TRAJECTORY-CALIBRATED MERGING OF EMBODIED POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Robot foundation models are increasingly adapted into task specialists to expand their embodied capabilities. Yet these capabilities remain fragmented across separate policies, and existing model-merging methods do not reliably preserve them during closed-loop execution: parameter-space alignment provides only an indirect proxy for behavior, while regression may fail to preserve behavior at states not covered by its calibration inputs. We present Trajectory-Calibrated Regression (TCR), an execution-aware, gradient-free framework that consolidates robot specialists into a single policy. Across two complementary timescales, TCR collects calibration inputs from expert rollouts spanning different stages of task execution and from intermediate states within iterative action generation. To account for representation shifts introduced by merging, TCR recomputes each block's input through the already merged blocks. It then uses each expert's computation within the block to form module inputs and fits merged weights to match the experts' linear outputs. A two-pass calibration procedure further refines the merged policy using separate rollout caches while retaining the frozen specialists as teachers. The resulting model preserves the original inference graph and requires neither expert routing nor additional gradient training. On LIBERO, TCR merges four $\pi_{0.5}$ specialists and achieves **77.83%** success across 1,200 evaluation episodes, outperforming FeatCal, RegMean, and direct weight averaging by 9.50, 24.17, and 33.00 percentage points, respectively. We also find that continued training from a merged checkpoint improves peak success: TCR reaches **97.75%** versus 94.00% when training from the common base. These results show that calibrating model merging around execution dynamics and evolving internal representations provides an effective path toward preserving diverse embodied skills in a unified policy. Code, experimental results, and visualizations are available at `https://anonymous.4open.science/r/vla-merge-6333/README.md`.

1 INTRODUCTION

General-purpose robots need policies that can perform diverse physical tasks. Vision-language-action models (VLAs) offer a promising foundation (Black et al., 2025a; Kim et al., 2025a), but demonstration data cover limited tasks and environments, making it difficult to acquire a broad skill set through a single training run. Training task specialists from a shared foundation model expands coverage, but leaves their capabilities in separate checkpoints. We seek to consolidate them into one policy.

Model merging can do this without joint training or selecting an expert at inference. We focus on post-hoc consolidation of independently trained VLA specialists into one static policy, without redesigning the experts or adding a task router. Parameter-space methods combine weights or align task updates to reduce interference (Wortsman et al., 2022; Yadav et al., 2023; Stoica et al., 2025). But the mapping from weights to actions is nonlinear: reducing conflicts between weights does not ensure that the merged policy preserves expert actions or closed-loop success. Regression-based methods instead fit merged weights to expert outputs on calibration inputs (Jin et al., 2023; Nguyen et al., 2026). Such a fit is only as useful as the inputs it covers. An action error can change not only the current output but also what the policy sees next; matching a fixed set of expert observations may leave later inputs unconstrained and allow errors to accumulate.

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.

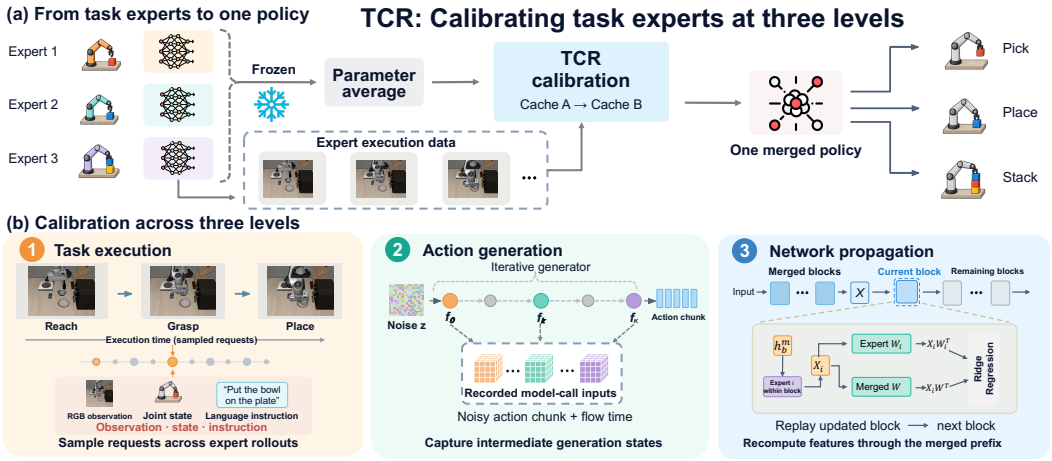

Figure 1: Overview of TCR. (a) Two-pass calibration starts from the parameter average and uses separate rollout caches (A/B) from the same frozen experts, yielding one fixed policy. (b) Calibration covers task execution, intermediate action-generation calls, and blockwise replay. Blockwise calibration recomputes merged-prefix features, forms expert-local module inputs for ridge regression, and replays each updated block before proceeding.

These dependencies arise at three levels. During a task, a slightly misplaced grasp can change the observation and robot state at the next decision. Within an iterative action generator, intermediate predictions determine inputs to later calls, so matching only the final output may miss states used to produce it. Across network blocks, merging early layers changes the features fed to later layers; a downstream block fitted only on the original expert features may no longer see those features after merging. Thus, the choice of calibration inputs matters alongside the regression solver itself (Figure 1).

We introduce Trajectory-Calibrated Regression (TCR), a gradient-free framework for merging compatible robot experts. TCR uses frozen experts as teachers and collects two kinds of inputs: observations, robot states, and instructions across expert rollouts, and intermediate call states from each expert's native action generator. The first covers stages reached after earlier expert actions; the second covers generator inputs, such as noisy actions and flow times, that an observation alone does not specify. TCR fits adapted perception, conditioning, and action modules to expert outputs on these inputs. Following blockwise feature replay (Nguyen et al., 2026), it recomputes each block's inputs through the blocks already merged before fitting that block. Downstream regressions therefore use features produced by the merged prefix rather than relying only on the original expert features. TCR makes two passes over separate rollout caches: the first updates the parameter average, and the second uses that checkpoint as its prior while still matching the frozen experts. The second cache adds expert calls not used in the first fit, and the updated prior carries the first pass into the second. The result is one policy with the original inference graph, no expert router, and no gradient training.

We evaluate TCR on LIBERO (Liu et al., 2023) using four $\pi_{0.5}$ specialists trained for 10,000 steps each. Across 1,200 episodes, TCR achieves **77.83%** success, the highest among the static post-hoc mergers evaluated in this $\pi_{0.5}$–LIBERO setting. This is 9.50 points above FeatCal (68.33%), 24.17 above RegMean (53.67%), and 33.00 above direct weight averaging (44.83%). The gains are strongest on Goal and Long, although FeatCal remains ahead on Spatial. All methods share the same experts and evaluation protocol, while retaining their own calibration budgets. Ablations test the calibration choices; real-robot and cross-benchmark evaluations test TCR beyond LIBERO. We also find that merged checkpoints are useful starts for joint training: all four attain higher peak success than common-base training, with TCR highest at **97.75%** versus 94.00% for the base.

Our contributions are:

- We sample expert rollouts and intermediate generator calls, covering both later task states and the inputs to an iterative action generator.

- We combine trajectory calibration with merged-prefix replay to fit adapted perception, conditioning, and action modules without gradients, producing one policy with the original inference graph and no expert router.
- On $\pi_{0.5}$–LIBERO, TCR reaches 77.83% success, 9.50 points above the strongest baseline. Ablations test the calibration choices, while real-robot and cross-benchmark studies evaluate TCR beyond LIBERO.

2 Related Work

2.1 Combining Expert Parameters

Some merging methods combine parameter updates without explicitly matching model outputs. Model Soups averages fine-tuned checkpoints; Task Arithmetic scales and combines updates from a common base; TIES and DARE address coordinate conflicts and redundancy (Wortsman et al., 2022; Ilharco et al., 2023; Yadav et al., 2023; Yu et al., 2024). Breadcrumbs and DELLA refine coordinate selection, while TSV, Iso-CTS, Subspace Boosting, and DC-Merge operate on singular directions or update geometry (Davari & Belilovsky, 2025; Deep et al., 2024; Gargiulo et al., 2025; Marczak et al., 2025; Skorobogat et al., 2025; Zhang et al., 2026). KnOTS aligns independently trained LoRA updates, and WUDI and its variants estimate useful input subspaces from weights (Stoica et al., 2025; Cheng et al., 2025; Wei et al., 2026). LoRA-specific and base-anchored methods further constrain update geometry (Lee et al., 2026; Ryum & Kang, 2026; Zheng et al., 2025; Shi et al., 2026).

2.2 Calibration-Guided Functional Merging

RegMean and RegMean++ fit merged linear weights to expert outputs, with the latter recomputing features through already merged layers (Jin et al., 2023; Nguyen et al., 2026). LOT Merging studies feature drift during model merging and introduces layer-wise task vector fusion to reduce representation mismatch (Sun et al., 2025). ESM uses task-induced layer-wise feature shifts to extract essential subspaces, while activation-guided coefficient methods optimize a more restricted merge parameterization (Li et al., 2026; Yao et al., 2025). Fisher merging and AdaMerging additionally use gradient information or optimization on calibration data (Matena & Raffel, 2022; Yang et al., 2024).

FeatCal performs closed-form feature calibration in forward order, refreshing features after preceding layers are updated (Gu et al., 2026). TCR builds on regression and sequential feature replay, focusing on calibration inputs collected across robot task execution and action generation.

2.3 Robot Policies and Specialist Consolidation

OpenVLA and the π family connect pretrained visual–linguistic representations to robot actions (Kim et al., 2025b; Black et al., 2025b;a). Action chunking predicts several actions per policy request, while diffusion and flow policies generate these chunks through repeated prediction steps (Zhao et al., 2023; Chi et al., 2023; Lipman et al., 2023); SmolVLA provides a smaller flow-based implementation (Shukor et al., 2025). OpenVLA-OFT instead decodes continuous actions in parallel (Kim et al., 2025a).

Adapting calibration to a different policy architecture requires matching its actual inference computation.

Combining robot skills. FLEET-MERGE studies policy merging for robot fleets by aligning separately trained policies under parameter permutation symmetries (Wang et al., 2024). MergeVLA explores robot skill consolidation by modifying expert construction with sparse masks to encourage mergeability (Fu et al., 2026). OSRM studies LoRA mergeability in language models through structured adapter subspaces (Zhang & Zhou, 2025). TCR instead starts with existing compatible experts and fits merged weight matrices using inputs recorded during task execution. CapVector transfers capability vectors for subsequent adaptation, while VLA-Pro retrieves and dynamically fuses adapter memories at inference time (Song et al., 2026; Si et al., 2026). RETAIN interpolates generalist and fine-tuned VLA parameters and uses the merged checkpoint to initialize adaptation to

a subsequent robot task (Yadav et al., 2026). In language models, Mashup Learning merges historical checkpoints into an initialization for further fine-tuning (Vaina et al., 2026). Our continuation study instead jointly trains the original source tasks after merging four specialists under matched updates. ReVLA restores visual generalization after adaptation through representation recovery (Dey et al., 2025).

The dependence of future inputs on current predictions is also central to imitation learning (Ross et al., 2011). Policy distillation further studies transferring behaviors from expert policies by matching student predictions to teacher outputs on expert-generated data (Rusu et al., 2016). TCR does not perform dataset-aggregation policy learning; instead, it also uses expert-generated inputs and fits the merged weights through fixed regression rather than training a student policy.

3 Trajectory-Calibrated Regression

TCR collects frozen expert inputs across task execution and action generation, then fits modules with merged-prefix replay (Figure 1). Full parameters are Θ, module weights W, finite caches $\mathcal{D}$, and input distributions μ; superscript M denotes merged quantities.

3.1 Experts, Output Policy, and Calibration Objective

Given M experts $\pi_i \equiv \pi_{\Theta_i}$ trained independently from the same initial model, our goal is one policy that retains their skills without selecting an expert at inference. Experts share an architecture, tokenizer, observation and action formats, and compatible normalization; the merged policy retains language conditioning.

Matching expert outputs provides a general objective for functional merging:

$$\Theta^\star = \arg\min_{\Theta} \sum_{i=1}^{M} \mathbb{E}_{x\sim\mu_i}\, \mathcal{L}\big(f_\Theta(x), f_{\Theta_i}(x)\big). \tag{1}$$

Here μ_i is the matching-input distribution and $\mathcal{L}$ is the output discrepancy. The x in Eq. (1) denotes a generic matching input; below, we distinguish request-level x_r from call-level $\tilde{x}_{r,f}$. For robot policies, actions and intermediate predictions affect later inputs, so the deployment distribution depends on the policy:

$$\mu_{\text{deploy}} = \mu_{\text{deploy}}(\Theta). \tag{2}$$

TCR approximates this objective by fitting local linear outputs on a finite set of expert execution inputs, rather than optimizing over the merged policy's deployment distribution.

Let $W_{i,b,j}$ denote expert i's weight for linear module j in block b. We fit full linear weight matrices across adapted perception, conditioning, and action modules, including directly trained interfaces and biases. Identical parameters remain unchanged; other adapted tensors follow explicit fallback rules (Appendix A.1).

3.2 Collecting Calibration Inputs

Across task execution. To include inputs reached after earlier expert actions, we sample from complete expert episodes. Write $x_r = (o_r, s_r, \ell)$ for the visual observation, proprioceptive state, and task instruction at request r. Let $\mathcal{H}_r$ contain retained policy history. Expert i's native generator produces the action chunk $\mathbf{a}_r$ and updates this history:

$$(\mathbf{a}_r, \mathcal{H}_{r+1}) = \mathcal{G}_{\Theta_i}(x_r, \mathcal{H}_r; \xi_r), \qquad x_{r+1} \sim \mathcal{P}_{\text{env}}\big(\cdot \mid x_r, \mathrm{Exec}(\mathbf{a}_r)\big). \tag{3}$$

Here ξ_r denotes sampling randomness, and Exec selects the actions actually executed from the predicted chunk. Here $x_{r+1} = (o_{r+1}, s_{r+1}, \ell)$, with ℓ fixed within the episode; the transition notation abbreviates the underlying environment dynamics, not a Markov assumption about observations. We sample requests throughout each expert's assigned-task episodes, retaining all inputs needed for replay.

Within action generation. An observation alone does not specify every input used by an iterative action generator. A calibration record $\tilde{x}_{r,f} = (x_r, \mathcal{H}_r, \mathbf{u}_{r,f}, f)$ contains the inputs needed to reconstruct an actual model call. The internal state $\mathbf{u}_{r,f}$ and call index f exist only where the generator uses them. Within each sampled request, we retain selected calls from the expert's native generation procedure.

For $\pi_{0.5}$, $\mathbf{u}_{r,f}$ includes a noisy action chunk and its flow time. Other generators can have different inputs: a parallel action decoder has no denoising coordinate, while a video-generating policy may require additional visual latents and history. Appendix A.6 outlines the checks needed before extending TCR beyond the evaluated $\pi_{0.5}$ model.

Sampling and budget. Let μ_i^E denote the call-input distribution induced by expert π_i under the sampling rule, and $\mathcal{I}_i^E$ its retained request–call index pairs. Equal expert sampling defines the mixture and finite caches:

$$\mu^E = \frac{1}{M}\sum_{i=1}^{M} \mu_i^E, \qquad \mathcal{D}_i^E = \{\tilde{x}_{r,f}^{(i)} : (r,f) \in \mathcal{I}_i^E\}. \tag{4}$$

We divide the samples by task, episode, position within the episode, and action generation step. We do not keep only successful episodes. For iterative models, each stored observation remains paired with the intermediate states generated from it.

The calibration budget counts feature rows $n_{i,b,j}$ actually used per expert and module, rather than stored calls, whose sequence lengths and branches may differ. We allocate row quotas across sample groups without duplicating scarce samples (Appendix A.3). Temporal order determines input collection, but shuffling the resulting input–teacher pairs leaves the regression objective unchanged.

3.3 Fitting the Merged Policy

Recompute features before fitting each block. Earlier merged layers can produce different features from the experts. Following sequential replay (Nguyen et al., 2026), we visit blocks in forward order, recomputing each block's inputs through the already merged prefix. Let $h_b^{\mathrm{M}}(\tilde{x})$ denote this input to block b for record $\tilde{x}$. Within the current block, we still use each expert's computations before the linear module being fitted; denote these computations by $\phi_{i,b,j}$. Its regression inputs are

$$X_{i,b,j} = \mathrm{Rows}\left\{\phi_{i,b,j}\big(h_b^{\mathrm{M}}(\tilde{x})\big) : \tilde{x} \in \mathcal{D}_i^E\right\}. \tag{5}$$

We solve the block's linear modules on these fixed features, write their merged weights, and recompute the block outputs before proceeding.

When the network has several branches, a component is processed after the components that supply its inputs. Conditioning representations, cross-attention keys and values, and time embeddings are refreshed when their producers change. The evaluated $\pi_{0.5}$ implementation covers 418 linear modules and 422 adapted tensors (Appendix A.4); these counts do not transfer to other architectures. Before calibration, each replay implementation must reproduce the original forward pass at identical weights and inputs. Stored generation states and execution history remain fixed during replay; neither is regenerated by the merged policy.

Match the experts' linear outputs. For each module, P is its fixed weight in the current pass's prior checkpoint: the expert mean in pass one, then the first-pass checkpoint. Suppressing block and module indices, $W_i, P \in \mathbb{R}^{d_{\mathrm{out}} \times d_{\mathrm{in}}}$ and $X_i \in \mathbb{R}^{n_i \times d_{\mathrm{in}}}$. Let ω_i be the expert loss mass, with $\sum_i \omega_i = 1$; dimensions and masses are module-specific. We fit the correction to preserve expert outputs on X_i, while ridge regularization limits changes from P:

$$C^\star = \arg\min_C \sum_i \frac{\omega_i}{n_i} \left\| X_i(P+C)^\top - X_i W_i^\top \right\|_F^2 + \lambda \|C\|_F^2, \qquad W^{\mathrm{M}} = P + C^\star. \tag{6}$$

Define $\Gamma_i = (\omega_i/n_i) X_i^\top X_i$ and $\Gamma = \sum_i \Gamma_i$. Following the regression-based merging principle of RegMean (Jin et al., 2023) , we use the following weighted prior-centered ridge solution:

$$W^{\mathrm{M}} = \left(\sum_i W_i \Gamma_i + \lambda P\right)(\Gamma + \lambda I)^{-1}. \tag{7}$$

We use linear solves rather than explicitly forming the inverse. Positive ridge makes the system well-defined even with dependent rows. For any direction $\mathbf{v}$ with $X_i\mathbf{v} = 0$ for every expert, $C^\star\mathbf{v} = 0$: the prior is retained. Thus regression retains the prior in directions not covered by calibration features. The equivalent smaller system, bias handling, ridge scaling, and post-solve safeguard are given in Appendix A.

Weighting the experts. In the first pass, TCR weights each expert by how closely the prior already matches its linear outputs:

$$e_i^{\text{prior}} = \frac{\|X_i(P - W_i)^\top\|_F^2}{n_i d_{\text{out}}}, \qquad \omega_i^{\text{rel}} = \frac{1/\max(e_i^{\text{prior}}, \epsilon)}{\sum_{i'=1}^{M} 1/\max(e_{i'}^{\text{prior}}, \epsilon)}. \tag{8}$$

These weights are computed before solving each module and remain fixed during that solve. Without the numerical floor, the data term is proportional to $\sum_i E_i(C)/E_i(0)$, where $E_i(C)$ is the unnormalized squared residual. This measures each expert's error relative to its error under the prior. The second pass instead assigns uniform expert masses, $\omega_i^{\text{uni}} = 1/M$. Uniform masses give equal weight to expert losses, not equal parameter averaging.

3.4 Two Calibration Passes, One Deployed Policy

Let $\mathcal{S}(\Theta, \mathcal{D}; \text{rule})$ denote one blockwise pass with fixed checkpoint prior Θ, expert cache $\mathcal{D}$, and weighting rule. Using separate caches $\mathcal{D}_A^E$ and $\mathcal{D}_B^E$ from the same frozen experts, TCR starts from $\bar{\Theta} = M^{-1}\sum_i \Theta_i$ and computes

$$\Theta^{(1)} = \mathcal{S}(\bar{\Theta}, \mathcal{D}_A^E; \text{rel}), \qquad \Theta^{\text{TCR}} = \mathcal{S}(\Theta^{(1)}, \mathcal{D}_B^E; \text{uni}). \tag{9}$$

Both passes match the frozen experts using forward execution and linear solves; the second fits cache B with $\Theta^{(1)}$ as its prior. Only Θ^{TCR} is deployed, retaining the original graph without expert routing or online recalibration. Per-pass sampling and budgets are given in Appendix A.3.

4 Experiments

We test skill retention, calibration choices, action agreement, real-robot behavior, and cross-benchmark performance. Supplementary controls separate execution coverage, repeated calibration, and data budget.

4.1 Evaluation Setting

LIBERO Spatial, Object, Goal, and Long define four ten-task specialists (Liu et al., 2023). The main LIBERO comparison uses the same four frozen $\pi_{0.5}$ experts, trained from a common initialization for 10,000 steps each. Preprocessing, action representation, chunk execution, and serving mode are fixed across mergers within each evaluation setting. The main LIBERO comparisons report success as the mean $\pm$ sample standard deviation across three evaluations of 400 episodes each (ten per task), with shared task and initial-state assignments across methods. Appendix C details data splits, evaluation repeats, and generation-noise handling.

4.2 Skill Retention after Merging

Can one fixed policy retain several specialists' skills? We compare average and suite-level retention against own-suite experts.

Simulation. Table 1 compares eight baselines on the common expert bank and protocol; Appendix C.2 details their recipes and calibration budgets.

TCR obtains 934/1200 successes (77.83$\pm$2.52%), exceeding FeatCal, the strongest baseline, by 9.50 percentage points (820/1200; 68.33$\pm$0.76%). The gain is concentrated in Goal and Long: TCR exceeds FeatCal by 20.67 and 15.33 percentage points, respectively, versus 5.00 on Object and a 3.00-point deficit on Spatial. Long also exposes a large gain over direct averaging (63.33% versus

Table 1: LIBERO success (%) with four 10k-step $\pi_{0.5}$ experts. Three 400-episode runs report mean $\pm$ sample standard deviation. Blue marks TCR; bold marks the best expert or merger in each column.

Method	Spatial	Object	Goal	Long	Average ↑
Experts^{ref}	$\mathbf{92.00}^{\pm 1.00}$	$\mathbf{99.33}^{\pm 1.15}$	$\mathbf{97.33}^{\pm 2.52}$	$\mathbf{97.00}^{\pm 2.00}$	$\mathbf{96.42}^{\pm 0.76}$
Model Soups	$68.33^{\pm 3.51}$	$54.33^{\pm 2.52}$	$46.33^{\pm 1.15}$	$10.33^{\pm 1.15}$	$44.83^{\pm 1.81}$
Task Arithmetic	$68.67^{\pm 1.15}$	$56.00^{\pm 2.65}$	$42.00^{\pm 1.73}$	$17.33^{\pm 1.53}$	$46.00^{\pm 0.75}$
TIES-Merging	$65.00^{\pm 5.00}$	$54.33^{\pm 0.58}$	$37.33^{\pm 1.53}$	$20.33^{\pm 2.52}$	$44.25^{\pm 1.30}$
WUDI-Merging	$67.00^{\pm 1.00}$	$60.67^{\pm 2.08}$	$38.00^{\pm 2.00}$	$18.33^{\pm 2.52}$	$46.00^{\pm 0.43}$
KnOTS-TIES	$69.33^{\pm 3.21}$	$69.33^{\pm 5.03}$	$44.67^{\pm 2.52}$	$6.67^{\pm 1.15}$	$47.50^{\pm 2.63}$
RegMean	$71.33^{\pm 3.51}$	$81.33^{\pm 0.58}$	$46.67^{\pm 2.52}$	$15.33^{\pm 2.08}$	$53.67^{\pm 0.63}$
RegMean++	$76.00^{\pm 3.00}$	$86.00^{\pm 1.00}$	$50.00^{\pm 1.00}$	$20.67^{\pm 2.08}$	$58.17^{\pm 1.04}$
FeatCal	$\mathbf{83.00}^{\pm 2.65}$	$89.00^{\pm 1.00}$	$53.33^{\pm 1.53}$	$48.00^{\pm 3.61}$	$68.33^{\pm 0.76}$
TCR (ours)	$80.00^{\pm 3.00}$	$\mathbf{94.00}^{\pm 2.65}$	$\mathbf{74.00}^{\pm 6.24}$	$\mathbf{63.33}^{\pm 3.21}$	$\mathbf{77.83}^{\pm 2.52}$

Baseline adaptations and numerical settings are documented in Appendix C.2.

10.33%); the specialist reference is 97.00%. Section 4.3 tests observation source, generation inputs, and network features directly.

Appendix B.8 shows matched execution examples, including a drawer-opening task where TCR succeeds and Soup, TIES, and FeatCal fail.

Real robot. The hardware study uses $\pi_{0.5}$ task specialists on a PiperX robot, trained from a common base, and shared calibration traces from their physical executions. Table 2 compares each merger (Model Soups, TIES, RegMean++, FeatCal, and TCR) against own-task specialists across 20 trials per task (40 per method). Each trial has a 60 s execution limit; timeouts and assisted completion do not count as autonomous success. Appendix C.8 gives collection and safety procedures.

Table 2: Real-robot success (20 trials/task, 60 s/trial) and offline action RMSE. Task 1 is plate placement; Task 2 is stacking; Avg. weights them equally. RMSE pools action elements against the corresponding expert, not task completion (Appendix C.8). Experts use own-task specialists; mergers use one checkpoint.

	Real-robot success (%, ↑)			Offline action RMSE (↓)	
Method	Task 1	Task 2	Avg.	Plate placement	Stacking
Experts^{ref}	**85.0**	**90.0**	**87.5**	0.0000	0.0000
Model Soups	15.0	5.0	10.0	0.0985	0.0648
TIES	20.0	10.0	15.0	0.1063	0.0826
RegMean++	45.0	55.0	50.0	0.0738	0.0563
FeatCal	70.0	45.0	57.5	0.1084	0.0693
TCR	**75.0**	**75.0**	**75.0**	**0.0471**	**0.0486**

Expert RMSE is zero by self-comparison. Bold marks best success per group and best merged-policy RMSE.

The own-task specialists succeed in 35/40 trials overall (87.5%), and TCR succeeds in 30/40 (75.0%). Compared with FeatCal's 23/40 (57.5%), TCR's advantage is larger on stacking (75% versus 45%) than plate placement (75% versus 70%). TCR also has the lowest expert-relative action RMSE among mergers on both tasks, but action agreement does not determine the success ranking: on plate placement, FeatCal succeeds more often than Soup (70% versus 15%) despite higher RMSE (0.1084 versus 0.0985). Errors can affect subsequent interactions differently; RMSE alone does not capture this distinction.

4.3 Which Calibration Choices Matter?

Actions shape observations, generator calls shape later inputs, and merged blocks shape downstream features. Table 3 replaces one calibration choice at a time, in both passes, with three 400-episode evaluations per variant.

Task execution: where do observations come from? We substitute demonstration observations, proprioception, and instructions for rollout inputs, retaining frozen teachers, their generator, and the regression rule. Expert outputs are recomputed on these inputs; demonstration actions are not regression targets. With task allocations, requests, calls, and rows matched, success falls from 77.83% to 73.00%. Every repeat and suite average declines, by 2.33 points on Spatial to 7.33 on Long (Table 11). Rollout-derived observations thus improve retention across all four suites in this comparison; Appendix B.6 reports task-cluster uncertainty, and Appendix B.3 controls for early rollout coverage.

Table 3: Calibration ablations on $\pi_{0.5}$–LIBERO. Success (%) is mean ± standard deviation over three 400-episode evaluations.

Variant	MSE ↓	Cos. ↑	Success ↑
Full	$\mathbf{0.0488}^{\pm 0.0011}$	$\mathbf{0.8620}^{\pm 0.0017}$	$\mathbf{77.83}^{\pm 2.52}$
No rollouts	$0.0513^{\pm 0.0008}$	$0.8545^{\pm 0.0015}$	$73.00^{\pm 1.52}$
Final call	$0.0516^{\pm 0.0014}$	$0.8571^{\pm 0.0022}$	$73.17^{\pm 2.31}$
No replay	$0.0655^{\pm 0.0008}$	$0.8302^{\pm 0.0018}$	$64.50^{\pm 1.52}$

Action generation: which model calls are calibrated? We keep the same expert requests but retain only the final generator call. For $\pi_{0.5}$, calls differ in noisy action state and flow time even at a fixed observation. With per-call sampling unchanged, first-pass rows fall from 4,441,200 to 1,480,400; second-pass rows remain 1,331,600. Success falls to $73.17 \pm 2.31\%$, supporting intermediate-call calibration in this recipe.

Network propagation: whose features are fitted? Replacing merged-prefix features with expert features, while keeping records, calls, regression, and parameter scope fixed, yields $64.50 \pm 1.52\%$, the lowest variant. This supports merged-prefix replay within the tested construction; the strategy follows prior work (Nguyen et al., 2026).

All variants retain first-pass relative-error weighting and second-pass uniform weighting from each arm's first-pass checkpoint. Together, the replacements support these three choices within the evaluated recipe. Appendix C.3 details numerical controls and supplementary studies.

Does action agreement accompany skill retention? We compare each policy's executed continuous actions with its expert on 400 requests from 80 held-out expert episodes. Policies share observations, history, initial generation noise, and action normalization, but run their own complete native generators. Metrics are mean ± sample standard deviation of three checkpoint-level task-macro scores. Full TCR has the lowest MSE (0.0488) and highest cosine similarity (0.8620) in Table 3; expert-prefix replay disagrees most. RegMean, RegMean++, FeatCal, and TCR also combine greater agreement with higher success (Table 10), though plate placement shows local RMSE does not determine closed-loop success. Appendix C.3 defines aggregation.

4.4 Evaluation Across Benchmarks

LIBERO-PRO. We evaluate frozen LIBERO checkpoints under position, object, language, and changed-goal perturbations, without recalibration or selection on PRO (Zhou et al., 2026). TCR obtains the highest average among the evaluated mergers (43.19%); the expert reference is 50.75%. Appendix C.7 reports results by perturbation type.

Table 4: Cross-benchmark success (%).

Method	LIBERO-PRO ↑	RoboTwin 2.0 ↑
Experts^{ref}	50.75	57.78
Model Soups	28.81	12.22
TIES	27.13	6.11
RegMean++	35.38	13.33
FeatCal	41.38	0.00
TCR (ours)	**43.19**	**15.56**

RoboTwin 2.0. Table 4 summarizes $\pi_{0.5}$ on LIBERO-PRO and RoboTwin 2.0. Unlike PRO's frozen-policy evaluation, RoboTwin uses separately trained bimanual experts for the Coordination, Receptacle, and Precision task groups (Chen et al., 2025). It tests consolidation of a new expert bank on its source tasks, not zero-shot transfer of the LIBERO policy.

On the separately trained RoboTwin expert bank, TCR leads the evaluated mergers at 15.56%, versus 13.33% for RegMean++; the specialist reference reaches 57.78%. Task-group results and PRO perturbation breakdowns appear in Appendix C.7. Appendix B.4 varies the expert set: TCR leads

at two and four experts, whereas FeatCal is higher at three; task coverage grows with the set. Appendix B.7 accounts separately for calibration, fusion, and deployment costs.

4.5 Further Training from Merged Initializations

Can a merged checkpoint improve joint training after its immediate zero-update advantage? We train the common base, Model Soups, RegMean++, FeatCal, and TCR on the same source tasks with matched data order, optimizer, schedule, batch size, and update budget, resetting optimizer states. Table 5 reports fixed-update scores and peaks on a shared evaluation grid; Appendix C.4 gives the full curves (Figure 6) and per-suite results.

The benefit persists during training: at 5,000 updates TCR reaches 96.75% versus 94.00% for the common base, and its 0–5,000-update nAUC is 92.83% versus 73.84%. All four merged initializations also surpass the base's best observed success (94.00%), with peaks from 95.50% to 97.75%. TCR has the highest observed peak, 3.75 points above the base and 0.50 points above FeatCal. Thus, in this matched setting, merging is useful not only for immediate deployment but also as an initialization from which joint training attains higher observed success.

Table 5: Further LIBERO training (% success; 400 episodes/checkpoint). Peak and its step use the extended grid; nAUC uses the 0–5,000-update window.

Initialization m	$S_m(0)$	$S_m(5{,}000)$	$S_m(10{,}000)$	$\mathrm{nAUC}_m^{0:5k}$	Peak (step)
Common base	0.25	94.00	92.00	73.84	94.00 (5,000)
Model Soups	47.50	95.25	93.00	88.36	95.50 (6,500)
RegMean++	58.75	94.75	94.50	88.50	95.50 (5,500)
FeatCal	65.50	95.25	**96.25**	92.09	97.25 (8,000)
TCR	**78.75**	**96.75**	**96.25**	**92.83**	**97.75** (6,500)

5 Discussion and Limitations

Execution-aware calibration improves skill retention: rollout observations, intermediate action states, and merged-prefix replay each increase success in the tested recipe. Thus, embodied merging must consider execution inputs and internal features alongside parameter compatibility. Gains vary across task groups, pointing to the role of calibration coverage.

TCR records and replays native inference states, yielding one static policy. Across LIBERO, LIBERO-PRO, RoboTwin 2.0, and a physical robot, it has the highest aggregate success among evaluated mergers, spanning different tasks, expert banks, and robot settings. It preserves the original inference graph without routing or deployment-time gradient training; added computation occurs during one-time offline calibration and merging.

6 Conclusion

Trajectory-Calibrated Regression records inputs across task execution and action generation, then fits merged weights using features recomputed through the partially merged policy. Its output is one policy with fixed weights and no additional expert router. TCR reaches 77.83% average success on LIBERO, exceeding the strongest baseline in the main comparison by 9.50 percentage points. The ablations support collecting expert execution inputs, retaining intermediate generation calls, and replaying merged features within the tested recipe. Continued training from TCR reaches 97.75% peak success, versus 94.00% when training from the common base.

AI use statement

Generative AI tools assisted with manuscript editing, experiment design, and coding support for the experimental harness. The authors are responsible for verifying the manuscript, code, citations, and supporting artifacts.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation, 2025. URL https://arxiv.org/abs/2506.18088.

Runxi Cheng, Feng Xiong, Yongxian Wei, Wanyun Zhu, and Chun Yuan. Whoever started the interference should end it: Guiding data-free model merging via task vectors. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 10121–10143. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/cheng25h.html.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.026.

MohammadReza Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision – ECCV 2024*, pp. 270–287, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-73226-3.

Pala Tej Deep, Rishabh Bhardwaj, and Soujanya Poria. Della-merging: Reducing interference in model merging through magnitude-based sampling, 2024. URL https://arxiv.org/abs/2406.11617.

Sombit Dey, Jan-Nico Zaech, Nikolay Nikolov, Luc Van Gool, and Danda Pani Paudel. Revla: Reverting visual domain limitation of robotic foundation models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8679–8686, 2025. doi: 10.1109/ICRA55743.2025.11128823.

Yuxia Fu, Zhizhen Zhang, Yuqi Zhang, Zijian Wang, Zi Huang, and Yadan Luo. Mergevla: Cross-skill model merging toward a generalist vision-language-action agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22335–22347, June 2026.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18695–18705, 2025. doi: 10.1109/CVPR52734.2025.01742.

Yanggan Gu, Shuo Cai, Zihao Wang, Wenjun Wang, Yuanyi Wang, Pengkai Wang, Sirui Huang, Su Lu, Jianmin Wu, and Hongxia Yang. Featcal: Feature calibration for post-merging models, 2026. URL https://arxiv.org/abs/2605.13030.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=FCnohuR6AnM.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025a. doi: 10.15607/RSS.2025.XXI.017.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

Wonyoung Lee, Wooseong Jeong, and Kuk-Jin Yoon. Label-free cross-task lora merging with null-space compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 847–859, June 2026.

Longhua Li, Lei Qi, Qi Tian, and Xin Geng. Model merging in the essential subspace. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 31145–31154, June 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

Daniel Marczak, Simone Magistri, Sebastian Cygert, Bartłomiej Twardowski, Andrew D. Bagdanov, and Joost van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=RBZpAa27ls.

Michael S Matena and Colin Raffel. Merging models with fisher-weighted averaging. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17703–17716. Curran Associates, Inc., 2022. doi: 10.52202/068431-1287. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/70c26937fbf3d4600b69a129031b66ec-Paper-Conference.pdf.

The-Hai Nguyen, Dang Huu-Tien, Takeshi Suzuki, and Le-Minh Nguyen. Regmean++: Enhancing effectiveness and generalization of regression mean for model merging. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=H5lDsSCS9i.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/ross11a.html.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. In *International Conference on Learning Representations*, 2016. URL https://mlanthology.org/iclr/2016/rusu2016iclr-policy/.

Keumseo Ryum and Joonhyuk Kang. Ct-merging: Consensus directions and task-level scaling for lora adapter merging, 2026. URL https://arxiv.org/abs/2607.20561.

Ningyuan Shi, Zhipeng Zhou, Hao Wang, Chunyan Miao, and Peilin Zhao. Pact: Preserving anchored cores in task-vectors for model merging, 2026. URL https://arxiv.org/abs/2606.18627.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Shengyu Si, Yuanzhuo Lu, Ruimeng Yang, Ziyi Ye, Zuxuan Wu, and Yu-Gang Jiang. Vla-pro: Cross-task procedural memory transfer for vision-language-action models, 2026. URL https://arxiv.org/abs/2605.29562.

Ronald Skorobogat, Karsten Roth, and Mariana-Iuliana Georgescu. Subspace-boosted model merging, 2025. URL https://arxiv.org/abs/2506.16506.

Wenxuan Song, Han Zhao, Fuhao Li, Ziyang Zhou, Xi Wang, Jing Lyu, Pengxiang Ding, Yan Wang, Donglin Wang, and Haoang Li. Capvector: Learning transferable capability vectors in parametric space for vision-language-action models, 2026. URL https://arxiv.org/abs/2605.10903.

George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model merging with svd to tie the knots. In *International Conference on Learning Representations*, volume 2025, pp. 4501–4519, 2025.

Wenju Sun, Qingyong Li, Wen Wang, Yang Liu, Yangliao Geng, and Boyang Li. Towards minimizing feature drift in model merging: Layer-wise task vector fusion for adaptive knowledge integration. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 172635–172663. Curran Associates, Inc., 2025. doi: 10.52202/085713-5745. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/fbb4f2269267de23e0240e1c01242981-Paper-Conference.pdf.

Sofia Maria Lo Cicero Vaina, Artem Chumachenko, and Max Ryabinin. Mashup learning: Faster finetuning by remixing past checkpoints. *arXiv preprint arXiv:2603.10156*, 2026.

Lirui Wang, Kaiqing Zhang, Allan Zhou, Max Simchowitz, and Russ Tedrake. Robot fleet learning via policy merging. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 51737–51771, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e28b3369186459f57c94a9ec9137fac9-Paper-Conference.pdf.

Yongxian Wei, Runxi Cheng, Xingxuan Zhang, Li Shen, Chun Yuan, Peng Cui, and Dacheng Tao. Closed-form spectral regularization for multi-task model merging, 2026. URL https://arxiv.org/abs/2606.07289.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 23965–23998. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in neural information processing systems*, 36:7093–7115, 2023.

Yajat Yadav, Zhiyuan Zhou, Andrew Wagenmaker, Karl Pertsch, and Sergey Levine. Robust fine-tuning of vision-language-action robot policies via parameter merging. In *International Conference on Learning Representations*, volume 2026, pp. 56385–56417, 2026.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations*, volume 2024, pp. 22743–22763, 2024.

Yuxuan Yao, Shuqi LIU, Zehua Liu, Qintong Li, Mingyang LIU, Xiongwei Han, Zhijiang Guo, Han Wu, and Linqi Song. Activation-guided consensus merging for large language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 124041–124069. Curran Associates, Inc., 2025. doi: 10.52202/085713-4133. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/b395e7bd5eaf3e980386debf2e9d7f11-Paper-Conference.pdf.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: absorbing abilities from homologous models as a free lunch. ICML'24. JMLR.org, 2024.

Han-Chen Zhang, Zi-Hao Zhou, Mao-Lin Luo, Shimin Di, Min-Ling Zhang, and Tong Wei. Dcmerge: Improving model merging with directional consistency, 2026. URL https://arxiv.org/abs/2603.06242.

Haobo Zhang and Jiayu Zhou. Unraveling LoRA interference: Orthogonal subspaces for robust model merging. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26459–26472, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1284. URL https://aclanthology.org/2025.acl-long.1284/.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

Shenghe Zheng, Hongzhi Wang, Chenyu Huang, Xiaohui Wang, Tao Chen, Jiayuan Fan, Shuyue Hu, and Peng Ye. Decouple and orthogonalize: A data-free framework for lora merging, 2025. URL https://arxiv.org/abs/2505.15875.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization, 2026. URL https://arxiv.org/abs/2510.03827.

A METHOD AND IMPLEMENTATION DETAILS

Throughout, i, r, f, b, j index experts, policy requests, generation calls, blocks, and linear modules; m, t, e, k, g index methods, tasks, evaluation episodes, repeats, and task groups. A cache request index retains its episode identity. Module indices are suppressed in local regression formulas.

A.1 ASSUMPTIONS AND SCOPE

Experts must have homologous computation graphs and compatible input, action, tokenizer, and normalization conventions. Each calibration state is associated offline with the expert whose task group generated it; this association is not an inference-time task ID and is absent from the deployed checkpoint. Finally, the calibration set must cover the activation directions on which expert behavior differs: Eq. (7) deliberately retains the prior in unsupported directions and therefore cannot infer a missing skill from parameter geometry alone. With $N_{\text{rows}} = \sum_i n_i$, the sample-space solve uses $O(N_{\text{rows}}^2)$ kernel storage and $O(N_{\text{rows}}^3)$ factorization time per module, in addition to replay and weight reconstruction.

Checkpoint materialization. TCR takes effective expert weights as input, independently of their fine-tuning parameterization. For an expert trained with LoRA, we materialize $W_i = W_0 + \gamma_i B_i A_i$ before fusion, where γ_i is its adapter scale (Hu et al., 2022). Directly trained weights and biases enter the same dense checkpoint without this conversion. Materialization preserves the expert's function and is an implementation step, not a LoRA-specific fusion rule or a separate novelty claim. The merged correction is solved in weight space rather than constrained to the original adapter factors. Every adapted tensor is accounted for by either a regression solve or a declared fallback; full adapted-parameter coverage does not mean that every base-model tensor is changed.

A.2 THE RELATIVE-RISK IDENTITY

The inverse-prior-error masses defined in Eq. (8) and used in Eq. (6) have an exact interpretation. Let $E_i(C) = \|X_i C^\top - X_i(W_i - P)^\top\|_F^2$, equivalently the output-matching residual in Eq. (6); then $E_i(0) = n_i d_{\text{out}} e_i^{\text{prior}}$. When the floor is inactive, $\omega_i^{\text{rel}} = (1/e_i^{\text{prior}})/Z_{\text{rel}}$, where $Z_{\text{rel}} = \sum_{i'=1}^{M} 1/e_{i'}^{\text{prior}}$, and

$$\sum_i \frac{\omega_i^{\text{rel}}}{n_i} E_i(C) = \sum_i \frac{E_i(C)}{Z_{\text{rel}}\, n_i e_i^{\text{prior}}} = \frac{d_{\text{out}}}{Z_{\text{rel}}} \sum_i \frac{E_i(C)}{E_i(0)}. \tag{10}$$

The objective thus minimizes a sum of *fractional* errors relative to the common prior, rather than raw residual energy, which one high-energy expert could otherwise dominate. The floor clips this interpretation only for numerically negligible prior errors.

A.3 NUMERICAL AND IMPLEMENTATION CHOICES

When there are fewer input rows than feature dimensions, the regression can be solved using a smaller matrix. Stack $\widetilde{X}_i = \sqrt{\omega_i/n_i} X_i$ and $\widetilde{Y}_i = \sqrt{\omega_i/n_i} X_i (W_i - P)^\top$ by rows; then

$$C^\star = \widetilde{Y}^\top (\widetilde{X}\widetilde{X}^\top + \lambda I)^{-1} \widetilde{X}. \tag{11}$$

This is equivalent to Eq. (7); it changes how the system is solved, not its objective.

The first pass sets $\lambda = 0.05 \max(\|\widetilde{X}\|_F^2/d_{\text{in}}, 10^{-12})$. The second pass reuses the recorded numerical ridge value for each module from its first-pass reference rather than recomputing it from the second cache. For a biased module we augment X with a constant column and append the bias vector $\mathbf{b}_{\text{bias}}$ to W, applying the same ridge penalty to the augmented correction; d_{in} then denotes the augmented width. A radial safeguard limits extrapolation from finite calibration data: with $R_{\text{cap}} = 3 \max(M^{-1} \sum_i \|W_i - P\|_F, 10^{-12})$,

$$\rho = \min\left(1, \frac{R_{\text{cap}}}{\max(\|C^\star\|_F, 10^{-12})}\right), \qquad W^{\text{M}} = P + \rho\, C^\star. \tag{12}$$

This is a numerical safeguard applied after ridge regression, not the exact solution of a trust-constrained problem. Whether it activates is checked per module and recorded in each checkpoint's construction manifest. Equation (7) follows from the normal equation for $Z = C^\top$ together with the push-through identity $(\widetilde{X}^\top \widetilde{X} + \lambda I)^{-1} \widetilde{X}^\top = \widetilde{X}^\top (\widetilde{X} \widetilde{X}^\top + \lambda I)^{-1}$, after transposing; positive ridge makes the system positive definite even when sampled activation rows are linearly dependent. Retaining an immutable prior within a pass prevents the ridge center from depending on incidental module-update order.

Sampling and weighting. The main $\pi_{0.5}$ recipe uses two expert caches with different row-allocation rules. Cache A supplies the first pass with 4,441,200 rows, using a cap of ten rows per stored call and view in its replay path. Cache B uses ten tasks, one episode per task, five requests per episode, and three of ten flow steps, giving 150 cached call states per expert. Its second-pass replay shares a cap of sixteen rows per request and module across selected stages, for 1,331,600 rows in total. The cumulative budget is 5,772,800 rows; a cap value alone is not a comparable budget across these replay paths. All retained flow states belonging to a sampled request are kept together, preserving the association between a robot observation and its denoising path. Because each state contributes sampled token rows, equal state counts need not imply equal regression budgets when sequence lengths or module shapes differ, so we record the realized row count for every expert and module. Within each expert, sample-source weights scale rows before regression. With unit source weights, sampled rows contribute equally, while the separate ω_i/n_i factor gives each expert its prescribed total objective mass. The main comparison holds the expert bank and evaluation protocol fixed while retaining each method's calibration recipe and reporting its budget. Component-removal studies report the rows lost with removed inputs.

A.4 $\pi_{0.5}$ Replay Structure

Table 6 gives the per-stage breakdown of the 422 modified tensors. Language replay refreshes the prefix keys and values consumed by the action expert; the two time projections are solved sequentially through their intervening activation; and the action output projection is solved once the final action hidden state is available. Replaying a block is validated against the native forward pass under matched weights and inputs. Comparing representations generated by *different* policies is a distribution diagnostic rather than an equivalence test, and we do not use it as one. Each artifact records expert and prior identities, replay manifests, per-module row counts, solver configuration, and checkpoint hashes.

Table 6: $\pi_{0.5}$ replay structure and adapted linear modules. The output interface is solved after the action blocks. Identical common-base parameters outside this scope are retained.

Stage	Blocks	Linear	Updated boundary
Vision	27	162	Visual tokens
Language	18	126	Prefix states, K/V
Action/time input	–	3	Action tokens, time
Action expert	18	126	Action hidden states
Action output	–	1	Predicted velocity
Total	63	418	422 tensors with bias

A.5 Fusion Procedure

Preparation materializes every LoRA update into its dense expert weight, loads directly adapted interfaces, and verifies architecture, tokenizer, normalizers, and action conventions. The first pass fixes the dense expert mean as its prior; the second fixes the first-pass checkpoint as its prior. Each pass visits all adapted blocks in computation order using its designated expert cache and weighting rule. For each block, the boundary is generated with the current merged prefix; each candidate block is run on its associated calibration states; linear inputs are captured; the prior-centered regression is solved; all solutions are written; and the completed block is replayed once to produce the next boundary. Deployment checks tensor scope, finite values, configuration, and artifact hashes, then

exports the merged weights with the original policy preprocessing. We preserve normalizers, tokenizer settings, action-chunk length, and task text across every evaluated method. The final static model needs no calibration source or task ID, although the natural-language instruction remains an ordinary policy input.

A.6 Implementation Scope

The evaluated implementation uses $\pi_{0.5}$ observations, proprioception, instructions, noisy action chunks, and flow times. Its replay covers the vision and language prefixes, action conditioning, and action blocks. We verify replay against the native forward pass at matched weights and inputs before merging.

Extending TCR to another architecture would require a separate recording and replay adapter for the inputs and computation used at inference. In particular, a WAM evaluation would have to specify whether video generation is part of the executed policy and validate the corresponding replay. We do not report such a comparison here.

A.7 LIBERO-PRO Aggregation

Our PRO evaluation covers position, object, language, and changed-goal perturbations. For each condition, success is macro-averaged over the 40 evaluated tasks. The PRO Average weights all four conditions equally, including Task and excluding clean LIBERO, using the same task manifest and denominators for all methods. We report the fixed repeat-01 score without a cross-repeat standard deviation, as specified in Appendix C.7. Per-condition results remain visible because changed-goal performance differs from same-task robustness.

B Supplementary Experiments

The supplementary studies address calibration passes and cache reuse, row budgets, and other choices in the final two-pass recipe. Matched-budget component controls, functional measurements, detailed success breakdowns, and cost accounting support these comparisons.

B.1 Calibration Passes and Fresh Expert Data

Does a second pass help because it repeats regression, or because it uses another set of expert execution inputs? We separate these questions from the three component replacements in the main text. Let A, B, and C denote expert execution caches from different episodes, collected with the same task allocation and sampling procedure. An arrow denotes one calibration pass, not an environment rollout. The two-pass reference is A→B, using the main recipe's first-pass budget $R_1 = 4{,}441{,}200$ and second-pass budget $R_2 = 1{,}331{,}600$.

Relation to the main result. The main TCR model also uses two-pass A→B calibration; the letters denote the roles of the caches, not identical data across studies. For these supplementary controls, we collect fresh A, B, and C caches from the same frozen expert bank and rebuild a shared first-pass A checkpoint and its A→B reference. This reference achieves 75.50% (302/400) and is reused in Tables 7–9. The main result, $77.83 \pm 2.52\%$, instead summarizes three main-experiment evaluation runs. It is not substituted for the supplementary reference when computing differences; the matching and repetition protocol is detailed in Appendix C.3.

Table 7 groups the comparisons by question. A, A→A, A→B, and A→C share the same first-pass checkpoint. A→B→B and A→B→C share the same two-pass checkpoint. All second and third passes use uniform expert masses, the same row quotas, and the numerical ridge map fixed by their first-pass reference. They recompute activations through their own merged prefixes; reusing a raw cache does not mean reusing unchanged activations. These comparisons use the shared evaluation selection and repetition protocol in Appendix C.3. All listed comparisons are reported together; a favorable cache is not selected after inspecting success. The comparisons distinguish three effects. First, A→A improves over A from 259/400 (64.75%) to 291/400 (72.75%): fresh episodes are not required for all of the second-pass gain. This refinement also changes the prior, recomputed features,

Table 7: Calibration passes and cache reuse. A, B, and C are distinct expert execution caches; each arrow is one pass. All rows use the same 400 evaluation episodes. A→B is the shared reference for the supplementary recipe and budget studies.

Calibration	Purpose	Success (%) ↑
A	One pass	64.75
A→A	Repeat the first cache	72.75
A→B	Two-pass reference with fresh data	75.50
A→C	Check a second fresh cache	78.50
A→B→B	Third pass, reuse B	72.00
A→B→C	Third pass, fresh C	**79.25**

Table 8: Row-budget comparisons. $R_1 = 4{,}441{,}200$ and $R_2 = 1{,}331{,}600$ are the reference pass budgets. Fractions denote target allocations; actual per-module counts are recorded. A→B reuses the reference from Table 7, not a separate evaluation. All rows use the same 400 evaluation episodes. A dash indicates that the second pass does not apply.

Configuration	First-pass rows	Second-pass rows	Success (%) ↑
A→B, fewer rows	R_1	$0.5R_2$	69.00
A→B, reference	R_1	R_2	75.50
A→B, more rows	R_1	$1.5R_2$	**77.75**
One pass on A∪B	$R_1 + R_2$	—	74.25
One pass on A∪B∪C	$R_1 + 2R_2$	—	75.00

and weighting relative to the first pass, so it is not simply solving an unchanged regression twice. Second, replacing the second-pass cache with B or C reaches 302/400 (75.50%) and 314/400 (78.50%), 2.75 and 5.75 points above A→A from the same parent checkpoint. Fresh execution inputs thus add a benefit beyond refinement on reused observations in these comparisons. Third, more passes are not monotonically better: A→B→B drops to 288/400 (72.00%), whereas A→B→C reaches 317/400 (79.25%). This pattern is consistent with repeated fitting becoming too specific to a finite cache, but does not establish overfitting as the cause; cache difficulty and representation changes remain possible explanations. These fixed single-seed comparisons support separating cache coverage from pass count rather than treating an extra pass as an automatic improvement.

B.2 Calibration Budget

How many regression rows are needed, and is the two-pass recipe preferable to using its row budget in one pass? We use the same A→B reference as the cache study, rather than constructing another reference for each question. Table 8 varies the second-pass row budget while holding its parent checkpoint, cache B, request coverage, expert masses, and numerical ridge fixed. The smaller and larger settings use nested row subsets with target budgets of $0.5R_2$ and $1.5R_2$; integer rounding and small-module capacity are recorded per module. Additional requests and duplicated rows are not used to meet a target.

The completed union control uses A and B in a single pass from Soup, with total row budget R_1+R_2. It retains the single-pass relative-error weighting rule. This compares the full methods, not just the number of passes: the two-pass version also changes its prior, the features used by later blocks, and the weighting rule. Equal row totals do not imply equal computation; we have not measured the time needed to build these controls. The smaller and larger second-pass budgets achieve 276/400 (69.00%) and 311/400 (77.75%), respectively, compared with 302/400 (75.50%) for the reference budget. Increasing the budget by $0.5R_2$ gains 6.50 points below the reference budget and 2.25 above it, suggesting diminishing returns over the tested range rather than a linear benefit from more rows. The one-pass A∪B recipe reaches 297/400 (74.25%), 1.25 percentage points below the two-pass reference with the same nominal row budget in this fixed single-seed comparison. A further control pools A, B, and C for a single pass from Soup with $R_1 + 2R_2 = 7{,}104{,}400$ rows, matching the

Table 9: Supplementary choices within the two-pass TCR recipe. All rows use the matching A→B reference from Table 7 and the same 400 episodes. Each variant changes its named rule; scope restrictions reduce the total regression budget.

Configuration	Change from reference	Success (%) ↑
Full	None; shared A→B reference	**75.50**
Early requests only	Earliest requests, unchanged request count	68.00
Uniform first-pass weights	Equal expert masses also in pass one	72.25
Action modules only	Other adapted parameters remain at Soup	52.25
Conditioning + action	Other adapted parameters remain at Soup	69.25

cumulative budget of A→B→C. It retains the union control's relative-error weighting and fixed numerical ridge and achieves 300/400 (75.00%) on the same evaluation episodes, compared with 317/400 (79.25%) for the three-pass recipe. This 4.25-point difference favors sequential calibration over pooling at matched row budgets; it does not isolate pass count from changes in the prior, prefix features, or weighting rule.

B.3 Other Choices in the Final Recipe

Table 9 tests four choices around the same two-pass reference: coverage of execution time, first-pass expert weighting, and two restrictions of the calibrated parameter scope. Each row changes only its named rule in the construction procedure. The early-request variant keeps the request count but uses the earliest eligible requests in each episode. The uniform variant changes only the first pass, since the second pass is already uniform. Both scope variants leave uncalibrated adapted parameters at the expert mean and replay through their own partially calibrated policy. They retain Full's row quotas and numerical ridge for the modules that remain in scope; total rows and cost therefore decrease with scope. All comparisons use the same evaluation episodes as their matching reference, following Appendix C.3. The one-pass control is A in Table 7, not a duplicate experiment here. Restricting observations to early requests reduces success from the matching Full's 75.50% to 272/400 (68.00%), despite preserving the request count. This complements the main observation-source comparison: using rollout data alone is not sufficient if sampling neglects later execution states. Uniform first-pass weights yield 289/400 (72.25%), a 3.25-point decrease, supporting the weighting choice within this supplementary reference rather than identifying it as the source of the main method's gain. Restricting calibration to action modules yields 209/400 (52.25%); adding conditioning recovers 17.00 points to 277/400 (69.25%), with Full a further 6.25 points higher. The tested action-only correction therefore leaves substantial skill loss that broader calibration can recover. Because broader scope also increases total regression rows and computation, these are recipe comparisons, not equal-budget causal attributions to individual parameter groups.

B.4 Number of Merged Experts

We use three fixed nested expert sets: Spatial+Goal, Spatial+Object+Goal, and Spatial+Object+Goal+Long. The two- and three-expert subset policies are each evaluated once on their included suites, using 100 episodes per suite from the fixed repeat-01 selection. Model Soups achieves 130/200 (65.00%) and 162/300 (54.00%); TCR achieves 173/200 (86.50%) and 259/300 (86.33%). TIES achieves 116/200 (58.00%) and 174/300 (58.00%). RegMean++ achieves 158/200 (79.00%) and 213/300 (71.00%). FeatCal achieves 149/200 (74.50%) and 263/300 (87.67%). The four-expert endpoint reuses the main-table comparison over three 400-episode evaluations: Model Soups achieves 538/1,200 (44.83±1.81%), TCR achieves 934/1,200 (77.83±2.52%), TIES achieves 531/1,200 (44.25±1.30%), RegMean++ achieves 698/1,200 (58.17±1.04%), and FeatCal achieves 820/1,200 (68.33±0.76%) (Figure 2). TCR leads the compared methods at two and four experts, whereas FeatCal is 1.33 percentage points higher at three experts (four additional successful episodes). Thus, the margin over direct averaging does not imply a consistent advantage over every functional merger as the expert set changes. The four-expert TCR checkpoints and number of evaluations therefore differ from the smaller fixed-subset points; the task set also expands with expert count. The connected points are descriptive and do not isolate an expert-count effect.

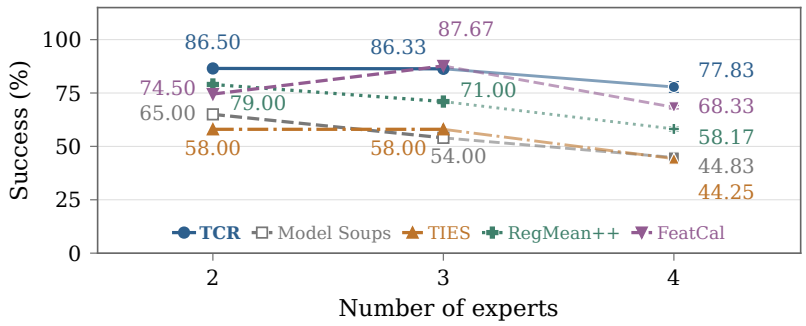

Figure 2: Success of TCR, Model Soups, TIES, RegMean++ (with spectral smoothing), and FeatCal for Spatial+Goal and Spatial+Object+Goal subsets (one evaluation each; no error bars), followed by the four-expert main-table endpoint (mean ± sample standard deviation over three evaluations). The four-expert TCR checkpoints differ from those of the smaller subsets. Each evaluation contributes 100 episodes per included suite. Task coverage grows with the expert set; this is not a fixed-task scaling comparison.

Table 10: Action agreement for the main LIBERO methods on 400 shared held-out expert requests. Success is copied from Table 1; no additional success evaluation is used here. Action metrics use the same request bank.

Method	Action MSE ↓	Action cos ↑	Success (%) ↑
Model Soups	0.1094	0.7547	44.83
Task Arithmetic	0.1196	0.7516	46.00
TIES-Merging	0.1169	0.7452	44.25
WUDI-Merging	0.1218	0.7427	46.00
KnOTS-TIES	0.0980	0.7661	47.50
RegMean	0.0881	0.7899	53.67
RegMean++	0.0773	0.8080	58.17
FeatCal	0.0678	0.8285	68.33
TCR	$\mathbf{0.0488}^{\pm 0.0011}$	$\mathbf{0.8620}^{\pm 0.0017}$	**77.83**

The RegMean row uses the same frozen $\alpha = 0.3$ checkpoint for action agreement and the success rate copied from Table 1.

B.5 Action Agreement

Does better task retention accompany closer agreement with the source experts? We measure normalized action MSE and cosine on 400 shared held-out expert requests, without rerunning the closed-loop success evaluations. Each policy generates its own complete action sequence from the same observation and initial noise. Table 10 compares the main methods; Table 3 reports the corresponding action metrics for the component study. The measurement definitions and pairing rules are in Appendix C.3. Lower action error is a diagnostic, not a guarantee of higher task success. Across RegMean, RegMean++, FeatCal, and TCR, MSE decreases from 0.0881 to 0.0488 while success increases from 53.67% to 77.83%; cosine follows the same ordering. This association is descriptive across the evaluated constructions, not an independently controlled effect of action error. It is not a strict ranking rule even within simulation: Soup has lower MSE than Task Arithmetic (0.1094 versus 0.1196), but lower success (44.83% versus 46.00%). The physical plate-placement reversal in Section 4.2 provides another counterexample; hardware RMSE and simulation MSE use different action representations and should not be compared numerically across domains. These observations motivate reporting functional fidelity alongside, rather than instead of, closed-loop success.

Table 11: Per-suite success (%, mean ± sample standard deviation) for the three component ablations in Table 3, over three evaluations of 100 episodes per suite. Bold marks the largest displayed mean in each column.

Configuration	Spatial	Object	Goal	Long	Average
Full	$80.00^{\pm 3.00}$	$\mathbf{94.00}^{\pm 2.65}$	$\mathbf{74.00}^{\pm 6.24}$	$\mathbf{63.33}^{\pm 3.21}$	$\mathbf{77.83}^{\pm 2.52}$
w/o rollout obs.	$77.67^{\pm 0.58}$	$90.00^{\pm 0.00}$	$68.33^{\pm 3.21}$	$56.00^{\pm 4.58}$	$73.00^{\pm 1.52}$
w/o intermediate calls	$79.33^{\pm 2.08}$	$91.67^{\pm 2.52}$	$70.67^{\pm 8.02}$	$51.00^{\pm 5.29}$	$73.17^{\pm 2.31}$
w/o merged replay	$\mathbf{81.33}^{\pm 1.53}$	$85.00^{\pm 4.36}$	$58.00^{\pm 3.61}$	$33.67^{\pm 4.73}$	$64.50^{\pm 1.52}$

Table 12: LIBERO-PRO evaluation of frozen $\pi_{0.5}$ LIBERO checkpoints under four perturbation types: position (Pos.), object (Obj.), language (Lang.), and changed task goals (Task). Success is reported in %. Average weights the four perturbation types equally and excludes clean LIBERO results. No PRO-driven recalibration or selection is permitted.

Method	Pos.	Obj.	Lang.	Task	Average
Experts^{ref}	18.00	75.75	91.25	18.00	50.75
Model Soups	17.00	37.50	43.25	**17.50**	28.81
TIES	20.75	30.50	41.25	16.00	27.13
RegMean++	23.00	49.00	58.00	11.50	35.38
FeatCal	**30.50**	**56.75**	63.50	14.75	41.38
TCR (ours)	**30.50**	56.50	**75.50**	10.25	**43.19**

B.6 Detailed Success Results

Table 11 expands all three component comparisons by suite; the matching and construction protocols are specified in Appendix C.3. For the observation-source comparison, the main-table Full and the demonstration replacement succeed in 934 and 876 of 1,200 episodes, respectively. Full wins 124 paired episodes and loses 66; a descriptive task bootstrap (20,000 resamples, seed 274921) gives a 95% interval of $[-0.75, 11.00]$ percentage points. The aggregate advantage is therefore not a statistically resolved benefit across the sampled tasks. This interval conditions on the constructed checkpoints, rather than estimating variation across independently trained expert banks. The generation-input and prefix rows reuse their completed matched evaluations.

Table 12 separates PRO perturbations; benchmark task groups are reported in Appendix C.7. These are breakdowns of the main experiments, not additional method-selection sets.

B.7 Construction and Deployment Cost

Table 13 reports construction time and sampled peak GPU memory for the reported checkpoints. Expert training and validation screening are excluded from merge times; data collection is included where applicable.

TCR uses 4,441,200 first-pass and 1,331,600 second-pass regression rows, totaling 5,772,800.

Task Arithmetic and TIES construction times profile only their selected checkpoints after screening. Both materialize on CPU with CUDA hidden, so peak GPU usage is zero. Model Soups was rematerialized to the reported checkpoint in 155.74 seconds, with 5 MiB sampled whole-device usage on an otherwise empty A100.

RegMean and RegMean++ each select 50 single-flow inputs per expert; FeatCal uses 50 observations with three noise replicas per expert, all at flow time $t = 1$. The regression times cover four source captures and solve/export; FeatCal additionally includes input preparation and parallel source capture. Row totals are rounded from the construction records.

TCR's merge time sums compatible historical stages rather than timing a full build. Its 31.75 GiB peak is measured from an isolated second-pass proxy using the final reported checkpoint as prior,

Table 13: $\pi_{0.5}$ construction accounting for the main comparison. States counts request–flow inputs; Demo. counts supervised states.

Method	States	Rows	Demo.	Grad. steps	Merge (h)	Peak GiB
Model Soups	0	0	0	0	0.04	< 0.1
Task Arithmetic (val.)	0	0	0	0	0.09	0
TIES-Merging (val.)	0	0	0	0	0.25	0
WUDI-Merging	0	0	0	300[†]	0.30	2.4
KnOTS-TIES	0	0	0	0	0.03	0.9
RegMean	200	2.37M	0	0	0.36	21.3
RegMean++	200	2.37M	0	0	0.35	21.3
FeatCal	600	4.44M	0	0	1.55	42.9
TCR (ours)	1,200	5.77M	0	0	1.03	31.75

[†]WUDI steps are per optimized matrix. TCR timing and peak-memory scope are detailed below.

not from a complete construction run. GPU peaks use one-second whole-device sampling, taking the largest device peak for FeatCal.

Deployment. TCR deploys one checkpoint covering all four skill families, whereas the expert bank stores four checkpoints of the same architecture. The fused policy retains the expert architecture and native inference procedure, without an additional router or ensemble execution. Thus, checkpoint storage is reduced to approximately one quarter of the full expert bank; per-request inference cost remains comparable to running one expert, with no claimed structural speedup.

B.8 Qualitative Execution Sequences

Figures 3 and 4 show selected drawer-opening and basket-placement episodes, comparing TCR, Model Soups, TIES, and FeatCal. Within each example, task identity and the initial simulator state are matched across methods. Frames are sampled at common recorded-frame indices without aligning task milestones or rescaling each policy's trajectory duration. The examples illustrate differences in execution; the aggregate comparison remains Table 1.

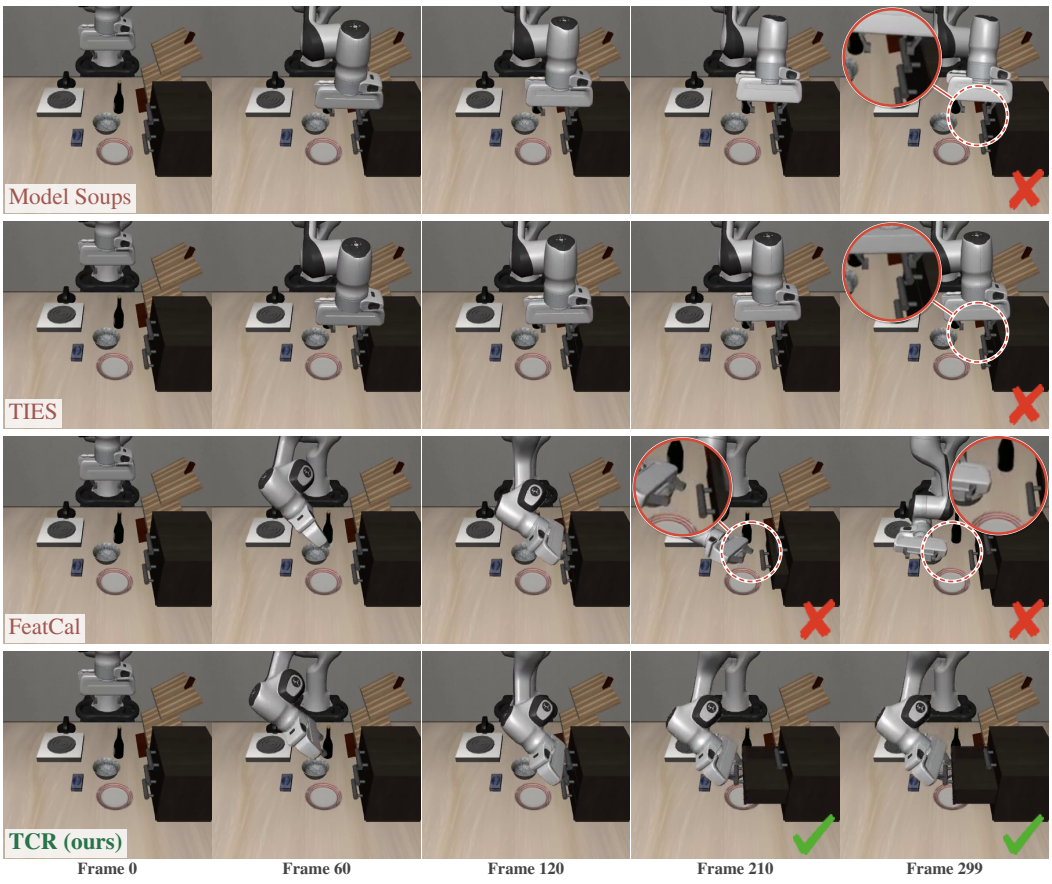

Figure 3: **Skill retention during execution.** Four merged policies attempt to open the middle drawer from the same initial state. Columns show shared recorded-frame indices 0, 60, 120, 210, and 299. TCR succeeds at its final recorded frame 140; that terminal image is held in the 210 and 299 columns, both marked with green checks. Red crosses mark FeatCal's visible wrong-drawer state at 210 and 299 and the terminal failures of Model Soups and TIES at 299. Circular insets show both the gripper and drawer handles, preserving their spatial relationship in the same frames. These marks identify sampled outcomes, not exact failure-onset times. This selected example illustrates behavior, not aggregate success.



Figure 4: Place soup and tomato sauce in the basket from a matched initial simulator state. Columns show recorded frame indices, not seconds. TCR and FeatCal succeed; Model Soups and TIES fail. Checks mark held terminal frames of successful rollouts; crosses mark visible errors at sampled frames, enlarged in circular insets. These selected examples illustrate behavior rather than estimate success rates.

C Experimental Details and Protocols

This appendix details the comparison protocols and task-level results.

C.1 Main LIBERO Evaluation Protocol

The unified comparison uses four experts adapted for 10,000 steps from one local $\pi_{0.5}$ base. Every evaluated method shares the normalizers, tokenizer, ten-step flow sampler, ten executed actions per policy request, action-chunk length, and task text of the source experts, and is scored by the same evaluator on ten episodes for each of ten tasks per suite, i.e. 400 episodes per checkpoint. Calibration inputs and development states are distinct from the formal procedural reset selections. The procedural reset bank was fixed for the reported comparisons but had been inspected during development; reset identities and random-number settings are recorded in the artifacts. Each method is evaluated three times, with 400 episodes per run; we report mean and sample standard deviation, using paired tasks and initial conditions across methods. Registered random seeds do not guarantee identical generation-noise streams throughout diverging rollouts. The primary main-table contrasts compare TCR with Model Soups and the strongest external baseline selected on development data. The uniform first-pass weighting control is a separate mechanism contrast for error weighting. The comparisons are conditional on the evaluated expert bank; they do not estimate variation across independently trained expert sets. Calibration resampling, repeated policy evaluation, and independent expert training are recorded as separate experimental factors. Let $\mathcal{T}$ be the task set, E_t the episode count for task t, and $y_{t,e}(\pi) \in \{0,1\}$ the benchmark-defined success of policy π in episode e. We report task-macro success $\mathrm{Succ}(\pi) = |\mathcal{T}|^{-1} \sum_{t\in\mathcal{T}} E_t^{-1} \sum_{e=1}^{E_t} y_{t,e}(\pi)$, expressed as a percentage, together with per-suite success. The method configuration is fixed before evaluation, and no repeat is selected by its success rate.

C.2 Baseline Configurations

This section documents the baseline configurations; method citations appear in the main text. The main table's RegMean++ uses sequential merged-block replay and the original off-diagonal Gram scaling with fixed $\alpha = 0.3$, plus explicitly added spectral smoothing. Calibration uses 200 static demonstration observations with initial Gaussian action noise at flow time $\tau_{\mathrm{flow}} = 1$, without demonstration action labels or later generation states. For a Gram eigenvalue σ, the inverse filter is $\sigma/(\sigma^2 + \tau_{\mathrm{spec}}^2)$ on the correction around the arithmetic expert mean, with $\tau_{\mathrm{spec}} = d_{\mathrm{in}}\, \epsilon_{\mathrm{FP32}} \widehat{\sigma}_{\max}$ fixed by input width and estimated spectral scale before evaluation. This is a regularized extension, not an algebraically equivalent implementation of the unmodified solver; it uses neither an additive ridge penalty nor a correction-norm cap, and biases use the expert mean. RegMean++ yields 698 successes in 1,200 episodes ($58.17 \pm 1.04\%$). The ordinary RegMean row uses each expert's own prefix and the same static observation pool, scope, and disclosed spectral smoothing rule as RegMean++. It fixes $\alpha = 0.3$ to align their Gram scaling; the remaining prefix propagation distinguishes the two methods. RegMean yields 644 successes in 1,200 episodes ($53.67 \pm 0.63\%$). The WUDI entry retains the original WUDI objective on 414 materialized task-vector matrices, followed by per-tensor RMS rescaling; the eight directly adapted action/time heads use dense averaging. It applies 300 Adam updates per optimized matrix: 409 nonzero matrices are optimized, while five zero-delta matrices and the eight heads receive no gradient updates. These are construction updates, not end-to-end policy-training steps. Model Soups uses uniform averaging (Wortsman et al., 2022): $\bar{\Theta} = M^{-1} \sum_{i=1}^{M} \Theta_i$, where each Θ_i is a materialized expert checkpoint. It uses neither greedy soup selection nor averaging of separate LoRA factors.

Tables use the standard method names; the VLA adaptations and numerical settings are specified in this section. Demonstration observations can supply native generator inputs without using demonstration actions as regression labels.

Method-specific construction details for real-robot evaluation, cross-benchmark comparisons, scaling, and further-training initializations are given in their respective protocols. The main method comparison fixes the expert bank and evaluation settings, retaining and documenting each method's construction recipe. Matched-row controls are reported separately. The hardware comparison shares physical expert traces, as specified in Appendix C.8.

KnOTS-TIES applies KnOTS alignment before TIES (Stoica et al., 2025; Yadav et al., 2023). It operates on the materialized LoRA updates; directly adapted interface tensors use a predeclared fallback whose parameter coverage is reported.

FeatCal. The FeatCal row uses a VLA adaptation of the published feature-calibration objective and forward-order procedure, initialized from Model Soups (Gu et al., 2026). For the displayed result, FeatCal starts from the average of the 10,000-step experts; the base model remains available for its anchor construction. It uses 200 demonstration observations with three independent Gaussian action latents per observation, all at flow time $\tau_{\text{flow}} = 1$, giving 4,441,200 actual regression rows. Demonstration action values are not used; the three noise replicas are not three distinct generation stages. TCR uses the same expert bank but two expert execution caches and 5,772,800 cumulative rows; its second-pass prior is its own first-pass checkpoint. FeatCal pairs features from experts and the model being calibrated using the same records and row indices within its own solve. It retains target interpolation, the merged/base anchor, inverse feature-statistic scale weights, and its own regularization rule. These are not replaced by TCR's expert targets or relative-error weights. The reported configuration fixes interpolation $\alpha_{\text{FC}} = 0.3$, anchor coefficient $\rho_{\text{FC}} = 2$, and regularization $\lambda_{\text{FC}} = 0.05$ before formal evaluation, without retuning on these outcomes. The port calibrates 418 linear weights across 47 forward stages using FP64 solves; biases, LayerNorm parameters, and other uncalibrated tensors retain their Model Soups values. This adapted parameter scope and initial-noise input convention are not an official released $\pi_{0.5}$ configuration. Construction accounting includes the initial average, feature collection, calibration, and validation search; sharing cached inputs does not erase their collection cost. This comparison evaluates complete recipes with a shared expert bank and different calibration budgets. Ablations with a fixed solver are still required to isolate execution coverage.

Ablation implementation and attribution. TCR ablations share a fixed expert bank and prior-centered solver; the intervention determines which inputs or calibrated modules may change. We solve ridge regression on sampled rows with bias augmentation and a declared regularization schedule. A post-solve safeguard limits the weight correction. The replay adapter follows multimodal dependencies, including conditioning keys and values and time projections (Appendix A). These choices define our implementation. TCR builds on regression and depth replay by sampling across interaction requests and native generation calls, using relative-error masses in the first pass and uniform masses in the second.

Configurations and grids. We select one global configuration per tunable method on the fixed development bank. Task Arithmetic and TIES screen 21 and 69 candidates, respectively, on 40 development episodes per candidate. WUDI screens two candidates on 40 episodes each; KnOTS-TIES validates one recipe on 400, RegMean and RegMean++ use 40 each, and FeatCal runs one fixed 40-episode development gate. For KnOTS-TIES, RegMean, and RegMean++, the evaluated scale, rank, pruning, and regularization settings are recorded where applicable. Each search grid and interface fallback is fixed before final evaluation. The methods have different hyperparameters and search sizes. We therefore disclose available validation budgets and fixed-default results; selection budgets are not equal. We use descriptive names for local finite-row, prior-centered ports and do not label a control as a distinct RegMean++ baseline when its calibration set, replay, weighting, and solver are identical to TCR.

Relation to MergeVLA. The released MergeVLA system trains Qwen2.5-0.5B experts with sparse task masks and cross-attention-only action blocks, then selects a stored task mask and action head with a test-time router (Fu et al., 2026). Its published Task Arithmetic, TIES, TSV, KnOTS, WUDI, and EMR numbers therefore measure merging experts constructed for that system, not post-hoc fusion of our fixed $\pi_{0.5}$ experts. Its expert construction could complement TCR, but empirical compatibility requires a separate audit of masks, heads, and native inference. We do not treat its published system scores as algorithm comparisons on our expert bank.

C.3 Shared Protocol for Calibration Controls

The component study in Table 3 and the supplementary studies ask different questions. The former replaces observation source, generation inputs, or the prefix used to compute features. The cache,

budget, and recipe studies in Tables 7–9 explain choices within the final two-pass construction. All use the same four 10k-step experts, Soup initialization, native inference settings, and full adapted scope unless scope is the named intervention.

References and repetition units. The three component contrasts use the same expert bank, calibration pools, and 400-episode repeat selections as the displayed Full recipe. Each configuration has three independently constructed models. The full recipe uses 4,441,200 first-pass and 1,331,600 second-pass rows. Removing intermediate calls reduces the variant's first-pass rows to 1,480,400; the remaining per-call sampling rule is unchanged.

For the supplementary cache, budget, and recipe matrices, A→B is one shared reference built at the main method's budgets. The supplementary A/B/C caches are newly collected execution episodes, not the original caches used to construct the three main-table TCR models; the expert weights and collection protocol are unchanged. The A parent and A→B child are reused wherever specified, instead of fitting or selecting a new reference for each row. In particular, A→A, A→B, and A→C branch from the identical A checkpoint, so their second-pass cache comparisons do not change the first-pass model. These supplementary comparisons use one construction per configuration and a fixed evaluation seed on the same 400 episodes, reporting success without a cross-seed standard deviation; their references are not the main-table three-repeat mean. Calibration episodes, held-out requests, and evaluation resets are separated by their recorded state identities. All prespecified arms and per-suite outcomes are reported, including regressions.

Inputs and numerical controls. Caches have the same task allocation and request-sampling rule; fresh caches use different episodes, not only different generation noise. Requests are selected without conditioning on success. The demonstration replacement changes observations, proprioception, and instructions, but still uses the frozen experts' native generation. It does not substitute demonstration actions for intermediate states or regression targets. Generation and prefix controls share the raw requests with Full. An early-request control retains the request count; episodes with insufficient distinct requests are not padded with duplicates.

Every variant records per-module and per-expert row counts. The named intervention determines whether rows, requests, or scope can change; other quotas remain fixed. Matched-budget studies do not duplicate rows to meet a quota. The reference's numerical ridge map is reused for each paired intervention rather than recomputed from a changed Gram scale. The first-pass relative-error rule is unchanged except in the uniform-weight control; its realized masses can nevertheless change with inputs. Second and later passes use uniform masses and the designated parent's weights as their fixed prior. Precision, module order, bias treatment, and the correction safeguard are held fixed. Changes to scope reduce total work and are not interpreted as equal-compute comparisons.

Measurements. Closed-loop evaluation uses the same tasks and reset selection within each contrast. Matching seeds does not imply identical generation-noise streams after policies visit different states. We report per-suite and aggregate success, paired episode wins and losses, and task-cluster uncertainty where available. Action metrics use the common held-out bank defined below, not training residuals. Construction records include actual rows and, where available, collection and solving time and peak memory. No cache or configuration is selected by its final evaluation score.

Held-out action agreement. These measurements reuse the checkpoints from each comparison; they are not an additional model-selection experiment. Every checkpoint is measured on the same held-out request bank, and construction-level scores are aggregated only within their declared comparison. These request-level measurements are separate from the 1,200 closed-loop episodes and do not require rerunning Full's success evaluations. The common bank contains two expert episodes per task and five requests at fixed trajectory-time quantiles per episode, giving 400 requests over 40 tasks. Initial-state content hashes are checked against the calibration and success-evaluation banks; selection does not condition on success or model disagreement. Episode, request, input, and actual initial-noise identities are retained for each measurement. Each merged policy and the corresponding task expert receives the same request and exact initial generation noise, but generates its own intermediate states using the native sampler and execution horizon. We compare only the action slots that would actually be executed, not unused future predictions, padded coordinates, or actions at the same time index of independently diverging rollouts.

Let $\mathbf{a}_{t,r}^{(m)}$ and $\mathbf{a}_{t,r}^{E}$ be the continuous action vectors for method m and task t's assigned expert $\pi_{i(t)}$ on request r. Use the same fixed coordinate scaling S_a for all arms, derived from the expert bank's training-data action statistics, never estimated separately from each evaluated model. The physical-to-normalized coordinate mapping, scale floor, action mask, and gripper indices are frozen in the measurement manifest. With $D_{t,r}$ valid continuous coordinates and N_t held-out requests for task $t \in \mathcal{T}$, report the task-macro average

$$\mathrm{MSE}(m) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{1}{N_t} \sum_{r=1}^{N_t} \frac{\|S_a(\mathbf{a}_{t,r}^{(m)} - \mathbf{a}_{t,r}^{E})\|_2^2}{D_{t,r}}. \tag{13}$$

Discrete gripper outputs are excluded from this continuous-action metric. Cosine similarity uses the same scaled continuous action vectors, with no model-specific recentering. Pairs with either norm below a fixed threshold are excluded from cosine only; report their count and coverage for every arm rather than assigning artificial agreement. All requests remain in MSE, including near-zero actions. For the completed $\pi_{0.5}$ measurements, normalization files are identical across teachers and candidates. We use normalized outputs with $S_a = I$, the first ten actions, six continuous coordinates, and a cosine norm threshold of 10^{-12}. The three final-call checkpoints yield action MSE 0.0516 ± 0.0014 and cosine similarity 0.8571 ± 0.0022; the three expert-prefix checkpoints yield 0.0655 ± 0.0008 and 0.8302 ± 0.0018, respectively. All 400 requests have valid cosine norms for each of these six checkpoints, so no requests are excluded. The reported standard deviations are across the three checkpoint-level task-macro scores, not across individual requests. Each future checkpoint must pass the same normalization and native-generation checks before joining that comparison.

Reduced action MSE does not guarantee improved closed-loop success.

Task-resolved action fidelity. Figure 5 expands the held-out comparison across all nine merging methods and all 40 tasks. Each cell or point averages the same ten requests for one task; the visualization uses the first evaluated policy for each method. The shared color scale exposes task-specific differences that an overall average can hide. For example, the Goal-suite MSE is 0.1969 for Model Soups, 0.1031 for FeatCal, 0.1123 for RegMean++, and 0.0618 for TCR. The ordering is not uniform: FeatCal has slightly lower Spatial-suite MSE than TCR (0.0563 versus 0.0569). These observations describe output agreement on expert observations, not neuron activation or causal explanations of closed-loop success.

C.4 Continued-Training Protocol

This is the detailed protocol for Table 5, not another supplementary method search. The common base, Model Soups, RegMean++, FeatCal, and TCR receive the same joint training data, task-sampling ratios, data order, optimizer, learning rate, schedule, and batch size. Optimizer states are reset and checkpoints are evaluated on a shared grid $\mathcal{B}$ that includes zero and B. We rerun all initializations in a common training environment and evaluate them on a shared fixed checkpoint grid. We report fixed-update scores and, separately, the best observed checkpoint on this prespecified grid as a convergence diagnostic. The common base is the shared initialization after 200 LIBERO adaptation steps, not an untouched official pretrained checkpoint. All initializations use one checkpoint fixed before continued training. Each initialization is reevaluated at step zero under the continued-training protocol rather than copying its historical main-table score. The original comparison uses evaluation seed 274001, $B = 5{,}000$ optimizer updates, and the grid of zero through 5,000 in 500-update increments, with 400 episodes per checkpoint. Continued training was extended to 10,000 updates, with the same 400-episode evaluation size at each additional checkpoint; Table 5 includes the 10,000-update score and best observed intermediate checkpoint as convergence diagnostics. It uses rank-64 LoRA with scale parameter 64 and directly trains the action/time interfaces, with an initial learning rate of 3×10^{-4}, 500 warmup updates, and final learning rate 2.5×10^{-6}. The training configuration specifies a global batch size of 128.

For initialization m, let $S_m(u)$ be success in percent after u training updates. The learning-curve summary is

$$\mathrm{nAUC}_m = \frac{1}{B} \int_0^B S_m(u)\, du. \tag{14}$$

The integral uses the trapezoidal rule on the common evaluation grid, with success expressed in percent; nAUC is therefore also in percent. We compute the reported nAUC over the original 0–5,000-

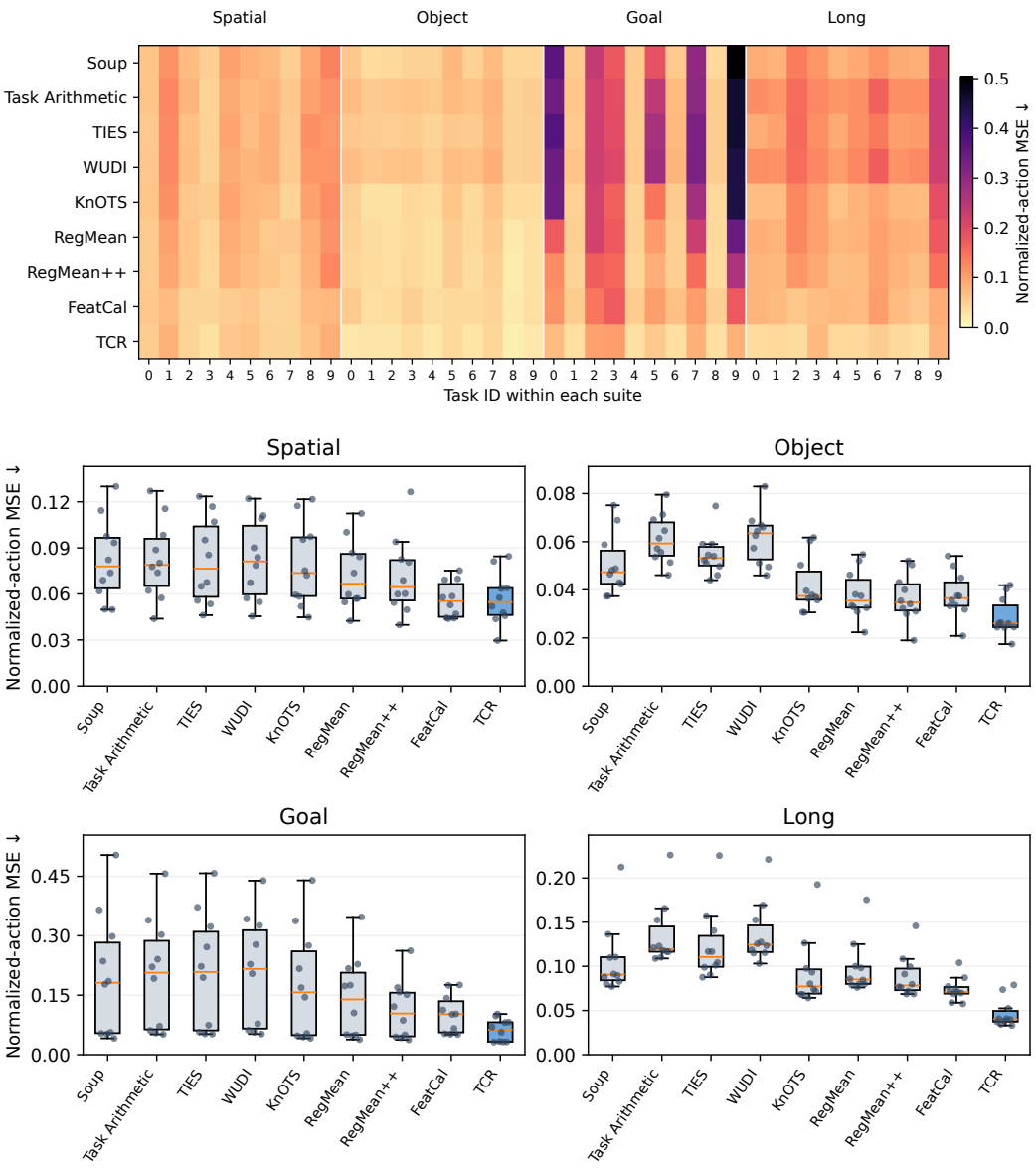

Figure 5: Task-resolved normalized-action error for nine $\pi_{0.5}$ merging methods on the common held-out request bank. Top: all 40 task means with one shared color scale (lower is better). Bottom: the ten task means within each suite, with suite-specific vertical-axis ranges starting at zero; points retain every task, while boxes summarize their distribution, not uncertainty across seeds. Predictions are compared with the assigned expert on identical observations and initial noise. Expert self-error is identically zero and is omitted.

update window directly from the recorded checkpoint scores, without selecting the best intermediate checkpoint. Table 14 reports initial and 5,000-update success per suite to expose recovery and forgetting over the original window. This measures practical utility as an initialization, not optimization difficulty after controlling for initial quality or end-to-end training cost.

Table 14: Per-suite change after source multi-task training with $\pi_{0.5}$. For initialization m and suite g, each cell reports $S_{m,g}(0) \rightarrow S_{m,g}(B)$ and the signed change in percentage points, where $S_{m,g}(u)$ is suite success after u updates. Negative changes indicate forgetting; positive changes indicate improvement. All arms share budget B.

Initialization	Spatial	Object	Goal	Long
Common base	$0 \rightarrow 92$ $(+92)$	$1 \rightarrow 100$ $(+99)$	$0 \rightarrow 93$ $(+93)$	$0 \rightarrow 91$ $(+91)$
Model Soups	$70 \rightarrow 91$ $(+21)$	$60 \rightarrow 100$ $(+40)$	$49 \rightarrow 94$ $(+45)$	$11 \rightarrow 96$ $(+85)$
RegMean++	$76 \rightarrow 94$ $(+18)$	$85 \rightarrow 100$ $(+15)$	$52 \rightarrow 94$ $(+42)$	$22 \rightarrow 91$ $(+69)$
FeatCal	$80 \rightarrow 94$ $(+14)$	$88 \rightarrow 99$ $(+11)$	$55 \rightarrow 96$ $(+41)$	$39 \rightarrow 92$ $(+53)$
TCR	$88 \rightarrow 97$ $(+9)$	$96 \rightarrow 99$ $(+3)$	$72 \rightarrow 97$ $(+25)$	$59 \rightarrow 94$ $(+35)$

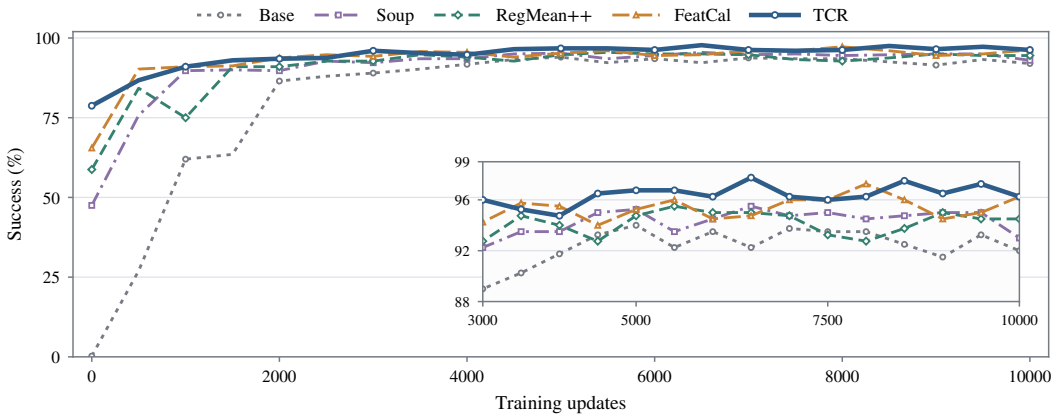

Figure 6: Further training from different initializations on the same LIBERO source tasks, through 10,000 updates. Each point averages success across four suites over 400 episodes; the inset enlarges the 3,000–10,000-update phase.

C.5 Cross-Benchmark Protocol

The reported benchmark comparisons use $\pi_{0.5}$ on LIBERO-PRO and RoboTwin 2.0. The PRO policies use the same four 10k-step LIBERO experts or their frozen mergers; archived 15k expert results are not substituted. No model is trained, recalibrated, or selected on PRO. The retained PRO comparison includes Model Soups, TIES, RegMean++, and FeatCal. All four mergers, TCR, and the preassigned expert reference have complete matched 10k first-repeat evaluations. These benchmarks differ in expert count, embodiment, and task difficulty; their scores do not isolate any one of those factors.

Our RoboTwin 2.0 evaluation uses a 30-task subset of its official 50-task list on Aloha-AgileX, partitioned into three ten-task expert groups. Demonstration counts, evaluation conditions, trial counts, and simulator and adapter revisions are fixed before the benchmark runs. Each main-table column refers to one specified evaluation setting. The expert reference executes the preassigned specialist on each task, rather than choosing an expert by test success. Image decoding, RGB ordering, and both arms' state/action dimensions are checked against the native evaluator. Training demonstrations, calibration traces, development states, and final resets have separate roles. Public generalist checkpoints do not substitute for independently trained task-group experts.

C.6 Benchmark-Specific Expert Groups and Evaluation Stages

The benchmark study tests frozen-policy robustness on LIBERO-PRO and skill retention after merging a separately trained RoboTwin expert bank. RoboTwin adds bimanual manipulation. Neither extension requires an unseen-task or composition claim. Expert partitions, source tasks, and evaluation conditions are fixed before training, without using comparative merger success. Negative

results remain part of the evaluation. Continued training is studied on LIBERO. The evaluated task lists and expert assignments are recorded separately for each benchmark.

LIBERO and LIBERO-PRO. The four LIBERO suites define four experts of ten tasks each. PRO reuses their frozen merged checkpoints and normalization/configuration without additional training, merging, trace collection for calibration, or hyperparameter selection (Zhou et al., 2026). We record checkpoint hashes for the retained mergers. Our $\pi_{0.5}$ study reports position, object, language, and changed-goal perturbations individually and as an equally weighted four-condition average. This aggregate includes changed-goal performance, not only same-task robustness; clean LIBERO results are excluded and remain in Table 1. Independent runtime isolation prevents PRO's task registrations from modifying the clean LIBERO evaluator.

RoboTwin groups. RoboTwin does not supply LIBERO-style expert suites. Our study uses the selected Coordination, Receptacle, and Precision groups, with task assignments recorded in the expert-bank manifest. This subset is not a full 50-task benchmark result. All merging methods receive the same source experts and use the same task list and evaluation conditions. Equal task and demonstration counts do not imply equal frame counts, difficulty, or training compute; these quantities are recorded for each expert group.

C.7 Cross-Benchmark Results by Task Group

The main table reports one task-averaged score per benchmark and method. Table 15 gives the group breakdowns, and Table 12 separates PRO perturbation types. Task IDs, group membership, demonstration counts, expert assignments, and evaluation conditions accompany the final task list.

Let $s_{m,k,t}$ be method m's success percentage on task t in evaluation repeat k. For disjoint task groups $\mathcal{T}_g$ with $\mathcal{T} = \bigcup_g \mathcal{T}_g$, we compute

$$S_{m,k,g} = \frac{1}{|\mathcal{T}_g|} \sum_{t \in \mathcal{T}_g} s_{m,k,t}, \qquad S_{m,k} = \frac{1}{|\mathcal{T}|} \sum_g |\mathcal{T}_g| S_{m,k,g}. \tag{15}$$

The expert reference substitutes the score of task t's preassigned specialist for $s_{m,k,t}$. It is an empirical reference, not an upper bound or a single generalist policy. All other method columns use one merged checkpoint per repeat across the complete task set.

For these two benchmarks, we report one fixed evaluation repeat (repeat-01) for every method, without cross-repeat standard deviations or selection of the highest-scoring repeat. LIBERO-PRO uses evaluation seed 274001 and its frozen perturbation/reset selections: ten episodes on each of 40 tasks under each of four perturbation types, totaling 1,600 episodes per method. RoboTwin uses the original repeat-01 task-specific seed/reset manifests: six episodes on each of 30 tasks, totaling 180 episodes per method; a single repeat does not mean using one identical environment seed for all episodes. The expert bank, hyperparameters, and corresponding first-repeat checkpoints remain fixed, and methods use paired task resets and evaluation seeds. Group and overall scores are computed from this complete repeat only; these results do not estimate variation across repeats or expert-training runs.

FeatCal on RoboTwin. The evaluated FeatCal adaptation uses static observations with three noise replicas, all at $\tau_{\text{flow}} = 1$. Its two time-MLP input caches consequently have rank one: the 450 calibration rows repeat the same 1024-dimensional time input. The FeatCal update in the unobserved complement is governed by its anchor rather than by feature matches at later flow times. Relative to the initial Model Soup, the time-conditioning vector changes by 3.28% at $\tau_{\text{flow}} = 1$ but 90.35% at $\tau_{\text{flow}} = 0.6$. In the 180 paired RoboTwin episodes, FeatCal has a systematic first-action offset relative to Soup and RegMean++, and several failed trajectories show absent or delayed task-relevant gripper closure; this behavior is not universal across failures.

To probe the time path, we restored only the two time-MLP weights from Soup in memory, leaving the other 811 state tensors unchanged, and ran native action generation on nine fixed calibration observations spanning the three task groups. The mean L2 distance of the first normalized action to its expert fell from 0.1765 to 0.1052; executed arm-target and gripper-command errors also decreased. This single-factor offline intervention shows that time-path drift contributes to action error. The nine

Table 15: Group-level cross-benchmark success with $\pi_{0.5}$ (%). LIBERO-PRO groups tasks by their source LIBERO suite; RoboTwin uses three preassigned expert groups. The overall score weights source tasks equally.

(a) LIBERO-PRO

Method	Spatial	Object	Goal	Long	Average ↑
Experts^{ref}	54.75	56.75	46.75	44.75	50.75
Model Soups	45.75	35.25	30.25	4.00	28.81
TIES	46.25	34.50	19.50	8.25	27.13
RegMean++	45.00	52.50	36.50	7.50	35.38
FeatCal	**56.25**	**58.25**	34.00	17.00	41.38
TCR	55.25	56.25	**40.00**	**21.25**	**43.19**

(b) RoboTwin 2.0

Method	Coordination	Receptacle	Precision	Average ↑
Experts^{ref}	60.00	56.67	56.67	57.78
Model Soups	3.33	20.00	**13.33**	12.22
TIES	3.33	10.00	5.00	6.11
RegMean++	3.33	25.00	11.67	13.33
FeatCal	0.00	0.00	0.00	0.00
TCR	**5.00**	**33.33**	8.33	**15.56**

observations are not held out, and no closed-loop evaluation of this diagnostic variant is used here; neither the intervention nor the trajectory inspection establishes that this drift alone caused FeatCal's 0/180 success.

Expert-count interpretation. The benchmarks use different task partitions and expert counts. These comparisons also change data, embodiment, and difficulty, so they do not establish scaling with expert count.

Inference latency, memory, and checkpoint size use the shared hardware protocol in Appendix B.7.

C.8 Real-Robot Protocol Details

The hardware study evaluates $\pi_{0.5}$ on a PiperX robot, using specialists fine-tuned on their respective tasks from the same base checkpoint. All newly introduced interfaces share their initialization before fine-tuning; camera processing, action representation, control rate, and normalization conventions remain compatible across specialists. Each merged checkpoint is evaluated under task instructions, without an expert router or post-merge training. This experiment measures source-task retention without post-merge training.

The comparison includes each task's corresponding specialist as an empirical reference, Model Soups, TIES, RegMean++, FeatCal, and TCR (Table 2). Every merger receives the same frozen expert bank. TIES supplies a sign-resolved, sparsified parameter-space baseline on the same materialized expert deltas (Yadav et al., 2023). Its retained density and task-vector scale are fixed using development data, with the selection budget disclosed and no tuning on final hardware trials. This is a TIES merger under the common-expert protocol, not a reproduction of the native MergeVLA system with its distinct expert construction and deployment setting (Fu et al., 2026). Calibration inputs for the hardware study are collected from physical executions of the frozen specialists. These real-robot expert rollouts, rather than demonstration observations or simulator trajectories, are the calibration source for RegMean++, FeatCal, and TCR. Regression methods share the recorded observation source and native input-generation convention; method-specific pass counts and realized row budgets are recorded separately. RegMean++ uses the explicitly documented stabilized VLA adaptation, with one sequential pass and equal expert masses; it is not the two-pass uniform-weight TCR ablation or a claim of an unmodified original solver. FeatCal starts from the uniform expert

mean and computes its own forward-order feature statistics and calibration updates; it is not implemented by relabeling TCR or reusing TCR-specific weighting. Solver, regularization, and replay differences are documented, so the hardware comparison is not interpreted as a pure weighting effect. Calibration collection and development resets are separate from final evaluation resets. Task 1 is plate placement and Task 2 is stacking, as identified in Table 2.

Offline action agreement. Table 2 additionally reports overall action RMSE against the corresponding frozen specialist on recorded observations for stacking and plate placement. The merged policy and reference expert receive identical observations and initial generation noise. For each task, let N_{req} denote the number of recorded requests, $H = 16$ the predicted action horizon, and $D_a = 7$ the action dimension (six joint coordinates and one gripper coordinate). The scalars $\hat{a}_{r,h,p}$ and $a^E_{r,h,p}$ denote merged and expert predictions at request r, horizon step h, and coordinate p. We pool squared errors over requests, future steps, and action coordinates before taking the square root:

$$\mathrm{RMSE}_{\mathrm{expert}} = \sqrt{\frac{1}{N_{\mathrm{req}} H D_a} \sum_{r=1}^{N_{\mathrm{req}}} \sum_{h=1}^{H} \sum_{p=1}^{D_a} \left(\hat{a}_{r,h,p} - a^E_{r,h,p} \right)^2}. \tag{16}$$

This is not the mean of per-request RMSE values, and it covers the predicted action chunk rather than only the executed portion. The reference is the expert's prediction, not an action label from the training dataset; errors against recorded training labels measure a different quantity and are not included in this table. Lower action RMSE indicates closer numerical agreement on the recorded inputs, not a guarantee of successful closed-loop execution. Unlike the simulation diagnostic, this hardware metric includes the gripper and all 16 predicted steps; the simulation metric uses normalized MSE on six continuous coordinates over ten executed steps. Their numerical magnitudes are not directly comparable.

Trial budget. The reported comparison contains 20 trials per method per task, for 40 trials per method. Each task attempt has a maximum execution time of 60 s for every method, including the expert reference; failure to complete the task within this limit counts as a failed trial. The five merged policies and the expert reference account for 240 reported trials in total. The expert-reference row evaluates only the corresponding specialist on each task, for 20 trials per specialist and 40 reference trials in total; specialists are not evaluated on the other task.

Reporting and attempt accounting. We report per-task success percentages from 20 trials and their equally weighted average. These trials are not independent training or merging repeats. The main table reports full-task autonomous success; any partial-progress scores are reported separately under a predefined rubric. Timeouts, human interventions, and safety stops are logged; intervention-assisted completion is not counted as autonomous success. Policy-induced timeouts, interventions, and safety stops count as failures, not technical-invalid exclusions. Any technical-invalid exclusions and replacement attempts use predefined, method-independent criteria and retain a complete attempt log. Hardware qualification requires operator supervision, tested emergency stopping, and platform-appropriate workspace, speed, and force limits before policy rollouts. Cost accounting includes robot time, calibration attempts, operator resets, and interventions in addition to training and fusion compute.