MIXTURE-OF-CHECKPOINTS: IMPROVING INITIAL-STATE COVERAGE OF VISION-LANGUAGE-ACTION POLICIES VIA ONLINE ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training for vision-language-action policies usually keeps the checkpoint with the highest aggregate success rate and discards the rest. This selection ignores whether different checkpoints cover different initial states. We study whether checkpoints that do not improve aggregate success can still improve initial-state coverage. We post-train OpenVLA-7B with Direct Preference Optimization on action preference pairs derived from Monte Carlo tree search value estimates; each post-training run yields one checkpoint. The resulting checkpoints match or underperform the base model on LIBERO. However, a two-round per-state audit shows that they succeed on different initial states. Together, they cover most of the initial states where the base model fails. Therefore, we deploy the base model and all checkpoints as a policy portfolio, which we call Mixture-of-Checkpoints, and add a router that selects one policy per episode. The router is a per-state contextual bandit: the initial-state index is the context, episode success is the reward, and an exponential moving average updates a per-state score table. Across eight LIBERO tasks, routing matches or outperforms the base model. On stable tasks with headroom, it adds 8–19 successful episodes per 100 episodes. Two tasks selected after the method was fixed confirm predictions recorded in advance. Ablations show that the gain requires online per-state measurement, and a two-round audit predicts which tasks benefit.

1 INTRODUCTION

Post-training for a robot policy usually produces several checkpoints. A standard pipeline keeps the checkpoint with the highest aggregate success rate and discards the rest. However, checkpoints with similar aggregate scores may succeed on different initial states. We ask whether these differences can improve the initial-state coverage of a vision-language-action (VLA) policy. We propose *Mixture-of-Checkpoints*, which keeps the base policy and its post-training checkpoints as a policy portfolio and routes each initial state to one policy online (Figure 1).

We study preference-based post-training of OpenVLA-7B on LIBERO (Kim et al., 2024; Liu et al., 2023). We construct action preference pairs from value estimates of Monte Carlo tree search (MCTS) (Coulom, 2006; Browne et al., 2012) and use direct preference optimization (DPO) (Rafailov et al., 2023) to fine-tune low-rank adaptation (LoRA) adapters (Hu et al., 2022) under different random seeds and training variants. We call the resulting checkpoints members. No member consistently improves the aggregate success rate over the base policy. On obj3, for example, the base policy achieves 75 successes in 100 episodes, while the five members achieve 62 to 72.

A two-round per-state audit gives a different result. We evaluate every policy twice on each of the 50 fixed initial states of a task and label the result as solid, split, or dead. The members succeed on different states from the base policy and from each other. On obj3, 11 of the 12 base-dead states have at least one solid member. The aggregate success rate therefore hides differences in initial-state coverage.

Mixture-of-Checkpoints uses these differences at deployment time. All policies remain frozen, and a per-state contextual bandit with the initial-state index as context, ϵ-greedy selection (Sutton & Barto, 2018), and an exponential moving average over a per-state score table selects one policy for

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.

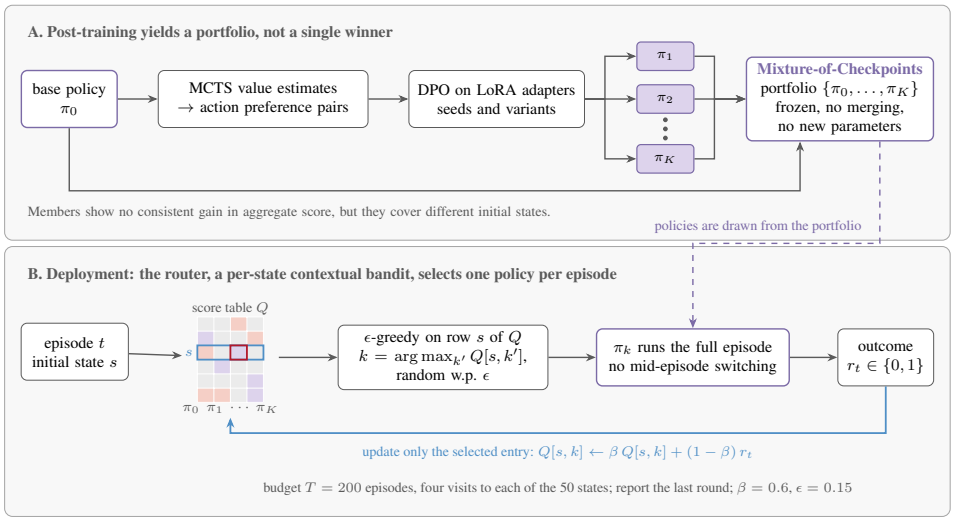


Figure 1: Overview of Mixture-of-Checkpoints. (A) Post-training yields checkpoints that do not improve aggregate success but cover different initial states, so the base policy and all checkpoints form a frozen portfolio. (B) At deployment, the router, a per-state contextual bandit over a score table Q, selects one policy per episode by ϵ-greedy on row s, runs the full episode with it, and updates only the selected entry from the outcome.

each full episode. This design borrows the experience-driven routing rule of AgentNet (Yang et al., 2025) for a portfolio of robot policies without training another model.

Across eight LIBERO tasks, online routing matches or outperforms the base policy. On stable tasks with headroom, it adds 8 to 19 successes per 100 episodes, for example from 75 to 94 and 90 on obj3 and from 70 to 88 on t8. Two tasks selected after the method and hyperparameters were fixed confirm predictions recorded before evaluation. Under the same budget, UCB1 (Auer et al., 2002) performs worse, Thompson sampling (Thompson, 1933; Chapelle & Li, 2011) performs similarly on most stable tasks, and AgentNet routing on first-frame features (Yang et al., 2025; Li et al., 2010) does not exceed our method on any stable task. Weight averaging (Wortsman et al., 2022), a learned gate (Buehler & Buehler, 2024; Yang et al., 2026), a frozen routing table, and a router without state context do not reliably recover the gain, while random LoRA perturbations recover most of it, which shows that weight-space diversity itself creates useful coverage.

The audit also identifies the scope of the method. Routing works on all six tasks with stable per-state outcomes and on neither of the two tasks whose outcomes are no more consistent than independent trials. Per-state outcomes are repeatable within one process but not reliably across processes, so the routing table must learn within the deployment process, and the method requires repeated and identifiable initial states.

We make three contributions. First, we show that post-training checkpoints with weak aggregate scores can provide complementary initial-state coverage, and we introduce a two-round audit that measures this structure. Second, we propose Mixture-of-Checkpoints, online per-state routing over the base policy and its checkpoints. Third, we show that audit stability predicts where routing works and identify within-process repeatability as an important condition. Section 3 describes the method, Section 4 the setup and audit, Section 5 the experiments and ablations, Section 6 the scope of the method, and Section 7 the limitations.

2 Related Work

Post-training of vision-language-action policies. OpenVLA fine-tunes a large vision-language model into a robot policy and releases public LIBERO checkpoints (Kim et al., 2024; Liu et al., 2023). Later work seeks a stronger single policy through parallel decoding, action chunking, and continuous actions in OpenVLA-OFT (Kim et al., 2025), outcome-driven reinforcement learning in

SimpleVLA-RL (Li et al., 2026), and preference alignment in GRAPE (Zhang et al., 2024). We instead study post-training that produces no stronger single policy, and we deploy all checkpoints together to improve initial-state coverage.

Preference-based fine-tuning with search. DPO optimizes a policy on preference pairs without an explicit reward model (Rafailov et al., 2023). We construct these pairs from MCTS value estimates (Coulom, 2006; Browne et al., 2012) and fine-tune LoRA adapters (Hu et al., 2022), following the search-and-distillation idea of AlphaGo and Expert Iteration (Silver et al., 2016; Anthony et al., 2017). However, our search produces preference pairs instead of action labels, and fine-tuning produces members for routing rather than one stronger policy.

Merging, gating, and checkpoint selection. Weight averaging merges models (Wortsman et al., 2022). Mixture-of-experts methods learn a gate (Jacobs et al., 1991; Shazeer et al., 2017), X-LoRA and InstructVLA gate LoRA experts with a learned router (Buehler & Buehler, 2024; Yang et al., 2026), and sparse upcycling initializes an MoE from a dense checkpoint (Komatsuzaki et al., 2023). Germino et al. (2025) use the name Mixture of Checkpoint Experts for per-sample selection among checkpoints from one training run, whereas our checkpoints come from separate post-training runs and our router learns online for each repeated initial state. Ensemble methods use member diversity (Dietterich, 2000), MG-Select selects among action samples from one model (Jang et al., 2026), and algorithm portfolios select one solver per instance (Rice, 1976; Gomes & Selman, 2001). Our method follows per-instance selection without merging weights or training a gate.

Bandits and randomness in fine-tuning. Our router is a contextual bandit whose context is the initial state (Lattimore & Szepesvári, 2020), with ϵ-greedy selection (Sutton & Barto, 2018). We compare it under the same budget with UCB1 (Auer et al., 2002), Thompson sampling (Thompson, 1933; Chapelle & Li, 2011), and AgentNet routing, a LinUCB contextual bandit with a linear model on input features (Li et al., 2010). The selection rule is not our contribution. Our contribution is the policy portfolio, per-state granularity, and online measurement. Fine-tuning outcomes vary across random seeds and training runs, and prior work on language-model fine-tuning, underspecified systems, and deep reinforcement learning treats this variation as noise to control (Dodge et al., 2020; D'Amour et al., 2022; Henderson et al., 2018; Agarwal et al., 2021). We measure it per initial state and use the resulting coverage for routing.

Multi-agent routing. LLM-based multi-agent systems assign tasks through predefined workflows (Wu et al., 2023) or decentralized routing. AgentNet routes tasks with capability vectors updated from experience (Yang et al., 2025). We apply this routing idea to frozen robot policies, where policies act as agents, initial states act as tasks, and episode outcomes update a per-state score table. The policies do not communicate or evolve, so we borrow only the routing rule, and our AgentNet routing baseline instantiates its capability vectors with first-frame visual features.

3 Mixture-of-Checkpoints: Online Per-State Routing

Portfolio and score table. Mixture-of-Checkpoints deploys the base policy and K members as a portfolio of $K + 1$ frozen policies. It does not merge weights or train new parameters. The router stores a score table Q with one row per initial state and one column per policy. Each entry $Q[s, k]$ estimates the success of policy k on state s. A cold start sets all entries to 0.5. A warm start sets entries labeled solid by the audit of Section 4 to 0.9 and leaves the others at 0.5. Section 5 compares the two.

Selection and update. At the start of each episode, the router reads the initial-state index s. With probability ϵ, it selects a policy uniformly at random. Otherwise, it selects a policy with the largest value in $Q[s, \cdot]$. It breaks ties at random. This is ϵ-greedy selection (Sutton & Barto, 2018). Mid-episode switching would mix the action distributions of different policies, so the selected policy executes the full episode. The episode t returns $r_t = 1$ for success and $r_t = 0$ for failure. The router updates only the selected entry:

$$Q[s, k] \leftarrow \beta Q[s, k] + (1 - \beta) r_t.$$

We use $\beta = 0.6$ and $\epsilon = 0.15$. Section 5 shows that the results are insensitive to both values. This problem is a contextual bandit (Lattimore & Szepesvári, 2020). The initial-state index is the context, the policies are the arms, and episode success is the reward. Algorithm 1 gives the full procedure.

Algorithm 1 Mixture-of-Checkpoints: online per-state routing

Require: frozen policies $\{\pi_k\}_{k=0}^{K}$, number of initial states S, budget T, β, ϵ, and initial table Q_0
Ensure: score table Q and outcomes $r_1, \ldots, r_T$ of the routed episodes

$Q \leftarrow Q_0$
for $t = 1$ to T **do**
  $s \leftarrow ((t-1) \bmod S) + 1$
  if $u \sim \mathrm{Uniform}(0,1)$ satisfies $u < \epsilon$ **then**
    sample k uniformly from $\{0, \ldots, K\}$
  else
    $k \leftarrow \arg\max_{k'} Q[s, k']$, with ties broken at random
  end if
  $r_t \leftarrow \mathbf{1}[\text{episode from } s \text{ under } \pi_k \text{ succeeds}]$
  $Q[s,k] \leftarrow \beta Q[s,k] + (1-\beta) r_t$
end for

Budget and reporting. Each deployment contains 200 episodes and cycles through the 50 initial states in index order. Each state is therefore visited four times, which gives four rounds. We double the number of successes in the fourth round and report it as successes per 100 episodes. Figure 3 reports all four rounds. Each configuration of our method uses two random seeds. The seeds affect only exploration and tie breaking. A 50-episode round has a standard error of about three successes, or six points per 100. We treat results within six points as comparable.

Baselines under the same budget. We keep the portfolio, budget, and reporting protocol fixed and change only the selection rule. UCB1 (Auer et al., 2002) stores a visit count $N[s,k]$ and a success count $W[s,k]$. It selects unvisited policies first. It then selects the policy with the largest value of

$$\frac{W[s,k]}{N[s,k]} + \sqrt{\frac{2 \ln n_s}{N[s,k]}},$$

where n_s is the total number of visits to state s. Thompson sampling (Thompson, 1933; Chapelle & Li, 2011) stores the posterior

$$\mathrm{Beta}\big(1 + W[s,k],\ 1 + N[s,k] - W[s,k]\big)$$

for each entry. At each episode, it draws one sample for each policy and selects the largest one. The third baseline implements the routing rule of AgentNet (Yang et al., 2025) on frozen policies, which we call AgentNet routing. AgentNet represents each agent by a capability vector, routes a task to the agent whose capability vector is most similar to the task embedding, and updates the capability vectors from task outcomes. Here the task embedding is a context vector x from the first frame, the mean-pooled tokens of the frozen vision backbone projected by a fixed random matrix to 32 dimensions, unit-normalized, plus a bias term. Each policy k keeps a capability vector $\theta_k = A_k^{-1} b_k$, where $b_k = \sum r_t x$ accumulates the features of the episodes on which it succeeded and $A_k = I + \sum x x^\top$ normalizes them, with both sums over the episodes in which policy k was selected. The router selects the policy with the largest similarity plus an exploration bonus, which is the LinUCB rule (Li et al., 2010):

$$\theta_k^\top x + \alpha \sqrt{x^\top A_k^{-1} x}, \qquad \alpha = 1.$$

It keeps the three elements of AgentNet routing, a task feature, a per-policy capability vector updated from outcomes, and similarity-based selection. It drops the cosine similarity and moving-average update in favor of the LinUCB estimate, adds an exploration term, and has no communication, memory, or agent evolution, which have no counterpart for frozen policies. It therefore conditions on visual features where our method conditions on the state index. All three baselines see the same episode outcomes as our method and differ only in how they select the next policy. Each state has only four visits. Therefore, UCB1 never repeats a policy when $K + 1 \geq 4$ and leaves some policies unvisited when $K + 1 > 4$. Section 5 uses this fact.

Alternatives. Our router trains no parameters and only updates a table. Section 5 compares it with weight averaging, a learned gate on first-frame features, a frozen routing table, and a router that shares one table row across all states. Only our method keeps measuring member differences per state during deployment.

4 Set-Up and Per-State Audit

Tasks and base policies. We evaluate eight LIBERO tasks (Liu et al., 2023): five from `libero_10` (t07, t1, t3, t5, t8) and three from `libero_object` (obj0, obj2, obj3), named by their suite indices. Each task has 50 fixed initial states, and each episode starts from one of them. The base policies are the public OpenVLA-7B checkpoints for the two suites (Kim et al., 2024). We run them with greedy decoding, the official image preprocessing, and a 224-pixel center crop. An episode is successful if the task is completed within the benchmark's default step limit. We report successes per 100 episodes. One 100-episode evaluation cycles through the 50 initial states twice. Each evaluation runs in its own process with one policy and one simulator instance. Section 6 uses this distinction.

Members. While the base policy executes each episode, an action-space MCTS planner (Coulom, 2006; Browne et al., 2012) is called every 12 steps, or every 4 steps in critical stages, runs 8 simulations to depth 3, assigns a search value to each candidate action chunk, and may override the greedy action with the best candidate. Each planning step pairs the highest- and lowest-valued candidates and yields at most 8 pairs, pairs from failed episodes are weighted 0.15 instead of 1.0, and 100 episodes per task yield 949 to 3,540 pairs. We then fine-tune LoRA adapters (Hu et al., 2022) of the base policy with DPO (Rafailov et al., 2023) at rank 16 for one epoch, with the remaining settings in Appendix B. For obj0, obj2, obj3, t1, t5, and t8, we train five members with five random seeds; the three t07 members are training variants, two DPO members with different pair-construction rules and one trained with a reinforcement learning objective and a verifiable outcome reward (Li et al., 2026).

The DPO objective decreases only slightly, and the mean per-layer Frobenius norm of the seed members' adapter updates ranges from 0.003 to 0.006, so these members stay close to the base policy in weight space, while two t07 variants move further. Table 4 in Appendix B reports each member's score over 100 episodes, and no member shows a consistent improvement over the base policy.

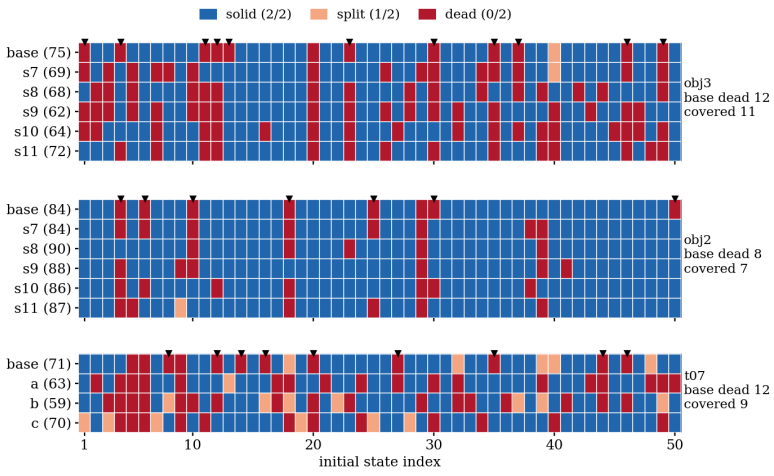


Figure 2: Two-round per-state audit on obj3, obj2, and t07. Rows are policies with successes per 100 episodes in parentheses, and columns are the 50 initial states. Blue is solid (2/2), orange is split (1/2), and red is dead (0/2). Triangles mark base-dead states with at least one solid member. Aggregate scores hide that the members cover most base-dead states.

Two-round per-state audit. We run two episodes for each policy-state pair and label the pair solid (2/2), split (1/2), or dead (0/2); σ denotes a policy's number of split states. If every episode were an independent trial with success probability p, the expected split count over 50 states would be $50 \times 2p(1-p)$, about 21 for p near 0.7. For the base policy (Table 7 in Appendix E), σ ranges from 0 to 5 on six tasks, far below this reference, and is close to it on t1 and t3, so within one evaluation process the outcome of a policy-state pair is near-deterministic on most tasks and near-random on t1 and t3. The coverage of a policy is the set of states on which it is solid, and the coverage of a portfolio is the union of these sets.

Observation: no single winner but different coverage. The audit shows a different picture at the state level (Figure 2). On obj3, the base policy has 37 solid, 1 split, and 12 dead states, and the five members score 62 to 72 against 75, yet 11 of the 12 base-dead states have at least one solid member, so the portfolio covers 11 states that the base policy does not; the counts are 7 of 8 on obj2 and 9 of 12 on t07. Each member also fails on some base-solid states: on obj3, every member rescues 3 to 5 base-dead states but loses 4 to 10 base-solid states, which explains its lower aggregate score. Selecting the best policy for each state gives an upper bound for any per-state selector, the per-state oracle (column Oracle in Table 1), which is 93 on t07 against 71 for the best single policy. Mixture-of-Checkpoints does not use the audit at deployment and approaches this bound from online episode outcomes alone.

5 EXPERIMENTS

Protocol. All experiments use the tasks, policies, and protocol of Sections 4 and 3. Base-policy and member scores come from separate 100-episode evaluations on the same reporting scale, and Appendix A lists all runs.

Main results. Table 1 reports all eight tasks in increasing order of σ.

Table 1: Main results under the same 200-episode budget. σ is the base policy's split count. Oracle is the per-state oracle. AgentNet routing is the routing rule of AgentNet implemented with LinUCB on first-frame visual features (Section 3). The t5 and t8 oracle values are lower bounds because their members are partially audited. Two values are two seeds. A dash means not measured.

Task	σ	Base	Best member	Oracle	UCB1	Thompson	AgentNet routing	Ours
obj0	0	96	93	100	98	98	98	**98 / 98**
obj2	0	84	90	98	90	88 / 92	86 / 84	**94 / 94**
obj3	1	75	72	98	68	84 / 76	68 / 72	**94 / 90**
t5	2	64	69	88	70	64 / 76	70 / 64	**80 / 72**
t8	4	70	71	88	50 / 68	86 / 76	80 / 78	**88 / 88**
t07	5	71	70	93	78 / 68	78 / 86	82 / 80	**84 / 86**
t1	16	76	79	96	80	**88 / 78**	78	84 / 74
t3	19	37	–	–	46	40	34	38 / 46

Routing matches or outperforms the base policy on all six tasks with $\sigma \leq 5$. On the five tasks with headroom, it adds 8 to 19 successes per 100 episodes. For example, obj3 rises from 75 to 94 and 90, and both t8 runs rise from 70 to 88. The final scores are 0 to 16 points below the oracle, and both t8 runs reach it. The oracle is 4 to 23 points above the best single policy, so routing recovers most of the portfolio's coverage gain within 200 episodes. Routing gives no consistent gain on t1 and t3, where σ is 16 and 19. Section 6 analyzes these tasks.

In the learning curves (Figure 3), first-round scores fluctuate around the base policy because most selections are first trials, and from the second round routing matches or outperforms the base policy on all six stable tasks.

Prospective tasks. We selected obj3 and obj2 after fixing the method, hyperparameters, and reporting protocol. We used only the four base-policy screening scores of 93, 84, 75, and 94 and selected the two tasks with the largest headroom. Before training any member, we predicted at least

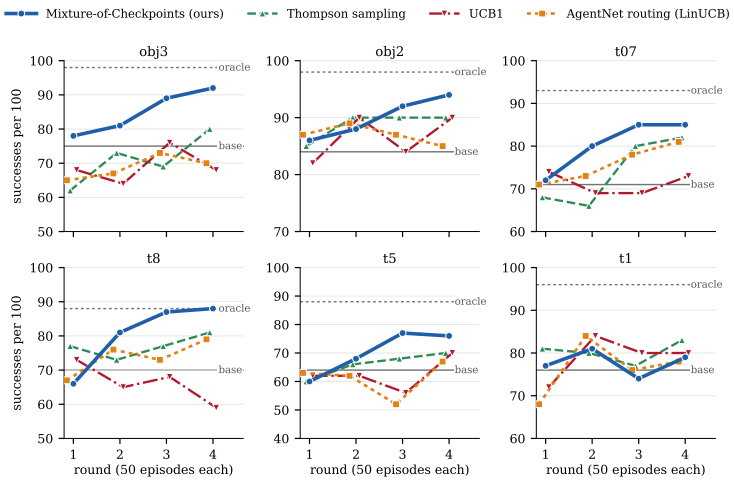


Figure 3: Four-round learning curves under the same 200-episode budget. Each curve is the mean over seeds, and the vertical range is chosen per task. Horizontal lines show the base policy and the per-state oracle. Routing improves on stable tasks and gives no consistent gain on t1.

85 on obj3 and at least 90 on obj2. Both predictions held in all four runs (Table 5 in Appendix C): routing reaches 94 and 90 on obj3 and 94 and 94 on obj2. All five obj3 members score below the base policy. These results provide prospective evidence because task selection and predictions preceded the routing results.

Routing baselines. All baselines route among the same portfolio and differ only in the selection rule; UCB1, Thompson sampling, and AgentNet routing update online, while the learned gate and the frozen table (Section 5) do not. UCB1 (Auer et al., 2002) performs worse than our method on every task with headroom: with four visits per state, it spends them on unvisited policies, never repeats a policy in the final round, and stays near the average member level. Thompson sampling (Thompson, 1933; Chapelle & Li, 2011) stays within the noise band of our method on the other stable tasks but scores 84 and 76 on obj3 against our 94 and 90. AgentNet routing stays 8 to 26 points below both of our runs on obj2, obj3, and t8, is within the noise band of at least one of our runs on t5 and t07, gives no consistent gain on t1 and t3, does not improve with α of 0.3 or 3.0 (Table 2), and selects a single member in at least 35 of 50 final-round episodes on obj3 and t07, so first-frame features do not separate the 50 initial states. Neither the update formula nor a feature-based context is therefore the source of the gain, which comes from the portfolio, per-state selection on the state index, and online measurement. A full AgentNet baseline is not possible here: its agents are language models that decompose tasks, exchange context, retrieve past trajectories, and revise their prompts and capability vectors, whereas a frozen VLA policy maps an image and an instruction to actions, so none of these mechanisms has a counterpart, and the task embedding can only come from the first frame because all episodes of a task share one instruction. The routing rule is the only transferable component, which AgentNet routing implements, so the comparison of selection rules above is the meaningful one.

Ablations. Table 2 reports the ablations, on t07 unless noted otherwise.

Why not merge the members? Weight averaging (Wortsman et al., 2022) scores 55 on t07 and 63 on t5, both below the base policies. It cancels coverage differences instead of combining them.

Why not learn a gate? A learned gate in the style of X-LoRA and InstructVLA (Buehler & Buehler, 2024; Yang et al., 2026) reaches 0.54 accuracy under five-fold cross-validation, compared with 0.72 for the majority-member baseline. On seen states, it makes the same selection as the frozen table in all 100 episodes and scores 82. On 20 novel states, it scores 80 against 85 for the base policy.

Why update continually? A table transferred to a new process scores 74 and 76 after cold-start learning and 92 and 72 after warm-start learning. These results are less consistent than online routing, which scores 84 to 88. Section 6 explains this result.

Table 2: Ablation results. Unless noted otherwise, results use t07 (base 71).

Ablation	Setting	Result	Reference
Weight averaging (soup)	members merged into one model	t07 55; t5 63	base 71; 64
Learned gate (X-LoRA / InstructVLA style)	MLP on first-frame features	seen 82; novel 80	frozen table 82; base 85
	five-fold CV accuracy	0.54	majority-member 0.72
Frozen table	table from a previous process, no updates	cold 74 / 76; warm 92 / 72	online 84 to 88
Warm start	audit-prior initialization, updates on	86 / 88	cold 84 / 86
Update rate β	0.3; 0.9	90; 90	0.6: 84 / 86
Exploration ϵ	0.05; 0.3	92; 88	0.15: 84 / 86
State context	one shared table row for all states (obj3; t07)	70; 76	per-state 94 / 90; 84 / 86
Member source (obj3)	five untrained random-LoRA members, norm-matched to DPO	84 / 86	DPO 94 / 90; base 75
	same members with $\|\Delta W\|_F$ at $0.1\times$	84	
Member source (t07)	random-LoRA members, norm-matched to the t07 members	82	DPO 84 / 86; base 71
AgentNet routing α (obj3)	0.3; 3.0	68; 74	1.0: 68 / 72

Does the state context matter? We keep the portfolio, the ϵ-greedy rule, and the update, but give all initial states one shared table row, so the router learns one global preference instead of a per-state one. This router scores 70 on obj3 and 76 on t07, compared with 94 and 90 and with 84 and 86 for per-state rows. The gain disappears without per-state measurement.

How sensitive are the hyperparameters? Warm starts and the tested values of β and ϵ stay within six points of at least one default run. The two β settings select the same policy in 198 of 200 episodes with identical outcomes, because with four visits per state β changes the score values but rarely their ordering.

Where do the members come from? We replace the DPO members with untrained random-direction LoRA perturbations whose per-layer Frobenius norms match the DPO adapters. Routing over them reaches 84 and 86 on obj3 and 82 on t07, against 94 and 90 and 84 and 86 for the DPO portfolios, and a tenfold smaller perturbation still gives 84. Weight-space diversity therefore provides most of the gain, and DPO adds 4 to 10 points on obj3 and nothing beyond the noise band on t07.

Single-model test-time sampling. MG-Select (Jang et al., 2026) improves over the base policy only on t3, where it scores 50 and 44 against 37 (Table 6 in Appendix D), so it and per-state routing work in complementary settings.

6 Analysis: Stability and Scope

σ predicts where routing helps. Figure 4 (left) plots routing gain against σ. On the six tasks with σ between 0 and 5, both routing runs exceed the noise band whenever the base policy has headroom. The exception is obj0, whose base-policy score is already 96. On t1 and t3, where σ is 16 and 19, the gains are not consistent across seeds, and their means remain inside the noise band. We do not claim a threshold because no task has σ between 5 and 16, and t3 has no member audit.

Figure 4 (right) and Table 7 show the mechanism. On the six stable tasks, the observed σ is far below the value expected under independent trials. On t1 and t3, it is close to the expected value. A split state behaves almost like a coin flip within one process. The initial-state index therefore carries little information on these tasks, and a per-state selector has little to learn. This explains why the four routing rules in Section 5 give no reliable gain on t3.

Within-process repeatability. The routing logs provide direct evidence. When the same policy is selected again on the same state within one deployment, the outcomes agree in 49/49 cases on obj3, 47/47 on obj2, 43/48 on t07, 42/47 on t5, and 40/49 on t8. The agreement falls to 31/50 on t1 and 16/41 on t3. Most final-round successes also repeat a policy that previously succeeded on the same state: 42/42 on obj3 and 35/36 on t07. This result explains why four visits per state are enough.

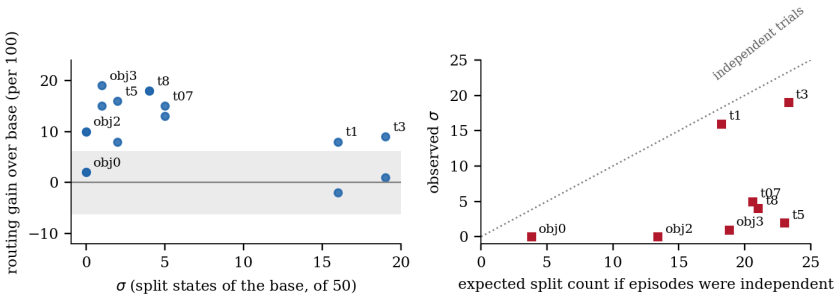


Figure 4: Left: routing gain over the base policy for both seeds against the base policy's split count σ. The gray band shows the six-point noise level. Right: observed σ against the split count expected under independent trials. Six tasks lie far below the line, while t1 and t3 lie close to it.

Cross-process repeatability and scope. The same base checkpoint keeps a similar aggregate score across processes (four t07 processes score 71, 70, 70, and 69), but whether its per-state labels are preserved depends on the task and the hardware. A second process on the same GPU type reproduces all 50 labels on obj2 and obj3 (Cohen's κ of 1.00 (Cohen, 1960)) but gives κ of 0.76 on t07 and 0.14 on t5, while processes on a different GPU type give 0.27 and 0.25 on t07, 0.14 on t5, 0.11 on t1, and -0.05 on obj0 (Table 7). Cross-process repeatability is therefore not guaranteed and cannot be known in advance, which explains the inconsistent frozen tables in Section 5. This defines the scope of the method: the routing table must learn online in the process where it is used, and the initial states must recur and be identifiable. The random-member control in Section 5 points the same way: differences between policies provide most of the gain, and DPO members add a further gain only on obj3.

Novel initial states. On 20 additional initial states of t07 that were not among the 50 audited states, cold-start online routing reaches 90 and warm-start routing 95 in the final round, against 85 for the base policy and 80 for the learned gate (Table 8 in Appendix F). Online routing therefore learns on novel states without an audit when those states recur during deployment, while the offline gate cannot.

7 Limitations

Our evidence has four limitations. First, all experiments use one simulator, one base-policy family, and one post-training method, with no real-robot experiments, so the results may not transfer. Second, our protocol revisits the same 50 initial states within one process, so the final round scores states the router has already seen three times; the novel-state experiment uses only 20 states from one task and still requires recurring states, and identifying repeated initial states in real deployment remains unsolved. Third, random-perturbation members recover most of the routing gain, so DPO training adds only 4 to 10 points on obj3 and nothing beyond the noise band on t07. Fourth, σ can change across processes (0 to 5 on t07 across four processes), so it is a diagnostic rather than a threshold; only two members are audited on t5 and t8, so their oracle values are lower bounds.

8 Conclusion

We study post-training that produces no stronger single policy. A two-round per-state audit shows that checkpoints with similar or lower aggregate scores cover different initial states and together cover most states where the base policy fails, and Mixture-of-Checkpoints recovers this coverage by routing among the base policy and all checkpoints online for each initial state. Across eight LIBERO tasks, routing matches or outperforms the base policy, adds 8 to 19 successes per 100 episodes on stable tasks with headroom, and confirms both prospective predictions. Audit stability predicts where routing works, and within-process and cross-process repeatability define the scope of the method. Future work should replace state indices with learned representations and test the same phenomenon with other post-training methods and real robots.

Reproducibility Statement

All results in this paper come from the runs listed in Appendix A. Appendix B gives the member training settings, and Appendix C gives the recorded predictions for the prospective tasks. The supplementary material contains the evaluation and routing code, configuration files for all runs, and the per-episode logs used to compute the tables and figures. The members' LoRA adapters will be released upon publication. All experiments use the public LIBERO benchmark and public OpenVLA-7B checkpoints.

Ethics Statement

This work uses only simulated robot manipulation tasks and publicly released models and benchmarks. It involves no human subjects, personal data, or real-world deployment. We see no direct ethical concerns beyond those common to robot learning research.

AI Use Statement

We used an AI assistant (Claude, Anthropic) to retrieve and identify related work and to improve the English and readability of the manuscript. The authors checked all suggested references against their original sources and reviewed all AI-edited text. The research questions, method, experimental design, interpretation, and claims are the authors' own. The authors take responsibility for the final content of this paper.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Eric L. Buehler and Markus J. Buehler. X-LoRA: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. *APL Machine Learning*, 2(2), 2024. arXiv:2402.07148.

Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, 2006.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 23(226):1–61, 2022.

Thomas G. Dietterich. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*, 2000.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

Joe Germino, Benjamin Brinkmann, Nitesh V. Chawla, Jie Cui, Emma Fortune, Nuno Moniz, and Louis Faust. Mixture of checkpoint experts for explainable seizure detection using wearable devices. *Scientific Reports*, 15:43976, 2025.

Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018.

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 (ICLR)*, 2022.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision-language-action models. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.05681.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024. arXiv:2406.09246.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Robotics: Science and Systems (RSS)*, 2025. arXiv:2502.19645.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations (ICLR)*, 2023.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, et al. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.09674.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *International Conference on World Wide Web (WWW)*, 2010.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. arXiv:2306.03310.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

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 *International Conference on Machine Learning (ICML)*, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. Vision-language-action instruction tuning: From understanding to manipulation. In *International Conference on Learning Representations (ICLR)*, 2026.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2504.00587.

Zijian Zhang, Kaiyuan Zheng, Zhaorun Chen, Joel Jang, Yi Li, Siwei Han, Chaoqi Wang, Mingyu Ding, Dieter Fox, and Huaxiu Yao. GRAPE: Generalizing robot policy via preference alignment. *arXiv preprint arXiv:2411.19309*, 2024.

A RUN RECORDS

Table 3 lists every run used in the paper with its tag, task, method, per-round successes, and reported score. Tags match the released configuration files and logs.

Table 3: All runs used in the paper. Per-round entries list successes in each 50-episode round; the score is the last round times two for routing runs and the total successes for 100-episode evaluations.

Run tag	Task	Method	Per round / episodes	Score
base6k_t07_eval	t07	base	100ep	71
t07_fresh_base	t07	base	100ep	70
baseH_t07_eval	t07	base	100ep	70
base6k_t1_eval	t1	base	100ep	76
t1_base_fresh	t1	base	100ep	79
base6k_t3_eval	t3	base	100ep	37
base6k_t5_eval	t5	base	100ep	64
baseH_t5_eval	t5	base	100ep	60
base6k_t8_eval	t8	base	100ep	70
baseH_obj0_eval	obj0	base	100ep	96
obj0_eval_base	obj0	base	100ep	94
objscreen_t1	obj_task1	base	100ep	93
objscreen_t2	obj2	base	100ep	84
objscreen_t3	obj3	base	100ep	75
objscreen_t4	obj_task4	base	100ep	94
t07_fresh_a	t07	member a	100ep	63
t07_fresh_b	t07	member b	100ep	59
t07_fresh_c	t07	member c	100ep	70
xrep_t1_twin_s7_eval	t1	member s7	100ep	78
xrep_t1_twin_s8_eval	t1	member s8	100ep	76
xrep_t1_twin_s9_eval	t1	member s9	100ep	79
xrep_t1_twin_s10_eval	t1	member s10	100ep	77
xrep_t1_twin_s11_eval	t1	member s11	100ep	77
xrep_t5_twin_s10_eval	t5	member s10	100ep	64
xrep_t5_twin_s11_eval	t5	member s11	100ep	69
xrep_t8_twin_s10_eval	t8	member s10	100ep	71
xrep_t8_twin_s11_eval	t8	member s11	100ep	59
obj0_eval_s7	obj0	member s7	100ep	93
obj0_eval_s8	obj0	member s8	100ep	90
obj0_eval_s9	obj0	member s9	100ep	90
obj2_eval_s7	obj2	member s7	100ep	84
obj2_eval_s8	obj2	member s8	100ep	90
obj2_eval_s9	obj2	member s9	100ep	88
obj2_eval_s10	obj2	member s10	100ep	86
obj2_eval_s11	obj2	member s11	100ep	87
obj3_eval_s7	obj3	member s7	100ep	69
obj3_eval_s8	obj3	member s8	100ep	68
obj3_eval_s9	obj3	member s9	100ep	62
obj3_eval_s10	obj3	member s10	100ep	64
obj3_eval_s11	obj3	member s11	100ep	72
router_ema_cold	t07	online routing (EMA)	34 40 44 43	86
router_cold_s2	t07	online routing (EMA)	38 40 41 42	84
router_cold_t1	t1	online routing (EMA)	41 39 33 42	84
router_cold_t1_s2	t1	online routing (EMA)	36 42 41 37	74
router_cold_t3	t3	online routing (EMA)	20 25 22 19	38
router_cold_t3_s2	t3	online routing (EMA)	23 22 22 23	46
router_cold_t5	t5	online routing (EMA)	35 34 43 40	80
router_cold_t5_s2	t5	online routing (EMA)	25 34 34 36	72
router_cold_t8	t8	online routing (EMA)	36 41 43 44	88
router_cold_t8_s2	t8	online routing (EMA)	30 40 44 44	88
router_cold_obj0	obj0	online routing (EMA)	45 48 49 49	98
router_cold_obj0_s2	obj0	online routing (EMA)	45 49 48 49	98
router_cold_obj2_s7	obj2	online routing (EMA)	41 44 46 47	94
router_cold_obj2_s8	obj2	online routing (EMA)	45 44 46 47	94
router_cold_obj3_s7	obj3	online routing (EMA)	40 42 45 47	94
router_cold_obj3_s8	obj3	online routing (EMA)	38 39 44 45	90
router_ucb_t07	t07	UCB1	39 34 33 39	78
router_ucb_t07_s2	t07	UCB1	35 35 36 34	68
router_ucb_t1	t1	UCB1	36 42 40 40	80
router_ucb_t3	t3	UCB1	22 21 19 23	46
router_ucb_t5	t5	UCB1	31 31 28 35	70
router_ucb_t8	t8	UCB1	39 32 35 25	50
router_ucb_t8_s2	t8	UCB1	34 33 33 34	68
router_ucb_obj0	obj0	UCB1	47 44 44 49	98
router_ucb_obj2	obj2	UCB1	41 45 42 45	90
router_ucb_obj3	obj3	UCB1	34 32 38 34	68
router_th_t07	t07	Thompson	31 31 44 39	78
router_th_t07_s2	t07	Thompson	37 35 36 43	86

Run tag	Task	Method	Per round / episodes	Score
router_th_t1	t1	Thompson	40 39 39 44	88
router_th_t1_s2	t1	Thompson	41 41 38 39	78
router_th_t3	t3	Thompson	23 20 23 20	40
router_th_t5	t5	Thompson	33 32 37 32	64
router_th_t8	t8	Thompson	40 37 41 43	86
router_th_obj0	obj0	Thompson	49 48 48 49	98
router_th_obj2	obj2	Thompson	43 46 46 44	88
router_th_obj3	obj3	Thompson	30 35 36 42	84
router_linucb_obj3_s7	obj3	AgentNet routing (LinUCB)	33 33 38 34	68
router_linucb_obj3_s8	obj3	AgentNet routing (LinUCB)	32 34 35 36	72
router_linucb_t07_s7	t07	AgentNet routing (LinUCB)	31 37 40 41	82
router_linucb_t07_s8	t07	AgentNet routing (LinUCB)	40 36 38 40	80
router_linucb_obj2_s7	obj2	AgentNet routing (LinUCB)	45 45 44 43	86
router_linucb_obj2_s8	obj2	AgentNet routing (LinUCB)	42 44 43 42	84
router_linucb_obj0_s7	obj0	AgentNet routing (LinUCB)	42 46 48 49	98
router_linucb_t8_s7	t8	AgentNet routing (LinUCB)	33 39 37 40	80
router_linucb_t5_s7	t5	AgentNet routing (LinUCB)	31 29 25 35	70
router_linucb_t1_s7	t1	AgentNet routing (LinUCB)	34 42 38 39	78
router_linucb_t3_s7	t3	AgentNet routing (LinUCB)	23 20 24 17	34
router_th_obj3_s8	obj3	Thompson	32 38 33 38	76
router_th_t8_s2	t8	Thompson	37 36 36 38	76
router_th_obj2_s2	obj2	Thompson	42 44 44 46	92
router_th_t5_s2	t5	Thompson	27 34 31 38	76
router_linucb_t8_s8	t8	AgentNet routing (LinUCB)	34 37 36 39	78
router_linucb_t5_s8	t5	AgentNet routing (LinUCB)	32 33 27 32	64
router_linucb_obj3_a03	obj3	AgentNet routing (LinUCB) with alpha 0.3	33 35 35 34	68
router_linucb_obj3_a3	obj3	AgentNet routing (LinUCB) with alpha 3.0	34 34 34 37	74
router_global_obj3	obj3	online routing without state context	35 32 36 35	70
router_global_t07	t07	online routing without state context	37 37 37 38	76
router_noise_t07_x1	t07	online routing over random-LoRA members	34 33 41 41	82
router_noise_obj3_x1_s8	obj3	online routing over random-LoRA members	34 40 40 43	86
obj3_base_r2	obj3	base re-run in a second process	100ep	75
obj2_base_r2	obj2	base re-run in a second process	100ep	84
t07_base_r3	t07	base re-run in a second process (same GPU type)	100ep	69
t5_base_r3	t5	base re-run in a second process (same GPU type)	100ep	60
router_ema_b03	t07	online routing beta=0.3	37 42 42 45	90
router_ema_b09	t07	online routing beta=0.9	37 42 42 45	90
router_ema_e005	t07	online routing eps=0.05	37 43 43 46	92
router_ema_e03	t07	online routing eps=0.3	37 42 40 44	88
router_ema_warm	t07	online routing warm start	33 36 42 43	86
router_warm_s2fix	t07	online routing warm start	40 42 45 44	88
router_qcold	t07	frozen table (cold-learned)	37	74
router_qcold_s2	t07	frozen table (cold-learned)	38	76
router_qwarm	t07	frozen table (warm-learned)	46	92
router_qwarm_s2	t07	frozen table (warm-learned)	36	72
router_table	t07	frozen audit table	39 41	82
router_gate	t07	learned gate	39 41	82
novel20_base	t07 novel 20 states	base	-	85
router_novel_gate	t07 novel 20 states	learned gate	15 14 16	80
router_novel_ema	t07 novel 20 states	online routing (EMA)	15 17 19 20 20 19 19 18 19 18	90
router_warm_s2	t07 novel 20 states	online routing warm start	15 16 18 19 20 19 18 19 19 19	95
router_noise_obj3_x1	obj3	online routing over 5 untrained random-LoRA members (matched norm)	35 42 41 42	84
router_noise_obj3_x01	obj3	online routing over 5 untrained random-LoRA members (0.1x norm)	34 44 45 42	84
soup_t07_eval	t07	weight averaging (soup)	100ep	55
soup_t5_eval	t5	weight averaging (soup)	100ep	63
mgsel_t07_eval	t07	MG-Select N=4	100ep	65
mgsel8_t07_eval	t07	MG-Select N=8	100ep	72
mgsel_t1_eval	t1	MG-Select N=4	100ep	80
mgsel_t1_r2_eval	t1	MG-Select N=4	100ep	77
mgsel_t3_eval	t3	MG-Select N=4	100ep	50
mgsel_t3_r2_eval	t3	MG-Select N=4	100ep	44
mgsel8_t3_eval	t3	MG-Select N=8	100ep	42
mgsel_t5_eval	t5	MG-Select N=4	100ep	60
mgsel_t8_eval	t8	MG-Select N=4	100ep	72
mgsel_obj0_eval	obj0	MG-Select N=4	100ep	92
mgsel8_obj0_eval	obj0	MG-Select N=8	100ep	93
oftm_sft_t07_eval	t07	OpenVLA-OFT public checkpoint under our stack	100ep	51
oftm_sft_t1_eval	t1	OpenVLA-OFT public checkpoint under our stack	100ep	10

Run tag	Task	Method	Per round / episodes	Score
oftm_sft_t3_eval	t3	OpenVLA-OFT public checkpoint under our stack	100ep	12
oftm_sft_t5_eval	t5	OpenVLA-OFT public checkpoint under our stack	100ep	8
oftm_sft_t8_eval	t8	OpenVLA-OFT public checkpoint under our stack	100ep	22

B MEMBER TRAINING DETAILS

All DPO members share the collection and training settings of Section 4. The planner runs 8 simulations to depth 3 at each call, candidates within L2 distance 0.02 are deduplicated, and pairs from successful episodes receive weight 1.0. Member c of t07 is trained with the verifiable-reward objective for a fixed number of steps; see the released configuration. Table 4 lists the per-task settings.

Table 4: Member training and scores. All DPO members use LoRA rank 16, $\alpha = 16$, learning rate 5×10^{-6}, $\beta = 0.1$, one epoch, effective batch size 8, and drop pairs whose search-value gap is below 0.005. Pairs are collected from 100 episodes per task. $\|\Delta W\|_F$ is the per-layer Frobenius norm of the adapter update averaged over the 439 adapted layers. Scores are successes per 100 episodes for each member evaluated alone.

Task	Members	Pairs	Seeds / variants	Mean $\|\Delta W\|_F$	Scores
obj0	s7, s8, s9	949	seeds 7, 8, 9	0.0033 to 0.0041	93, 90, 90
obj2	s7 to s11	1428	seeds 7 to 11	0.0054 to 0.0060	84, 90, 88, 86, 87
obj3	s7 to s11	1384	seeds 7 to 11	0.0047 to 0.0050	69, 68, 62, 64, 72
t1	s7 to s11	3257	seeds 7 to 11	0.0026 to 0.0027	78, 76, 79, 77, 77
t3	s7 to s11	2934	seeds 7 to 11	0.0046 to 0.0047	not audited
t5	s7 to s11	3048	seeds 7 to 11	0.0052 to 0.0059	64, 69 (s10, s11)
t8	s7 to s11	3266	seeds 7 to 11	0.0058 to 0.0063	71, 59 (s10, s11)
t07	a	3027	DPO, pair variant 1, seed 7	0.0069	63
t07	b	3540	DPO, pair variant 2, seed 8	0.0205	59
t07	c	11939	RL with verifiable reward, seed 8	0.1085	70

C PROSPECTIVE PREDICTIONS

The predictions for obj3 and obj2 were written in the project log on September 1, 2026, after the base-policy screening and before any member was trained (member training started on September 3). The entry reads: both new tasks have $\sigma \leq 1$, so online routing should approach the per-state oracle of the portfolio; if the members cover at least half of the base-dead states, the last round should reach at least 85 on obj3 and at least 90 on obj2; if the members do not cover those states, routing will stay at the base level. Table 5 lists the screening scores, the predictions, and the outcomes.

Table 5: Prospective validation on two `libero_object` tasks. Predictions were written down before any routing run. Two values are two seeds.

Task	Screening	Prediction	Members	UCB1	Thompson	AgentNet routing	Ours
obj3	75	≥ 85	62 to 72	68	84 / 76	68 / 72	**94 / 90**
obj2	84	≥ 90	84 to 90	90	88 / 92	86 / 84	**94 / 94**

D SINGLE-MODEL TEST-TIME SAMPLING

Table 6 reports MG-Select on six tasks. It helps only on the high-σ task t3, where per-state routing gives no consistent gain, so the two methods work in complementary settings.

Table 6: MG-Select with verifier-free best-of-N selection. It helps only on the high-σ task t3. A dash means not measured.

	obj0	t07	t5	t8	t1	t3
Base	96	71	64	70	76	37
MG-Select $N = 4$	92	65	60	72	80 / 77	50 / 44
MG-Select $N = 8$	93	72	–	–	–	42

E Audit Counts and Cross-Process Agreement

Table 7 lists the two-round audit counts of the base policy on all eight tasks and the cross-process agreement of its per-state labels; Sections 4 and 6 discuss them.

Table 7: Two-round audit of the base policy and cross-process repeatability. Solid, split, and dead count states with two, one, or zero successful episodes. IID split is the expected split count under independent trials. κ measures agreement between labels of the same checkpoint in separate processes. For t07 and t5, the first value is a second process on the same GPU type and the others use a different GPU type. A dash means that no second process was run.

Task	Base	Solid	Split (σ)	Dead	IID split	Cross-process κ
obj0	96	48	0	2	3.8	-0.05
obj2	84	42	0	8	13.4	1.00
obj3	75	37	1	12	18.8	1.00
t5	64	31	2	17	23.0	0.14, 0.14
t8	70	33	4	13	21.0	–
t07	71	33	5	12	20.6	0.76, 0.27, 0.25
t1	76	30	16	4	18.2	0.11
t3	37	9	19	22	23.3	–

F Novel Initial States

Table 8 reports results on 20 additional initial states of t07 that were not among the 50 audited states.

Table 8: Results on 20 additional initial states of t07 outside the 50 audited states. Online routing runs ten rounds from a uniform (cold) or audit-prior (warm) table; the learned gate does not update and runs three rounds.

Method	Last round
Base	85
Learned gate	80
Online routing, cold	90
Online routing, warm	**95**