen Qin, Jiangmiao Pang, et al. Expertise need not monopolize: Action-specialized mixture of experts for vision-language-action learning. *arXiv preprint arXiv:2510.14300*, 2025.

Zhanyi Sun and Shuran Song. From prior to pro: Efficient skill mastery via distribution contractive rl finetuning. *arXiv preprint arXiv:2603.10263*, 2026.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, et al. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

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

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. Daydreamer: World models for physical robot learning. *Conference on Robot Learning*, 2022.

Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12156–12163. IEEE, 2024.

Philipp Wu, Yide Shentu, Qiayuan Liao, Ding Jin, Menglong Guo, Koushil Sreenath, Xingyu Lin, and Pieter Abbeel. Robocopilot: Human-in-the-loop interactive imitation learning for robot manipulation. *arXiv preprint arXiv:2503.07771*, 2025.

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Yuqi Xie, Fengyuan Hu, Jimmy Wu, Zhengyi Luo, Linxi Fan, et al. Self-improving vision-language-action models with data generation via residual rl. *arXiv preprint arXiv:2511.00091*, 2025.

Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4804–4811. IEEE, 2024.

Zhenjie Yang, Yilin Chai, Xiaosong Jia, Qifeng Li, Yuqian Shao, Xuekai Zhu, Haisheng Su, and Junchi Yan. Drivemoe: Mixture-of-experts for vision-language-action model in end-to-end autonomous driving. *arXiv preprint arXiv:2505.16278*, 2025.

Brent Yi, Hongsuk Choi, Himanshu Gaurav Singh, Xiaoyu Huang, Takara E. Truong, Carmelo Sferrazza, Yi Ma, Rocky Duan, Pieter Abbeel, Guanya Shi, C. Karen Liu, and Angjoo Kanazawa. Flow policy gradients for robot control. *arXiv preprint arXiv:2602.02481*, 2026.

Justin Yu, Andrew Goldberg, Kavish Kondap, Karim El-Refai, Ethan Ransing, Qianzhong Chen, Mac Schwager, Fred Shentu, Philipp Wu, and Ken Goldberg. Warp-rm: A warp-augmented relative progress reward model for data curation. *arXiv preprint arXiv:2606.28320*, 2026.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. *arXiv preprint arXiv:2505.10911*, 2025a.

Tonghe Zhang, Chao Yu, Sichang Su, and Yu Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning. *Advances in Neural Information Processing Systems*, 2025b.

Xiucheng Zhang, Yang Jiang, Hongwei Qin, Jiashuo Bai, and Maocheng Bai. Language-conditioned representations and mixture-of-experts policy for robust multi-task robotic manipulation. *IEEE Robotics and Automation Letters*, 11(5):6153–6160, 2026.

Yinuo Zhao, Huiqian Jin, Lechun Jiang, Xinyi Zhang, Kun Wu, Pei Ren, Zhiyuan Xu, Zhengping Che, Lei Sun, Dapeng Wu, et al. Real-world reinforcement learning from suboptimal interventions. *arXiv preprint arXiv:2512.24288*, 2025.

Chuanyang Zheng, Jiankai Sun, Yihang Gao, Enze Xie, Yuehao Wang, Peihao Wang, Ting Xu, Chang Matthew, Liliang Ren, Jingyao Li, Jing Xiong, Kashif Rasul, Mac Schwager, Anderson Schneider, Zhangyang Wang, and Yuriy Nevmyvaka. Understanding the mixture-of-experts with nadaraya-watson kernel. *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

6 APPENDIX

6.1 HARDWARE SETUP

For our real-world experiments, we use a bimanual tabletop robotic platform. The system consists of:

- Two 6-DOF YAM robot arms manufactured by I2RT (I2RT-Robotics, 2025).
- Three RealSense D405 cameras: one mounted on each wrist and one fixed overhead camera for observing the workspace (RealSense, 2025).

Demonstration data is collected through a leader-follower GELLO teleoperation system (Wu et al., 2024). The environment is recorded at 30 fps and includes synchronized streams from three camera views: the left wrist, the right wrist, and the fixed top view. It also records robot joint angles and commanded joint-angle actions.

Autonomous policy rollouts use the same three synchronized camera views and execute under a 30 Hz control loop. During execution, the operator remains outside the robot's reach and does not provide corrective actions. Intervention is limited to terminating an episode at its time limit or stopping dangerous actions that could damage the robot or workspace; the scene is then reset manually between episodes. Before each reset, the safety monitor records whether the preceding episode succeeded or failed, providing the sparse reward for policy learning.



Figure 6: The physical station used for data collection and policy evaluation.

6.2 TASK DESCRIPTION

We describe all 10 evaluation tasks below, grouped by subset, with the corresponding visualizations consolidated in Figure 7. $\mathcal{S}_1$ (*classic*) contains the two pick-and-place tasks, two cloth-folding tasks, and pulling plugs, all of which align well with primitives that dominate general reward model training distributions. $\mathcal{S}_2$ (*unconventional*) contains the remaining five tasks, which require tool use, deformable or articulated object manipulation, and multi-stage compositional execution. For tasks such as pick-and-place in which the same action sequence recurs many times within a single episode, the robot is required to repeat that sequence until the table is cleared.

$\mathcal{S}_1$: CLASSIC TASKS

Pick and place plates into bin. Several plates are arranged on the board, and the robot is required to clear them one at a time. In each cycle, the robot first picks up a plate from the center of the board and then places it in the plastic bin. The robot repeats this cycle until no plates remain on the board.

Pick and place plates into dish rack. This task is a variant of the bin task in which each plate must instead be inserted vertically into a slot of a dish rack, which requires precise pose alignment between the gripper and the rack opening. In each cycle, the robot first picks up a plate from the board and then inserts it into an available slot of the dish rack. The robot repeats this cycle until the board is cleared, which typically requires 5 to 10 insertion cycles per episode.

Fold the t-shirt. This is a long-horizon deformable-object task. The robot first grasps a T-shirt from the pile and transports it to the center of the board. It then flattens the garment and performs two consecutive folds. Finally, the robot places the neatly folded T-shirt in the corner of the board and resets both grippers. The flattening stage is the longest and visually most variable, whereas the folding stages are bimanual and require coordinated pinch grasps.

Fold the shorts. The procedure for this task closely mirrors that of folding the T-shirt. The robot first grasps the shorts from the pile and moves them to the center of the board. It then flattens the garment, performs two consecutive folds, and finally places the folded shorts in the corner of the board.

Pull plugs off the socket. The robot first repositions a power strip from its initial location on the board to a more convenient working location. It then iteratively removes each plug from the socket and deposits it at the center of the board. After several plugs have been removed, the robot may translate the power strip again to expose the remaining plugs from a more accessible angle, after which the unplug-and-place cycle continues. The episode terminates once all the plugs are removed from power strip and the power strip has been placed back at the center of the board.

$\mathcal{S}_2$: UNCONVENTIONAL TASKS

Clean whiteboard with whiteboard eraser. One arm closely holds the small whiteboard to stabilize it throughout the task, while the other arm retrieves the whiteboard eraser placed at the center of the board. The holding arm then tilts the whiteboard to expose its surface, and the other arm wipes the letters from the whiteboard. After wiping, the robot places the eraser back at the center of the board and finally returns the whiteboard to its original position. The task is bimanual throughout, with one arm stabilizing the tilted whiteboard while the other performs the wiping motion.

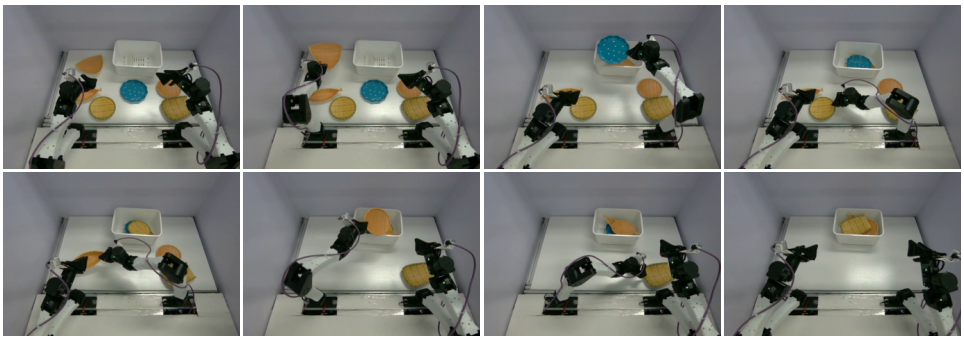
Set dinner table. This is a composite long-horizon task that combines deformable manipulation with multi-object placement. The robot first spreads a tablecloth across the workspace and places a plate at the center of the cloth. It then picks up a glass with the right arm and sets it down at the appropriate location. Finally, the robot arranges the utensils, which involves left-to-right and right-to-left hand-overs to bring each utensil (a knife and a fork) into the appropriate gripper before its final placement on the tablecloth.

Sweep paper scraps with broom. The robot first picks up a broom from the center of the board, performing one or more bimanual hand-overs as needed to reach a stable wielding grasp, and then picks up a dustpan with the other arm. The main body of the task is a long repeated cycle: the robot sweeps paper scraps from the center of the board, sweeps them into the dustpan, and dumps the scraps from the dustpan into a trash bin. This sweep-gather-dump cycle is repeated until no scraps remain, after which the dustpan and broom are placed back on the board.

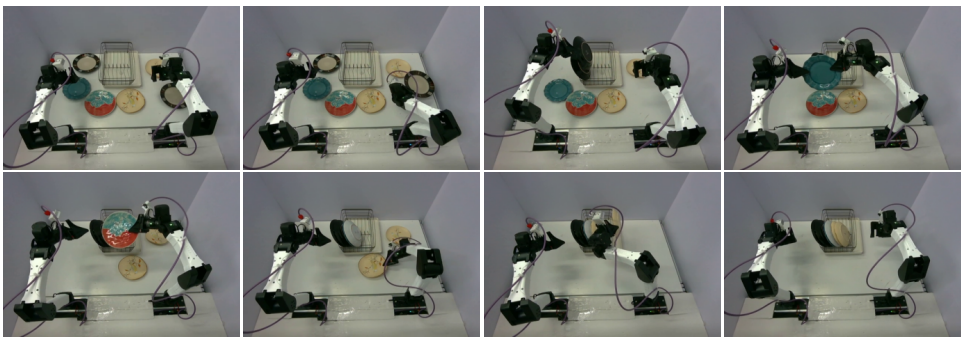
Coil and wrap headphones. This is a precision-demanding task. The robot first picks up a pair of headphones from the center of the board. It then performs a long bimanual wrapping motion that coils the cable around the body of the headphones. Once wrapping is complete, the robot places the wrapped headphones back at the center of the board and resets both arms.

Put away an umbrella. This is a multi-stage articulated-object task. The robot first picks up an open umbrella from the center of the board and passes it from the right arm to the left. It then presses the snap button against the runner to release the canopy and presses the shaft to collapse the umbrella. Next, the robot adjusts and folds the canopy along its natural creases. Finally, the canopy is rolled around the shaft, the strap is grasped and the velcro is fastened, and the closed umbrella is placed back at the center of the board.

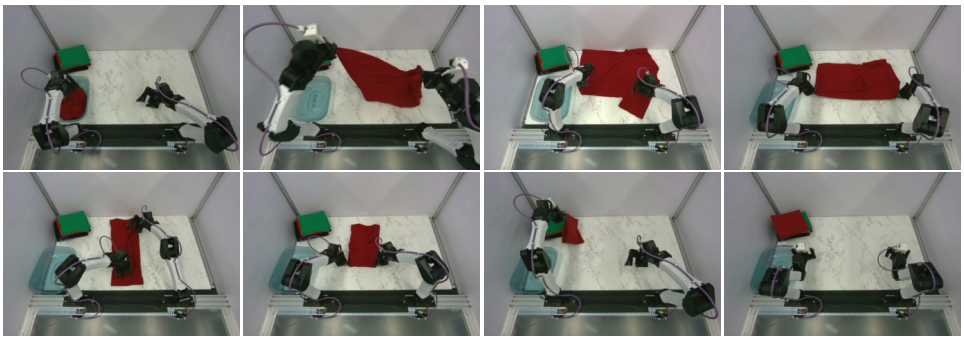
Pick and place plates into bin



Pick and place plates into dish rack



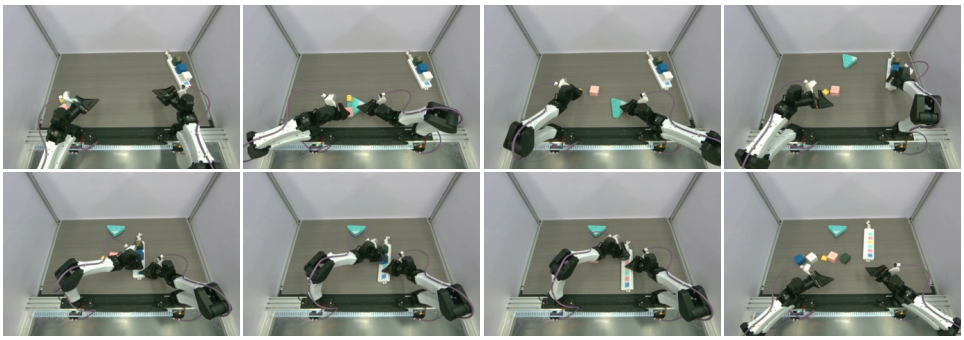
Folding the t-shirt



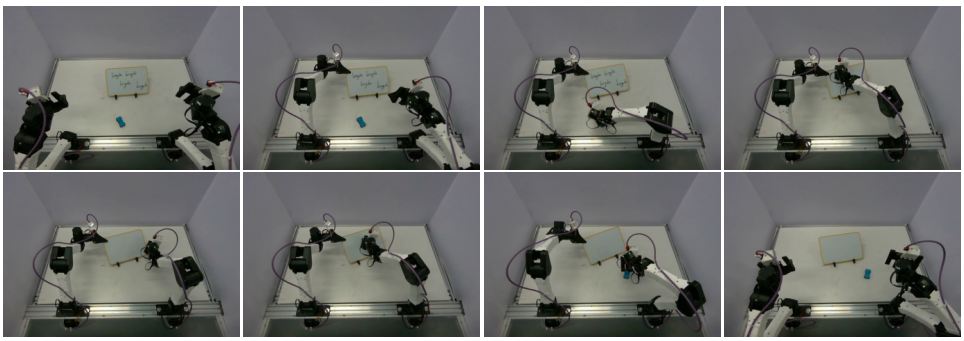
Folding shorts



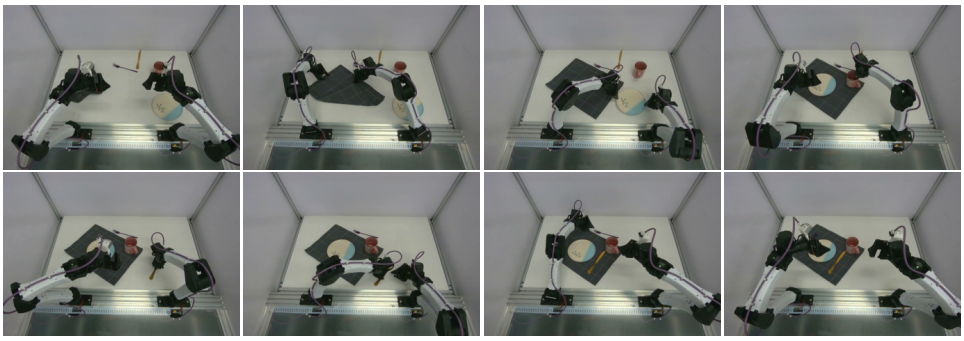
Pull plug off the socket



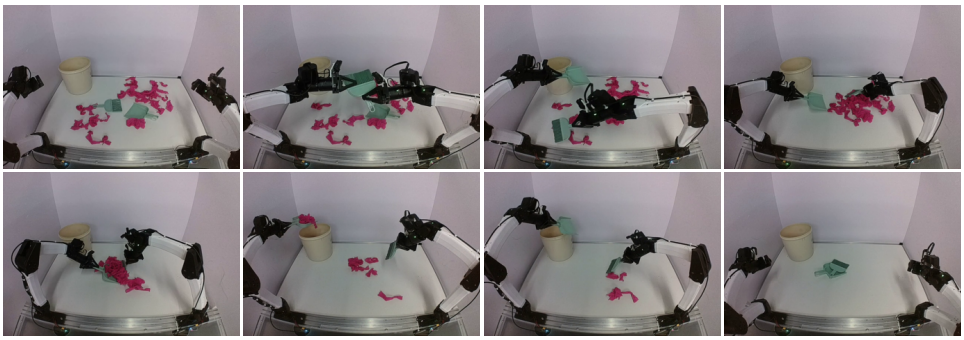
Clean whiteboard with whiteboard eraser



Set dinner table



Sweep paper scraps with broom



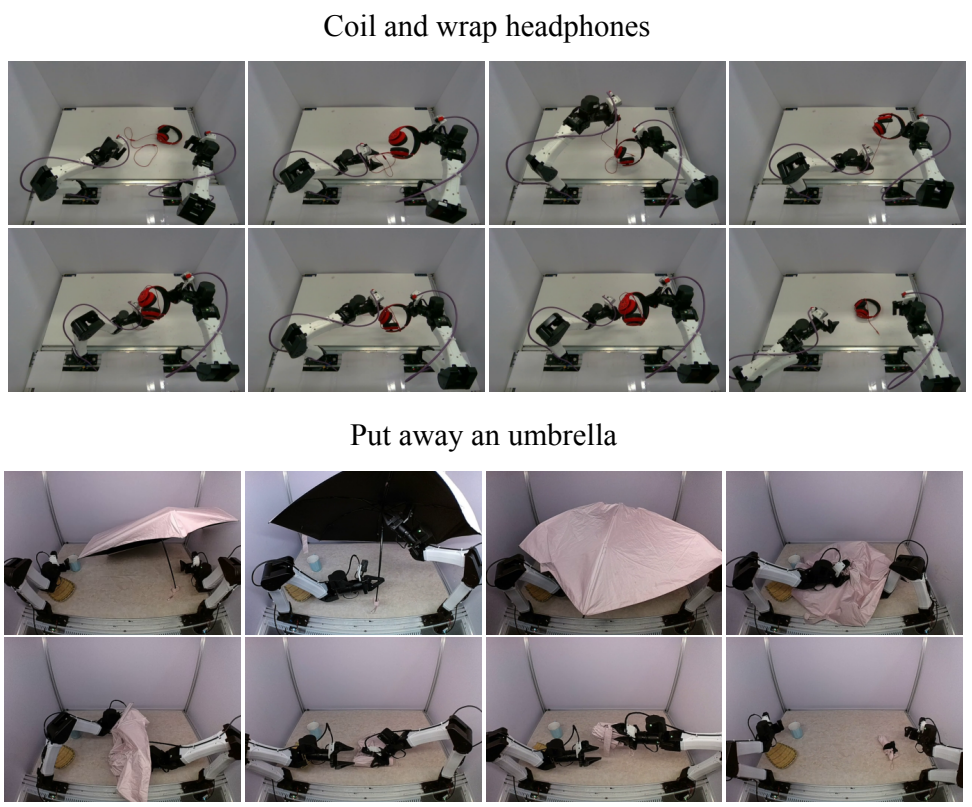


Figure 7: **The ten reward model evaluation tasks.** In display order: $\mathcal{S}_1$: (1) place plates into a bin, (2) place plates into a dish rack, (3) fold a T-shirt, (4) fold shorts, and (5) pull plugs from a power strip; $\mathcal{S}_2$: (6) clean a whiteboard, (7) set a dinner table, (8) sweep paper scraps, (9) coil and wrap headphones, and (10) put away an umbrella.

6.3 SPIRAL Offline-to-Online Algorithm

Algorithm 1 SPIRAL with a replaceable policy-update backend

Require: Demonstrations $\mathcal{D}_{\text{demo}}$; pretrained policy π_{VLA}; pretrained reward model RM_1; reward model data $\mathcal{D}_{\text{RM}}$; policy-update backend $\mathcal{A}$; maximum rounds N
Ensure: Improved policy π_{N+1}

// Stage 1: Behavior-cloning initialization
$\pi_1 \leftarrow \textsc{BehaviorClone}(\pi_{\text{VLA}}, \mathcal{D}_{\text{demo}})$; $\mathcal{R}_1 \leftarrow \textsc{Rollout}(\pi_1, 50\text{–}100)$
// Stage 2: One-time reward model adaptation
$\mathcal{R}_1^{\text{human}} \leftarrow \textsc{AnnotateSegments}(\mathcal{R}_1)$; $\mathcal{D}_{\text{adapt}} \leftarrow \mathcal{R}_1^{\text{human}} \cup \textsc{Subsample}(\mathcal{D}_{\text{RM}}, 0.5)$
$\text{RM}_2 \leftarrow \textsc{FinetuneReward}(\text{RM}_1, \mathcal{D}_{\text{adapt}})$
// Stage 3: Offline reward-guided warm-up (parallel execution with Stage 2)
$\widetilde{\mathcal{D}}_{\text{demo}} \leftarrow \textsc{LabelRewards}(\mathcal{D}_{\text{demo}}, \text{RM}_1)$
$\pi_2 \leftarrow \textsc{PolicyUpdate}_{\mathcal{A}}(\pi_1, \widetilde{\mathcal{D}}_{\text{demo}})$; $\mathcal{R}_2 \leftarrow \textsc{Rollout}(\pi_2, 50)$
// Stage 4: Iterative self-improvement
for $i = 2, 3, \ldots, N$ **do**
  $\widetilde{\mathcal{R}}_i \leftarrow \textsc{LabelRewards}(\mathcal{R}_i, \text{RM}_2)$
  $\pi_{i+1} \leftarrow \textsc{PolicyUpdate}_{\mathcal{A}}(\pi_i, \widetilde{\mathcal{R}}_i)$; $\mathcal{R}_{i+1} \leftarrow \textsc{Rollout}(\pi_{i+1}, 50)$
  if $\textsc{SuccessRate}(\mathcal{R}_{i+1}) \geq \tau_{\text{success}}$ **then**
    break
  end if
end for
return π_{i+1}

6.4 ACTION-PRIMITIVE GROUPING

To assign each of the $K{=}21$ action primitives to an MMoE gate, we cluster them into $M{+}1{=}8$ semantic groups according to shared visual appearance and end-effector motion patterns, so that primitives within a group exhibit similar progress signatures and can plausibly share the same gate's routing distribution. Table 3 lists the resulting groups, their member primitives, and a short description of the motion pattern each group captures.

Table 3: **Action-primitive grouping for the multi-gate MoE decoder.** The $K{+}1{=}22$ primitives are clustered into $M{+}1{=}8$ semantic groups by shared visual/motion characteristics; each group maps to one gate of the MMoE decoder while the expert pool is shared. `Other` is a catch-all for undefined or task-specific actions.

Group Name	**Action Primitives**	**Description**
Acquire	scoop, pull, pick/grab	Lift or move a target object up from its initial support surface into robot control.
Release	place, pour/dump	Move a controlled object down to a target location, container, or support surface.
Translate	erase, zip/unzip, sweep, pass	Move the end-effector or object along a primarily linear trajectory.
Insert	install, insert	Guide an object into a constrained target region with spatial alignment.
Shape Clothes	hang, fold/flatten	Reconfigure cloth-like objects into a desired geometric state or arrangement.
Rotate	wrap, roll, tie/untie, rotate/turn	Change an object's orientation or configuration through angular motion.
Force	clamp, press	Apply controlled contact force through the gripper or end-effector.
Other	dummy	Represent undefined or task-specific actions outside the main primitive taxonomy.

6.5 HYPERPARAMETER TABLE

Table 4 consolidates all hyperparameters introduced in the main text, covering the action-primitive stage estimator, the multi-gate MoE value decoder, and the SPIRAL self-improvement loop.

Table 4: **Hyperparameter summary.** Symbols, descriptions, and values for all hyperparameters referenced in the main text.

Symbol	Description	Value
Action-primitive stage estimator		
K	Number of action primitives	21
N	Number of recent context frames per timestep	6
Δ	Frame sampling stride	30
L_{AP}	Transformer encoder layers	4
d	Transformer hidden dimension	512
t_k	Hours of data per primitive in $\mathcal{D}_{\text{AP}}$	3
lr_{AP}	Learning rate	5×10^{-5}
B_{AP}	Batch size	96
T_{AP}	Training steps	10,000
opt_{AP}	Optimizer	AdamW
Multi-gate MoE value decoder		
L_{VD}	Transformer encoder layers	6
M	Number of primitive groups (gates)	8
E	Number of experts in shared pool	10
k	Top-k routing per timestep	2
d_{exp}	Expert hidden width	256
$n_{\text{h}} \times d_{\text{exp}}$	Expert MLP shape (depth $\times$ width)	3×256
λ_{bal}	Load-balance auxiliary loss weight	2.5
λ_{ent}	Entropy auxiliary loss weight	0.5
N_{rw}	Number of rewinding frames	3
lr_{VD}	Learning rate	5×10^{-5}
B_{VD}	Batch size	64
T_{VD}	Training steps	10,000
opt_{VD}	Optimizer	AdamW
Behavior Cloning		
h	Action chunk horizon	50
lr_{BC}	Learning rate	5×10^{-5}
B_{BC}	Batch size	32
T_{BC}	Training steps	30,000
sched_{BC}	Learning-rate schedule	Cosine
opt_{BC}	Optimizer	AdamW
Shared policy training settings		
γ	Per-control-step discount for TD targets and MC returns	0.9995
DICE-RL backend		
α	Sparse-MC weight in the DICE-RL critic loss	0.5
β	BC regularizer weight in residual actor loss	30
N_{critic}	Critic ensemble size	5
σ	Target-policy smoothing noise std	0.2
c	Target-policy smoothing clip range	0.5
κ	Latent samples per observation during DICE-RL training	4
$n_{\text{RL}} \times d_{\text{RL}}$	Actor/Critic MLP shape (depth $\times$ width)	3×1024
τ_{tgt}	Target network soft-update rate	0.005
lr_{RL}	Learning rate (actor / critic)	1×10^{-4}
B_{RL}	Batch size	16
T_{RL}	Training steps per round	5,000
opt_{RL}	Optimizer	AdamW
AWR backend		
β_{AWR}	Policy-extraction exponent	1
ϵ_ℓ	Task-specific advantage threshold	0
$\lambda_{\text{MC}}^{\text{AWR}}$	MC weight in mixed advantage	0.1
lr_{AWR}	Policy learning rate	5×10^{-5}
B_{AWR}	Batch size	32
T_{AWR}	Training steps per round	5,000

6.6 Reward Model Evaluation Protocol

Demonstration evaluation. We sample 10 held-out episodes per task and report the per-frame mean squared error (MSE) between the predicted and ground-truth progress. For clarity of presentation, we report MSE separately on $\mathcal{S}_1$ and $\mathcal{S}_2$, alongside a combined score. We note that the combined MSE is computed as a *micro-average* over all evaluated frames; it is therefore not the arithmetic mean of the two per-subset values.

Rollout evaluation. Following the protocol of SARM, we assess **robot rollout progress estimation** on a real robotic platform. We fine-tune a $\pi_{0.5}$ (Intelligence et al., 2025) policy on each target task and deploy intermediate checkpoints from different training stages, yielding a balanced rollout set of 36 trajectories per task: 12 successful episodes (`SE`), 12 partially successful episodes (`PSE`), and 12 failed episodes (`FE`). For consistency with the $[-1, 0]$ value target, we convert the prediction to progress as $P_t = 1 + \hat{V}_t \in [0, 1]$. Each rollout is classified by the reward model according to:

$$\texttt{Label} = \begin{cases} \texttt{SE}, & \text{if } P_{\text{final}} > 0.8 \ \wedge \ \frac{3}{T}\sum_{t=2T/3}^{T} P_t > 0.6, \\ \texttt{PSE}, & \text{if } \frac{1}{T}\sum_{t=1}^{T} P_t \geq \xi, \\ \texttt{FE}, & \text{otherwise}, \end{cases} \tag{4}$$

where P_t denotes the predicted progress at frame t, T is the trajectory length, and ξ is set to the median of the average progress across all non-successful rollouts. This adaptive threshold yields an equal split between **PSE** and **FE** predictions and avoids the bias introduced by manually tuning a fixed cutoff.

We further summarize rollout classification accuracy via the score

$$\rho = \frac{\#\text{correct} - \#\text{incorrect}}{36} \in [-1, 1], \tag{5}$$

which assigns $+1$ for each correctly classified rollout and -1 otherwise, normalized by the total number of rollouts. A perfect classifier achieves $\rho = 1$, while random ternary guessing yields $\rho \approx -1/3$.

MoE routing diagnostics. For all MoE-based reward models, we additionally report the **top-k routing score**, defined as

$$\mathcal{S}_{\text{route}} \ = \ \frac{r-1}{N/k-1}, \qquad r \ = \ \frac{N}{k}\sum_{i=1}^{k} \bar{p}_{(i)}, \tag{6}$$

where N is the total number of experts, k is the number of activated experts per token, and $\bar{p}_{(i)}$ is the i-th largest entry of the routing distribution averaged over the evaluation set. The raw concentration r measures the mass placed on the top-k experts relative to a uniform baseline $1/N$, ranging from 1 (perfectly balanced) to N/k (fully collapsed); the affine normalization rescales $\mathcal{S}_{\text{route}}$ to $[0, 1]$, removing the dependence on N and k so expert utilization can be compared directly across configurations. A healthy router yields $\mathcal{S}_{\text{route}}$ well below 1.

6.7 Policy Evaluation Protocol

Folding Shorts. We evaluate flat and crumpled initial states separately, using 30 trials for each policy checkpoint. An episode is successful only if the policy completes the flattening and folding sequence, places the folded shorts at the designated location, and terminates within the three-minute time limit used for both flat and crumpled initial states.

Cleaning Whiteboard – five-tier scoring. Each Cleaning Whiteboard episode has a two-minute time limit. Each episode of the Cleaning Whiteboard evaluation is scored using five progress tiers: **0%** for no letters cleaned; **25%** for successfully grasping the eraser, stabilizing the board, and cleaning at least one letter; **50%** for cleaning more than half but not all letters; **75%** for cleaning all letters without stopping correctly, or stopping early with letters remaining; and **100%** for cleaning all letters and terminating correctly by releasing the eraser on the table and board. The reported **Avg. Prog.** metric is the mean tier value across all 30 evaluation episodes, while **SR** counts only episodes that reach the 100% tier.

6.8 MoE Design: Comparison with Dense Model

To put the architectural comparison on equal footing, we contrast our decoder-level Mixture-of-Experts model (henceforth **MoE-Decoder**) against a parameter-comparable dense baseline. Here, **Dense** denotes the multitask SARM baseline, which uses the same predicted action-primitive stage estimator as SARM2 and supplies the selected primitive as a language input to its dense value head. Both models share an identical encoder backbone and differ only in the reward-prediction head, which is applied once per timestep to the fused feature vector of dimension $(N{+}3)\,d = 3072$, where the $N{=}3$ camera streams together with the language, task, and state tokens give $(N{+}3){=}6$ modality streams of length T. The **Dense** baseline computes the scalar reward with a 3-hidden-layer MLP of width 512. The **MoE-Decoder** variant replaces this MLP with a sparse top-k mixture of E deep experts, each itself a 3-hidden-layer MLP of width 256, applied to the same fused vector. Because the MoE activates only $k{=}2$ of $E{=}10$ experts per timestep, the relevant comparison quantity is its *activated* parameter count, not its total. With this configuration, MoE-Decoder is matched to the Dense baseline within $\sim 10\%$ on both activated parameters and FLOPs (Tables 6 and 7), so any quality difference is attributable to the sparse routing structure rather than to capacity or compute headroom.

Table 5 shows that the dense SARM baseline has higher overall demo MSE than MoE-Decoder (0.030 vs. 0.020) and lower rollout ρ on both T1 (0.667 vs. 0.833) and T2 (0.611 vs. 0.667). These results support the benefit of primitive-conditioned expert routing for multitask value prediction at comparable compute.

Table 5: **MMoE and dense value-head comparison.** Demo $\mathcal{L}$ is progress MSE; rollout ρ is evaluated on Folding Shorts (T1) and Cleaning Whiteboard (T2). Lower MSE and higher ρ are better.

Method	**Demo $\mathcal{L}\downarrow$**			**Rollout $\rho\uparrow$**		**MoE Density**
	$\mathcal{S}_1$	$\mathcal{S}_2$	Overall	T1	T2	
SARM2 (Primitive + MoE)	0.006	0.0306	0.020	0.833	0.667	0.10
SARM (dense, no MoE)	0.019	0.038	0.030	0.667	0.611	N/A

Table 6: Activated-parameter breakdown of the MoE-Decoder and Dense (SARM) reward heads. Configuration: fused input dimension 3072, $E{=}10$ experts of width 256 with $n_{\mathrm{h}}{=}3$ hidden layers, $k{=}2$.

Component	**Params**
MoE-Decoder (activated)	
Active gate (LN, Linear $3072{\to}10$)	0.04 M
Active expert (LN, Linear $3072{\to}256$, Linear $256{\to}256{\times}2$, Linear $256{\to}1$)	0.92 M
$\times\, k{=}2$ active experts	1.85 M
MoE-Decoder total (activated)	**1.89 M**
Dense (SARM) baseline	
MLP head: LN, Linear $3072{\to}512$	1.58 M
Linear $512{\to}512$ ($\times 2$)	0.53 M
Linear $512{\to}1$	<0.01 M
Dense (SARM) total	**2.11 M**
MoE-Decoder / Dense (SARM) (activated)	$0.90\times$

6.9 MoE Settings Ablation study

Our default configuration **T2E10** attains the best overall demo loss (0.020) and the strongest T_1 rollout success (0.833), indicating that $k{=}2$ over $E{=}10$ experts essentially reaches the optimum within this sweep. Thanks to the multi-gate design and the load-balance auxiliary loss, all variants maintain non-trivial load scores (0.02–0.20) and remain away from routing collapse, so the observed differences reflect specialization quality rather than degenerate gating. Empirically, varying top-k (T1E10: 0.026, T4E10: 0.029) degrades the overall demo loss more than varying expert count (T2E5: 0.024, T2E20: 0.022); we attribute this to the fact that k directly controls the effective

Table 7: Per-timestep compute of the reward head, in FLOPs (one multiply-accumulate counted as 2 FLOPs). Both heads run once per timestep on the fused vector.

Component	FLOPs / call	Calls / timestep	FLOPs / timestep
MoE-Decoder			
Active gate	0.08 M	1	0.08 M
Active expert ($\times k$=2)	1.85 M	2	3.70 M
MoE-Decoder total			**3.78 M**
Dense baseline			
MLP head	4.20 M	1	4.20 M
Dense total			**4.20 M**
MoE-Decoder / Dense			0.90×

Table 8: **Ablation over MoE routing configurations of SARM2.** TaEb: top-k=a over b experts; **T2E10** is our default. We sweep $k \in \{1, 2, 4\}$ at fixed E=10 and $E \in \{5, 10, 20\}$ at fixed k=2; all other components are held identical. Lower demo MSE is better.

Metrics	T1E10	T4E10	T2E5	T2E20	T2E10 (ours)
Demo $\mathcal{L} \downarrow$					
$\mathcal{S}_1$	0.015	0.018	0.016	0.011	**0.006**
$\mathcal{S}_2$	0.033	0.037	0.032	**0.031**	**0.031**
Overall	0.026	0.029	0.024	0.022	**0.020**
Rollout $\rho \uparrow$					
T1	0.667	0.722	0.611	0.778	**0.833**
T2	0.556	0.5	0.5	**0.722**	0.667
MoE Density					
	0.02	0.13	0.20	0.12	0.10

capacity and gradient sparsity per token, where k=1 removes the redundancy needed for stable specialization and k=4 over-mixes experts and dilutes the routing signal, whereas changing E only re-partitions an otherwise well-balanced expert pool. This suggests that the routing sparsity level is a more sensitive hyperparameter than the raw expert budget in our per-timestep MoE decoder.

6.10 MoE Design: MoE-Decoder vs. MoE-FFN

Setup. The dominant MoE recipe in large language models places experts inside the Transformer Feed-Forward Network (FFN) sublayer (Fedus et al., 2022; Lepikhin et al., 2020; Jiang et al., 2024; Dai et al., 2024; Zheng et al., 2026). We ask whether the same placement transfers to multimodal reward modeling by comparing two variants of SARM2's MMoE value decoder that are identical in every respect except where the inner MoE is inserted:

- **MoE-FFN** replaces the feed-forward sublayer of the last Transformer block of the value model backbone with a sparse MoE FFN; the per-timestep output head is a plain MLP.
- **MoE-Decoder (ours)** keeps the value model backbone fully dense and places the sparse MoE in the output head, acting on the fused-per-timestep representation produced by the encoder.

Both variants share the same SigLIP-2 features, the same 6-layer causal Transformer backbone, the same expert count E=10 and k=2, and the same primitive-group-conditioned MMoE outer gate (selected by the stage estimator's predicted primitive $\tilde{y}_t$, as in Section 3.1.2). The dense FFN sublayer of the value-model backbone follows the standard $4\times$ expansion, giving hidden width $4d_{\text{model}} = 4\times512 = 2048$. For MoE-FFN, each expert is a feed-forward block of hidden width 1024, and with k=2 activated per token the active hidden capacity is $1024\times2 = 2048$, which matches the dense FFN exactly, so the comparison is compute- and capacity-matched at the FFN sublayer. The inner top-k router is content-driven on whatever vector the placement hands it: for MoE-FFN that is one (modality, timestep) hidden state; for MoE-Decoder that is the fused timestep

vector concatenating all modalities. Primitive-group specialization is therefore not a differentiator; the only thing that changes between the two is the sublayer in which the experts live.

We report demo MSE on the $\mathcal{S}_1$ and $\mathcal{S}_2$ subsets (lower is better), rollout score ρ on T1 (Folding Shorts) and T2 (Cleaning Whiteboard) (higher is better), and the normalized top-k routing score $\mathcal{S}_{\text{route}}$ of Equation 6 as a routing-health / compute-parity check (below 0.4 is generally healthy).

Table 9: **MoE-Decoder vs. MoE-FFN on SARM2.** The only difference is the sublayer in which the inner MoE is placed; all other components, including the primitive-group-conditioned MMoE outer gate, are held identical. MoE-Decoder matches MoE-FFN's routing sparsity (MoE Density) yet reduces overall demo MSE by 39.4% and roughly triples rollout score on both tasks.

Metrics	**MoE-FFN**	**MoE-Decoder (ours)**
Demo $\mathcal{L} \downarrow$		
$\mathcal{S}_1$	0.026	**0.006**
$\mathcal{S}_2$	0.038	**0.031**
Overall	0.033	**0.020**
Rollout $\rho \uparrow$		
T1	0.222	**0.833**
T2	0.278	**0.667**
MoE Density		
	0.09	0.10

Findings. MoE-Decoder reduces overall demo MSE by 39.4% (0.020 vs. 0.033) and roughly triples rollout score on T1 (0.833 vs. 0.222) and T2 (0.667 vs. 0.278) at essentially identical routing sparsity. The result inverts the LLM intuition that MoE belongs in the FFN. The rest of this section explains why, structured around what each sublayer is for.

What each sublayer is for. A Transformer *FFN sublayer* performs position-wise representation refinement: attention mixes information across positions, and the FFN re-encodes each position in isolation. Stacking attention and FFN is the mechanism by which the value model backbone builds a single coherent fused-per-timestep representation from all modalities. The *output head* (which we call the decoder in this paper) consumes that fused representation and maps it to the predicted progress $\hat{r}_t$. Placing the inner MoE in the FFN therefore specializes *representation building*; placing it in the output head specializes *the prediction mapping*. These are very different things.

(i) MoE-FFN specializes the wrong stage. Per-position experts in the encoder FFN route each (modality, timestep) hidden state through a different MLP, and the following attention layer must re-fuse those divergently-processed positions back into a coherent representation. This fragments the backbone's job. Worse, the per-position vector the FFN router sees inherits its identity primarily from the modality projection it came through, not from scene content. Even when content-driven specialization does emerge, it acts on intermediate per-position representations that the prediction does not consume directly; $\hat{r}_t$ depends only on the final fused timestep representation, which is several attention layers downstream of the MoE.

(ii) MoE-Decoder specializes the right stage. With the value model backbone dense, all attention+FFN layers cooperate to produce one unified fused-per-timestep representation, as the backbone is designed to. The inner MoE then routes that fused vector to one of several scalar output paths, conditioned on its content and primitive-group gate. This is the standard "shared backbone, specialized head" pattern made soft and top-k: no representation is fragmented, and specialization lands on the axis that genuinely benefits from it. This makes different primitive groups (e.g., `pick`/`place`, `rotate`, `fold`) call for different progress computations on top of similar fused features. Critically, every routing decision in the output head is supervised directly by $\hat{r}_t$, so the routing gradient is high signal-to-noise.

(iii) Why this flips the LLM intuition. In language modeling, the per-position token *is* the prediction unit: each position predicts the next token from its own representation, so per-position MoE-FFN specialization aligns with the prediction axis, and the long context length ($L \sim 10^3 - 10^4$)

supplies enough routing decisions per sample to stabilize specialization. In multimodal reward modeling neither holds. The prediction unit is the fused timestep, not the per-modality position; the per-sample token budget is small ($L = (N{+}3)T$); and modality typing biases per-position routing toward modality-identity shortcuts. MoE-Decoder places specialization at the level where prediction actually happens, paying the sparsity cost where it buys differentiation rather than where it fragments shared representation.

Summary. The Transformer FFN is for building a unified representation; the output head is for mapping that representation to the prediction. Sparse routing helps when it specializes the right thing. In LLMs the FFN *is* the right thing because per-position prediction makes per-position specialization useful. In SARM2's MMoE value decoder the right thing is the output head: the value model backbone must stay coherent, and the head is where primitive-conditioned specialization pays off. MoE-Decoder matches placement to prediction; MoE-FFN does not. The numbers in Table 9 follow from this mismatch.

6.11 Reward Model Prediction Visualizations

Figure 8 compares timestep-level progress predictions with ground truth on all ten tasks in the classic $\mathcal{S}_1$ and unconventional $\mathcal{S}_2$ subsets. TOPReward and unadapted Robometer frequently saturate near completion too early. ReWiND and Robometer-FT better follow the trajectory but still drift on several long-horizon examples. Consistent with the aggregate MSE in Table 1, SARM2 tracks the reference curves most closely across the two subsets.

Figures 9 and 10 pair progress predictions with frames from policy rollouts. SARM2 rises during visible progress, remains flat during adjustment, and decreases after regressions or failures. The task-adapted Robometer baseline misses several of these local transitions, sometimes increasing while the robot is stalled or remaining flat while it advances. Because SPIRAL uses temporal value differences, these local errors can directly change the sign or magnitude of the policy-learning signal.

6.12 Action-Primitive and Expert-Routing Visualizations

The action-primitive estimator attains 85.22% micro-averaged accuracy across all timesteps. Table 10 reports the per-task result on ten held-out episodes, and Figures 11–12 pair representative primitive predictions with the resulting expert-routing pattern.

Primitive-corpus overlap. Eight of the ten evaluation tasks occur in the 200-hour primitive source corpus. Folding Shorts and Pick-and-Place into a Dish Rack do not occur in this corpus; the shared estimator achieves stage-estimation accuracies of 97.52% and 79.01% on these tasks, respectively (Table 10).

Table 10: **Action-primitive estimation accuracy on the ten reward model tasks.** Accuracy is measured over all timesteps in ten held-out episodes per task.

Task Name	Stage Estimator Accuracy (%)
Pick and place plates into bin	89.32
Pick and place plates into dish rack	79.01
Folding shorts	97.52
Folding the t-shirt	98.27
Pull plug off the socket	79.40
Clean whiteboard with whiteboard eraser	96.22
Set dinner table	70.85
Sweep paper scraps with broom	91.15
Coil and wrap headphones	97.72
Put away an umbrella	57.74

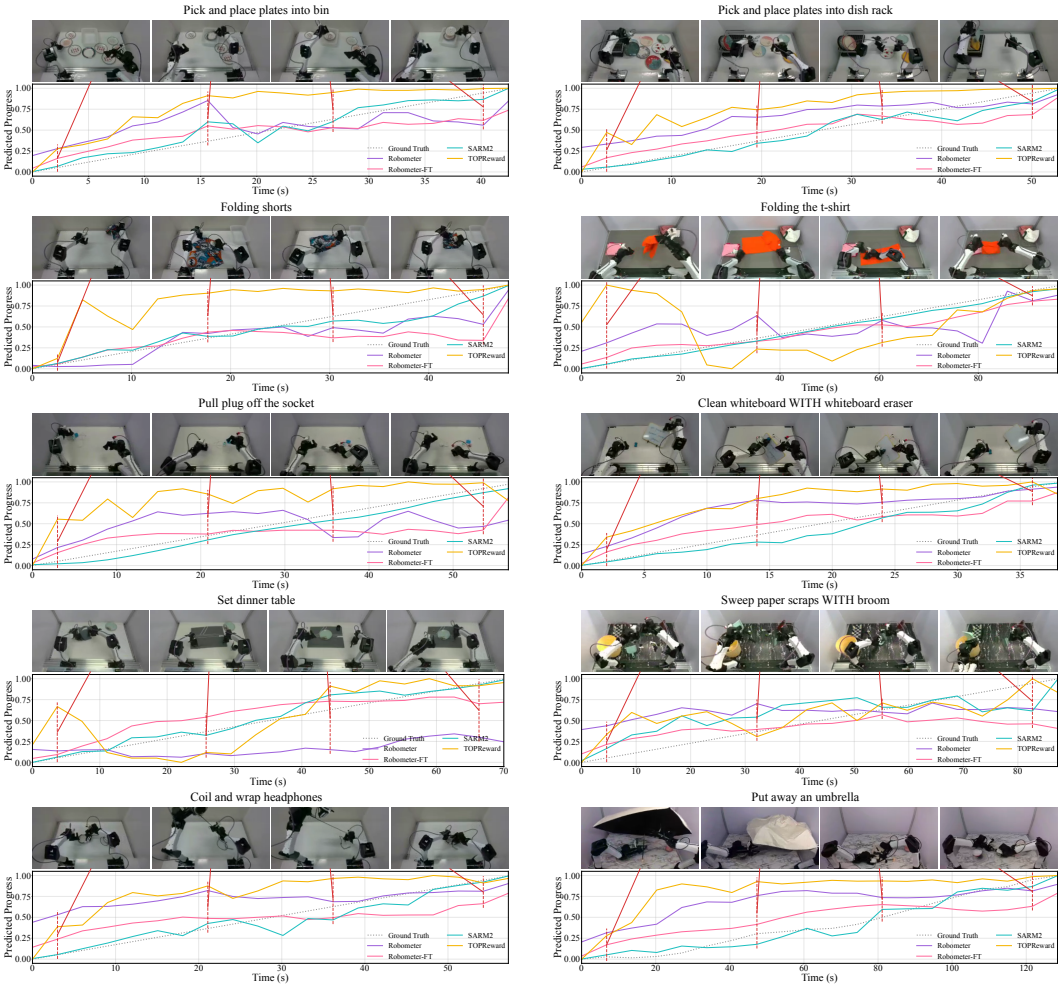


Figure 8: **Progress prediction on held-out demonstrations from all ten tasks.** Each panel overlays ground truth with TOPReward, Robometer, Robometer-FT, ReWiND, and SARM2. General-purpose VLM baselines often predict completion prematurely, whereas SARM2 more closely follows the reference trajectory. For clear comparison, we convert SARM2's prediction to progress as $P_t = 1 + \hat{V}_t \in [0, 1]$, same for the following figures.

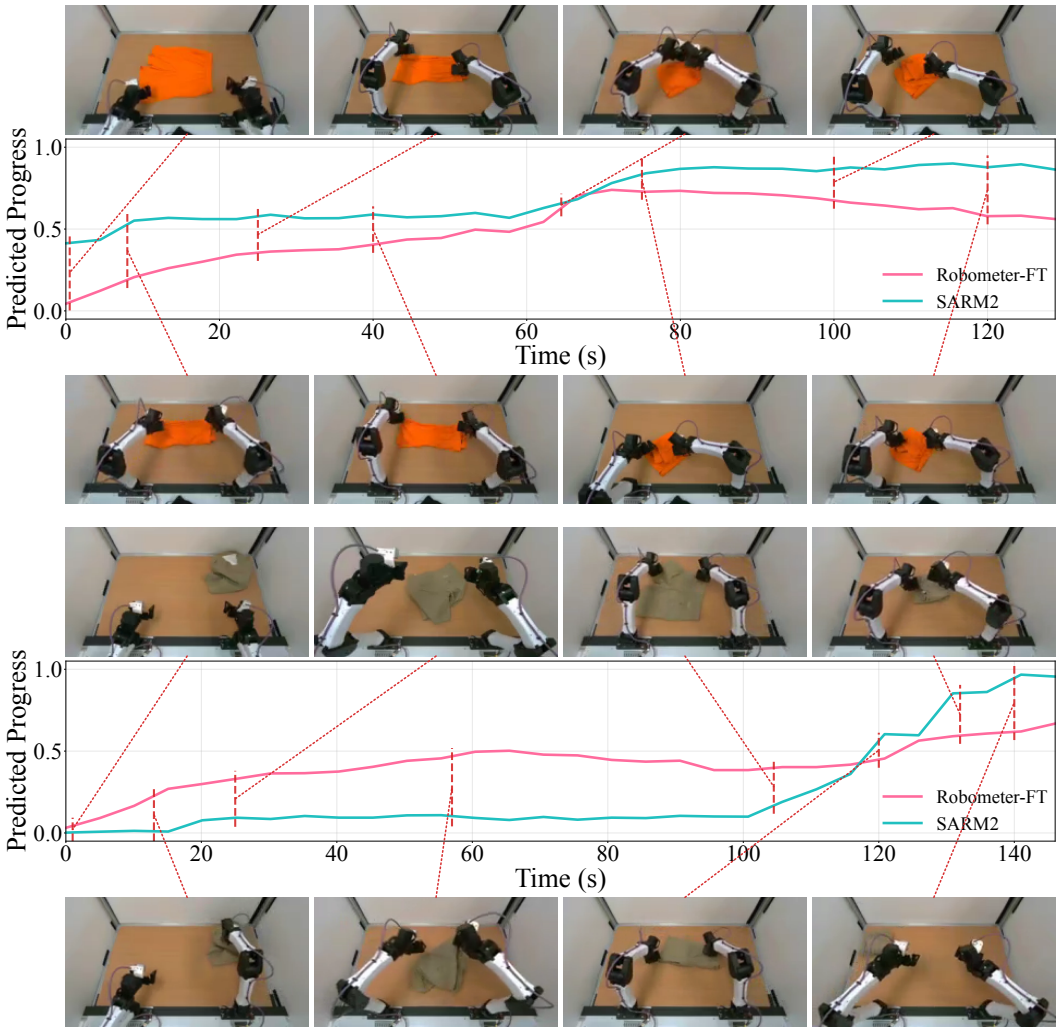


Figure 9: **Progress prediction on two Folding Shorts rollouts.** Eight frames provide visual context for each trajectory. SARM2 follows intervals of progress and struggle more closely than the task-adapted Robometer baseline.

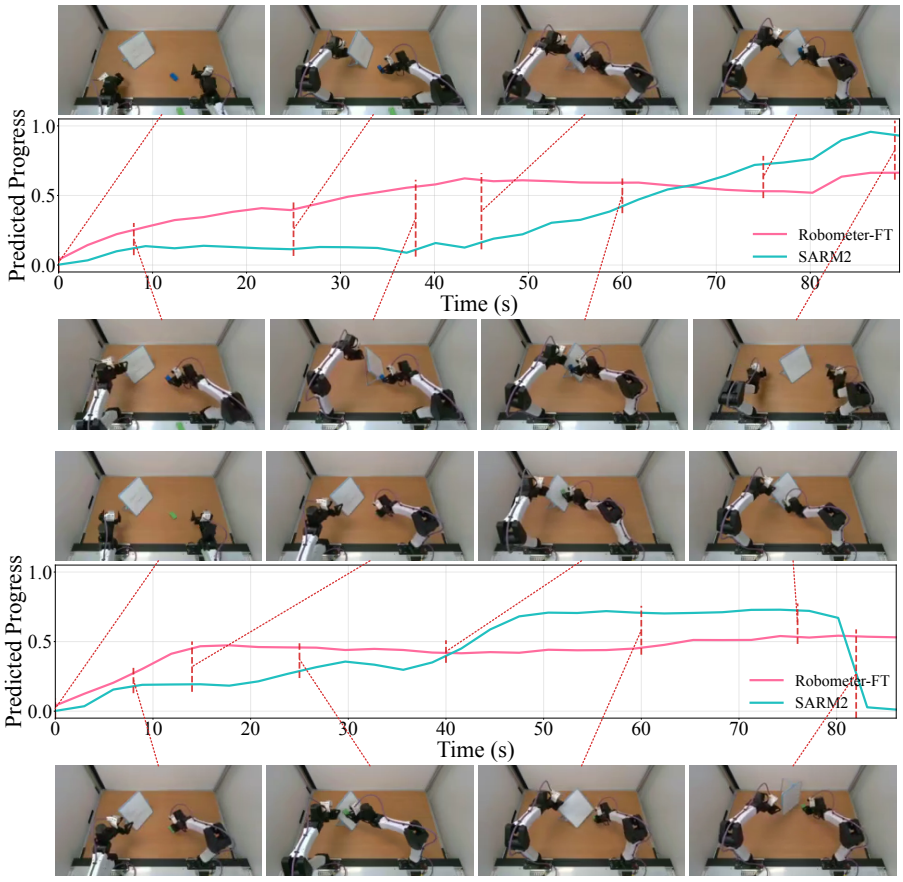


Figure 10: **Progress prediction on two Cleaning Whiteboard rollouts.** Eight frames provide visual context for each trajectory. SARM2 reflects progress, adjustment, and failure more faithfully than the task-adapted Robometer baseline.

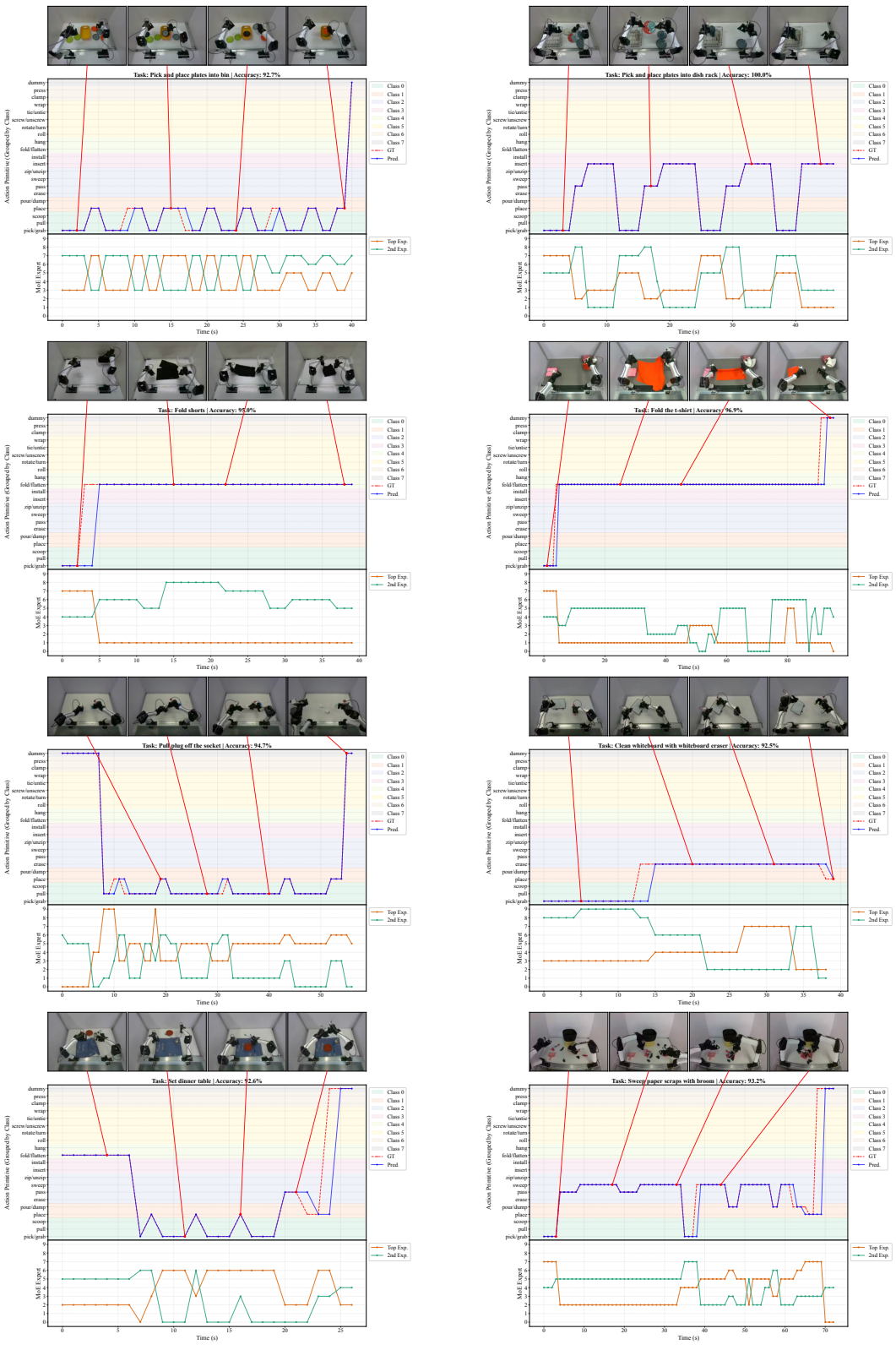


Figure 11: **Action-primitive predictions and expert routing (part 1).** Each task shows representative frames, predicted and ground-truth primitives, and the active MMoE experts over time. Colors follow the groups in Appendix 6.4.

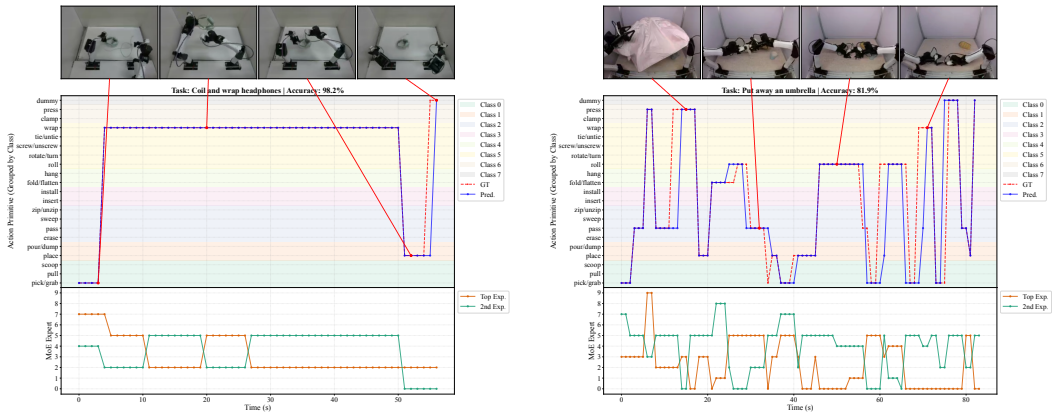


Figure 12: **Action-primitive predictions and expert routing (part 2).** Each task shows representative frames, predicted and ground-truth primitives, and the active MMoE experts over time. Colors follow the groups in Appendix 6.4.

6.13 Auxiliary Formulae

Let $\bar{p}_e^{(m)}$ denote the average routing probability assigned to expert e by gate m over a batch, and $f_e^{(m)}$ the fraction of tokens for which expert e is selected in the top-k. Following (Huang et al., 2024),

$$\mathcal{L}_{\text{balance}} = \sum_{m=1}^{M} E \cdot \sum_{e=1}^{E} f_e^{(m)} \cdot \bar{p}_e^{(m)}, \qquad \mathcal{L}_{\text{entropy}} = -\sum_{m=1}^{M} \sum_{e=1}^{E} \bar{p}_e^{(m)} \log \bar{p}_e^{(m)}. \tag{7}$$

Balance encourages uniform expert utilization within each gate; entropy prevents routing distributions from becoming overly peaked early in training.

6.14 Lightweight One-Time SARM2 Calibration

We collect approximately 100 initial rollouts $\mathcal{R}_1$ from the behavior-cloned policy π_1, whose failures expose states not present in the demonstration trajectories. An annotator partitions each rollout into segments, assigns one of four qualitative labels, and selects the final task progress from predefined options specified by the task's scoring protocol (e.g., the Cleaning Whiteboard progress tiers in Appendix 6.7); this process does not collect corrective actions, new demonstrations, or manual frame-level labels. The fixed rule assigns relative value slopes of 2, 1, 0, and -2 to `fast progress`, `slow progress`, `adjust`, and `mistake`, respectively. The annotator's final progress $P_T \in [0, 1]$ is converted to the reward model's value convention as $V_T = P_T - 1$. A script then expands the segment annotations to timestep-level value targets and uses V_T as the endpoint anchor for the integrated sequence. The resulting rollout set is mixed with a 50% subsample of the original reward model training data before fine-tuning RM_1 into RM_2. Here $\mathcal{D}_{\mathrm{RM}} \subset \mathcal{D}_{\mathrm{demo}}$; the retained data prevents the small rollout set from erasing the original multitask representation. Fine-tuning uses roughly one tenth of the reward model pretraining compute and takes approximately one hour per task in our setup.

Annotating the 100-rollout set takes fewer than two human hours per task. This is a one-time, offline task-specific cost: RM_2 is fixed after calibration, and later policy-improvement rounds require no additional progress annotations. Intervention-based training demands sustained attention and skilled teleoperation: operators must closely monitor robot motion, choose precisely when to take over, and execute accurate corrective actions that recover from errors and advance the task. SPIRAL reduces these demands by using lightweight one-time offline annotation of recorded rollouts in place of continuing online corrective interventions.

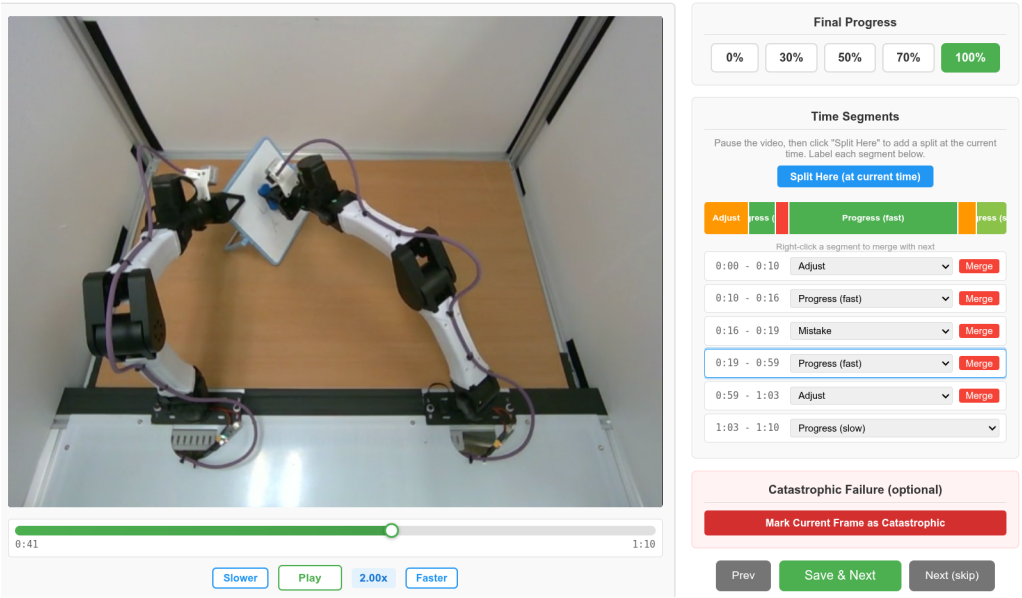


Figure 13: **Annotation interface for lightweight one-time SARM2 calibration.** The annotator segments each rollout, selects one of four progress labels, and provides final task progress without supplying corrective actions or new demonstrations. Per task, annotating approximately 100 rollouts takes fewer than two hours and reward model fine-tuning takes approximately one hour.

6.15 DICE-RL Backend Details

This backend directly inherits the distribution-contractive design of DICE-RL (Sun & Song, 2026). The behavior-cloned flow policy remains fixed, and a residual actor predicts bounded corrections to its action chunks. We retain the action-chunk critic, a five-member critic ensemble, target-policy smoothing, and multi-sample expectation training over $\kappa = 4$ latent samples per observation. Network sizes and optimization settings are listed in Table 4.

DICE-RL action-chunk targets. Let $Q_\phi(\mathbf{o}_t, \mathbf{a}_t)$ denote the critic for the action chunk $\mathbf{a}_t$ of length h, and let $\bar{Q}$ denote its target network. The TD target uses the chunk reward from Equation 1 and the successor observation $\mathbf{o}_{t+h}$, while the MC target uses the sparse reward R_T^{sparse} derived from the episode's success or failure. With a per-control-step discount γ and an episode-termination indicator $d_{t,h}$, the targets and losses are

$$\begin{aligned} y_{t,h}^{\text{TD,dense}} &= r_{t,h}^{\text{dense}} + \gamma^h (1 - d_{t,h}) \bar{Q}(\mathbf{o}_{t+h}, \mathbf{a}_{t+h}), \\ y_t^{\text{MC,sparse}} &= \gamma^{T-t} R_T^{\text{sparse}}, \\ J_{\text{TD}}^{\text{dense}} &= \mathbb{E}_{\mathcal{D}} \left[\left(Q_\phi(\mathbf{o}_t, \mathbf{a}_t) - y_{t,h}^{\text{TD,dense}} \right)^2 \right], \\ J_{\text{MC}}^{\text{sparse}} &= \mathbb{E}_{\mathcal{D}} \left[\left(Q_\phi(\mathbf{o}_t, \mathbf{a}_t) - y_t^{\text{MC,sparse}} \right)^2 \right]. \end{aligned} \tag{8}$$

Here $\mathbf{a}_{t+h}$ denotes a target-policy action chunk at the successor observation, and $d_{t,h} = 1$ when the current chunk ends the episode, removing the bootstrap term. For a truncated final chunk, h is replaced by its executed length in both the successor index and the discount exponent. The dense TD term assigns credit to action chunks, while the sparse MC term regularizes the critic against errors in dense reward estimates. DICE-RL trains Q_ϕ with the weighted critic loss

$$J_{\text{value}} = J_{\text{TD}}^{\text{dense}} + \alpha J_{\text{MC}}^{\text{sparse}}. \tag{9}$$

The weighted loss is applied to each critic ensemble member, with $\alpha = 0.5$. For multi-sample expectation training, we estimate the TD bootstrap value by averaging over $\kappa = 4$ target-policy chunks, with clipped Gaussian target-policy smoothing as in TD3.

Policy improvement. Let $\pi_1(\mathbf{o}_t, z)$ denote the frozen behavior-cloned flow policy and $s_\theta(\mathbf{o}_t, z)$ the trainable residual actor, where $z \sim \mathcal{N}(0, I)$ is a latent sample. During actor optimization, the corrected action chunk is $\mathbf{a}_t = \pi_1(\mathbf{o}_t, z) + s_\theta(\mathbf{o}_t, z)$, and we minimize the TD3+BC-style objective

$$\min_\theta \mathbb{E}_{\substack{\mathbf{o}_t \sim \mathcal{D} \\ z \sim \mathcal{N}(0,I)}} \left[-Q_\phi(\mathbf{o}_t, \mathbf{a}_t) + \beta \left\| s_\theta(\mathbf{o}_t, z) \right\|_2^2 \right]. \tag{10}$$

The critic term encourages corrections that increase predicted return, while the penalty with $\beta = 30$ keeps the corrected chunks close to the base policy. Only the residual actor is optimized in this step; the base policy and critic parameters remain fixed. Following DICE-RL (Sun & Song, 2026), we sample $\kappa = 4$ latent noises per observation to form diverse candidate chunks from the frozen flow prior and residual actor. During training rollouts, critic-guided best-of-κ selection steers execution toward high-value candidates, providing a structured exploration space around the pretrained policy. The actor objective averages over these candidates to improve the residual policy across the latent-induced action distribution. At deployment, for fast inference speed, we sample a single latent $z \sim \mathcal{N}(0, I)$ and execute $\mathbf{a}_t = \pi_1(\mathbf{o}_t, z) + s_\theta(\mathbf{o}_t, z)$, without evaluating the critic.

The matched sparse-reward DICE-RL rows in Table 2 use this same backend, policy prior, and rollout schedule while removing SARM2 dense rewards. They therefore isolate the reward source within the DICE-RL optimizer. The SARM2-guided offline and rollout rows show the effect of adding SPIRAL's iterative data loop to the offline warm start.

6.16 AWR Backend Details

Our AWR backend uses a binary advantage filter to select action chunks for fine-tuning the VLA's action head. It draws on advantage-weighted regression (Peng et al., 2019) and the policy-extraction formulation of $\pi^*_{0.6}$ (Physical Intelligence et al., 2025), while using SARM2 as the value estimator. Let π_{ref} denote the reference policy, $\hat{V}_{\text{SARM2}}(\mathbf{o}_t, \ell) \in [-1, 0]$ the normalized remaining-steps value, $\mathbf{a}_t$ the action chunk spanning h control steps, and ℓ the task instruction. For policy learning, we use the shifted value

$$\bar{V}_t = 1 + \hat{V}_{\text{SARM2}}(\mathbf{o}_t, \ell) \in [0, 1].$$

Mixed advantage over an action chunk. To regularize policy updates against errors in the learned dense value estimates, we combine the chunk's progress gain with an MC advantage term derived from sparse rewards. For a complete chunk, the successor observation is $\mathbf{o}_{t+h}$; for a chunk truncated by episode termination at T, replace $t + h$ by T. The dense reward is the change in SARM2 value over the chunk, and the sparse MC return is the discounted episode reward:

$$\begin{aligned} r^{\text{dense}}_{t,h} &= \bar{V}_{t+h} - \bar{V}_t, \\ G^{\text{sparse}}_t &= \gamma^{T-t} R^{\text{sparse}}_T. \end{aligned} \tag{11}$$

Here R^{sparse}_T is the sparse reward derived from the success/failure outcome recorded by the safety monitor before the next environment reset, and γ denotes a per-control-step discount. Using the shifted SARM2 value as a fixed baseline, the MC advantage term and mixed filtering score are

$$\begin{aligned} A^{\text{MC}}_t &= G^{\text{sparse}}_t - \bar{V}_t, \\ \widetilde{A}_{t,h} &= \frac{r^{\text{dense}}_{t,h} + \lambda^{\text{AWR}}_{\text{MC}} A^{\text{MC}}_t}{1 + \lambda^{\text{AWR}}_{\text{MC}}}. \end{aligned} \tag{12}$$

The dense term rewards local progress, while the MC term compares the observed episode return with the fixed baseline to account for eventual task success or failure. The nonnegative coefficient $\lambda^{\text{AWR}}_{\text{MC}}$ controls their relative contribution and is distinct from the DICE-RL critic-loss coefficient α. This is a progress-based advantage surrogate for filtering, rather than a TD residual that adds a second copy of the value difference. It requires no separate learned value network or bootstrap term beyond the episode endpoint.

Filtering policy updates. We use the binary improvement indicator to select which action chunks contribute to the VLA's flow-matching loss:

$$\begin{aligned} p(I \mid \widetilde{A}_{t,h}) &= \delta(\widetilde{A}_{t,h} > \epsilon_\ell), \\ w_t &= \delta(\widetilde{A}_{t,h} > \epsilon_\ell), \\ \mathcal{L}_{\mathrm{AWR}}(\theta) &= \mathbb{E}_{(\mathbf{o}_t, \mathbf{a}_t, \ell) \sim \mathcal{D}} \left[w_t \, \mathcal{L}_{\mathrm{FM}}(\theta; \mathbf{o}_t, \mathbf{a}_t, \ell) \right]. \end{aligned} \tag{13}$$

Only the action-head parameters θ are optimized; the vision-language backbone remains frozen. Here $\delta(\cdot) \in \{0, 1\}$ is the indicator function, and $\epsilon_\ell = 0$ for both downstream tasks. Action chunks with $I = 0$ contribute zero regression loss, and the policy receives no advantage-conditioning input. Using the policy-extraction notation of $\pi^*_{0.6}$, the target distribution can be written as

$$\hat{\pi}(\mathbf{a} \mid \mathbf{o}, \ell) \propto \pi_{\mathrm{ref}}(\mathbf{a} \mid \mathbf{o}, \ell) \left(\frac{\pi_{\mathrm{ref}}(\mathbf{a} \mid I = 1, \mathbf{o}, \ell)}{\pi_{\mathrm{ref}}(\mathbf{a} \mid \mathbf{o}, \ell)} \right)^{\beta_{\mathrm{AWR}}}. \tag{14}$$

With $\beta_{\mathrm{AWR}} = 1$, this reduces to the selected-data distribution $\pi_{\mathrm{ref}}(\mathbf{a} \mid I = 1, \mathbf{o}, \ell)$, which our update fits by filtering the training samples. The same filtering rule applies to demonstrations and policy-generated rollouts. The MC mixing weight, learning rate, batch size, and update count are listed in Table 4.

6.17 Additional Qualitative Discussion

Beyond error correction. Labeling rollouts does more than fix erroneous behaviors and recover from OOD states: it implicitly steers the policy toward the most efficient and stable solution strategies, aligning execution style with human preferences. Figure 14 illustrates this on Cleaning Whiteboard. The top frames show the canonical strategy: the left arm grasps the top corner of the board to hold it rigid, while the right arm wipes with the eraser under a stable contact. The bottom frames show a failure mode of the initial BC policy: the left arm instead grips the lower bar of the stand, an unstable support such that every wiping stroke tilts the board and lets it spring back, making cleaning slow and error-prone. This grasp does not appear in the demonstration set; it is an OOD behavior. Strictly speaking, it is not a mistake. Progress can still increase (the board does get cleaner), but the motion is inefficient and risks toppling the whiteboard entirely. We therefore label such segments as "adjust" with zero reward, injecting human preference and high-level prior knowledge into RM_2 via fine-tuning. After the SPIRAL loop, the policy distills this preference and no longer adopts the unstable grasp.

The crumpled-shorts setting remains difficult. Folding Shorts from a crumpled initial state remains the least saturated policy condition, reaching 66.7% success over 30 trials after three SPIRAL rounds in Table 2. The task begins with a long, weakly structured flattening phase in which cloth appearance varies substantially and multiple attempts may be required. The 20 hours of demonstrations are unfiltered, span roughly 40–100 seconds per trajectory, and contain operator-dependent grasp preferences. The reward model can consequently be uncertain about whether early flattening attempts improve the cloth state, yielding flat or noisy dense rewards. This setting exposes a real boundary of the current approach: subtle deformable-state changes may require higher-quality demonstrations, more targeted reward supervision, or representations designed specifically for cloth geometry.

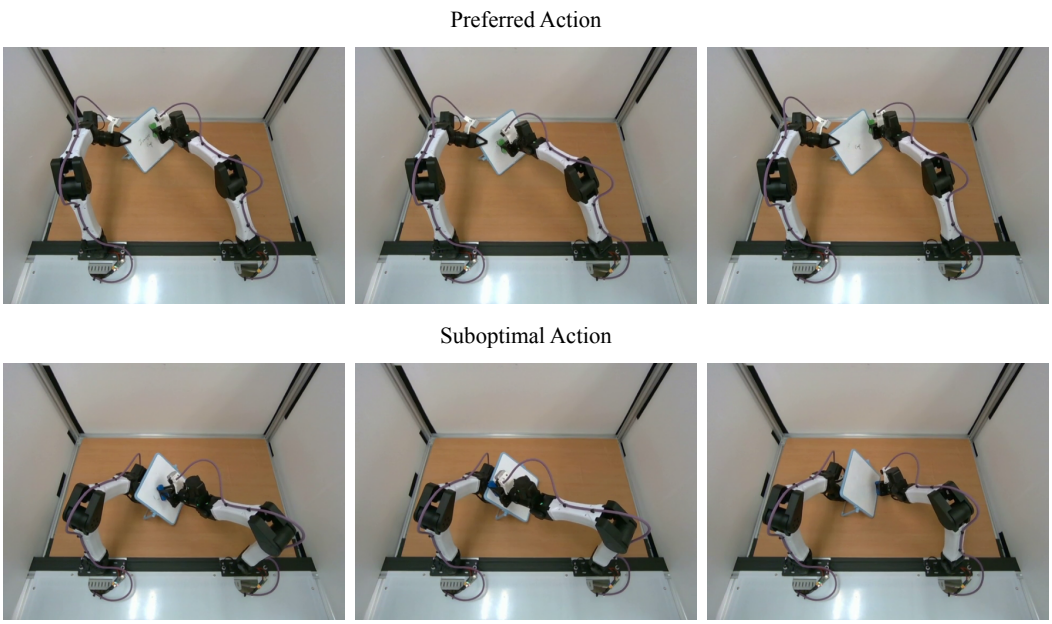


Figure 14: **Qualitative policy preference on Cleaning Whiteboard.** Top: a stable post-training strategy that supports the board near its upper corner. Bottom: an inefficient initial-policy strategy that grips the lower stand and allows the board to tilt during wiping.

6.18 Policy Rollout Examples

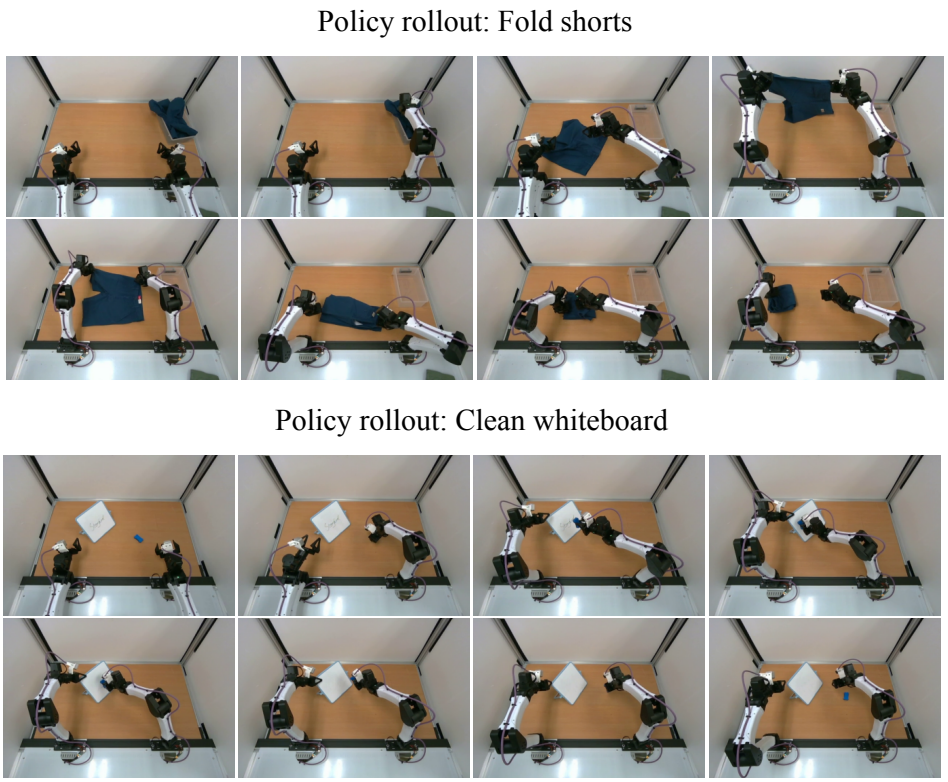


Figure 15: **Example policy rollouts.** Top: Folding Shorts. Bottom: Cleaning Whiteboard.