AdaMoE-VLA: ADAPTIVE MIXTURE OF ACTION EXPERTS FOR VISION-LANGUAGE-ACTION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models show strong potential as generalist robot policies, yet enhancing their action-generation capability remains challenging due to the diversity of robot behaviors and the need for fine-grained action modeling. While Mixture-of-Experts (MoE) offers an efficient way to increase model capacity through sparse activation, its role in VLA is not simply generic model scaling. Long-horizon manipulation trajectories often contain heterogeneous and frequently switching action modes, such as reaching, grasping, aligning, inserting, and recovery behaviors, making multi-expert action modeling especially suitable for VLA policies. However, directly applying conventional MoE to VLA action heads introduces an optimization conflict: the router is required to both select experts under a load-balancing objective and provide precise state-dependent expert weighting for continuous action generation. This coupling can interfere with fine-grained action learning, particularly when behavior modes change frequently. Therefore, we propose **AdaMoE-VLA**, an adaptive mixture of action experts for VLA learning. **AdaMoE-VLA** decouples expert selection from expert weighting by preserving the standard router for trainable sparse selection while introducing a lightweight *Scale Adapter* for task-conditioned expert reweighting. This design mitigates gradient interference between load balancing and action prediction, enabling task-adaptive expert reweighting while preserving real-time inference. Across simulation and real-world benchmarks, **AdaMoE-VLA** consistently improves π-series VLA baselines, and its gains become more pronounced as action-mode transitions become more frequent. It reaches **98.3%** on the LIBERO benchmark, while delivering larger improvements in more behaviorally diverse settings, with a **9.3%** gain on RoboTwin 2.0 and a **20% – 21.5%** absolute gain in real-world experiments.

1 INTRODUCTION

Vision-Language-Action (VLA) models have recently emerged as a promising paradigm for generalist robot policies (Team et al., 2024; Kim et al., 2024; Black et al., 2024; Zhao et al., 2025; Wen et al., 2025; Bjorck et al., 2025; Zhou et al., 2025b; Pertsch et al., 2025; Li et al., 2025; Intelligence et al., 2025). Yet their effectiveness hinges on temporally precise, high-frequency, and physically grounded action generation. In long-horizon manipulation, especially with human-collected demonstrations and diverse task families, the action distribution is rarely unimodal (Mandlekar et al., 2021; Shafiullah et al., 2022; Jia et al., 2024). A single trajectory may involve reaching, grasping, aligning, inserting, recovery behaviors, and frequent transitions between these stages. Modeling such heterogeneous and frequently switching action modes places a strong representational burden on a single dense action expert. Direct dense expansion, however, is often computationally expensive and data-inefficient given the limited size of robot datasets (Liu et al., 2023; Mu et al., 2025).

This motivates sparse multi-expert action modeling. Mixture-of-Experts (MoE) provides a natural mechanism for expanding the representational capacity of the action expert while keeping the activated computation small (Shazeer et al., 2017; Lepikhin et al., 2020; Dai et al., 2024). Sparse experts can provide reusable action bases for different behavior modes, and applying MoE to the action expert is particularly suitable because this module receives direct supervision from the action prediction or flow-matching objective. This motivation is supported by the action-mode structure observed across LIBERO (Liu et al., 2023), RoboTwin (Chen et al., 2025a), and real-world demonstrations. As shown in Figure 1, DBI-optimal clustering (Davies & Bouldin, 1979) yields 12, 20, and 21 action modes on

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.

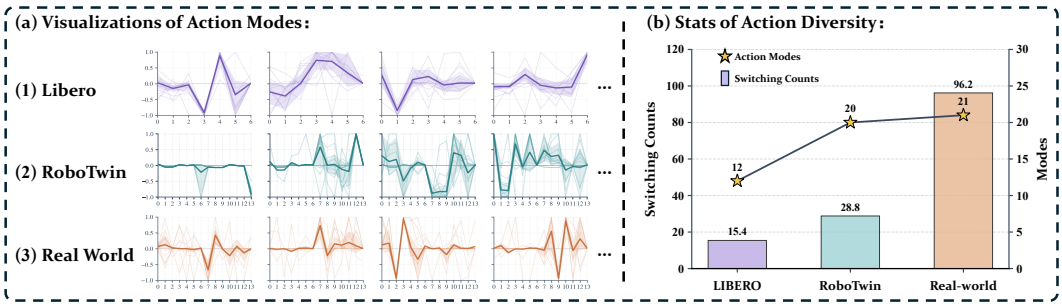


Figure 1: **Action-mode analysis across LIBERO, RoboTwin, and real-world data.** (a) Representative action modes extracted from trajectories. (b) The number of action modes and average switching counts per episode both increase from LIBERO to RoboTwin and real-world demonstrations.

LIBERO, RoboTwin, and real-world data, respectively, while the average number of action-mode switches per episode increases from 15.4 to 28.8 and 96.2. These statistics suggest that realistic VLA deployment involves both diverse skills and frequent temporal transitions, which is precisely the regime where multi-expert action modeling becomes useful.

However, directly applying standard MoE to VLA action modeling introduces an optimization conflict. In conventional MoE layers, the same router scores determine both *which* experts are selected and *how much* each selected expert contributes to the final output. MoE training typically relies on a load-balancing auxiliary loss to encourage uniform expert usage (Fedus et al., 2022; Wang et al., 2024). This design is effective for large-scale language or vision-language pretraining, but it can be undesirable for continuous robot action generation: the load-balancing objective encourages balanced expert utilization, while the flow-matching objective may require highly state-dependent and non-uniform expert contributions at each timestep. Since both objectives update the same router scores, the weighting signal used for continuous action prediction can be perturbed by the balancing objective, especially when action modes switch frequently.

In this paper, we propose **AdaMoE-VLA**, an adaptive mixture of action experts for VLA learning. **AdaMoE-VLA** decouples expert selection from expert weighting through a lightweight Scale Adapter. The router is preserved for trainable top-k expert selection and load balancing, while the *Scale Adapter* provides an additional task-conditioned residual weight for selected experts. This design separates balanced expert selection from flexible expert contribution, thereby reducing the interference between the auxiliary load-balancing objective and the main action-learning objective. As a result, **AdaMoE-VLA** preserves the efficiency of sparse activation while allowing more flexible expert contributions for fine-grained robot control.

We evaluate **AdaMoE-VLA** across simulation benchmarks and real-world manipulation tasks. On LIBERO, **AdaMoE-VLA** consistently improves both π_0 and the stronger $\pi_{0.5}$ baselines, reaching 98.3% average success rate with the $\pi_{0.5}$ base. On RoboTwin 2.0, where the dataset is larger and action behaviors are more diverse, **AdaMoE-VLA** achieves a more substantial improvement of 9.3% over the π_0 baseline and also outperforms Vanilla MoE. In real-world ALOHA (Zhao et al., 2023) experiments, **AdaMoE-VLA** further yields large gains on both short-horizon and long-horizon manipulation tasks, validating its practical effectiveness under physical robot constraints. Beyond accuracy, our parallelized sparse implementation maintains real-time action-expert inference, making the method suitable for high-frequency control settings.

Our contributions are summarized as follows:

- We identify that long-horizon VLA action modeling faces strong representational pressure from heterogeneous and frequently switching action modes, and motivate sparse MoE action experts as a natural way to model such multi-modal action distributions.
- We reveal and address an optimization conflict in vanilla MoE action heads: the same router is required to support both load-balanced expert selection and precise state-dependent expert weighting for continuous action generation. **AdaMoE-VLA** decouples these two roles through a lightweight *Scale Adapter*, enabling task-driven weighting with balanced selection.
- We validate **AdaMoE-VLA** across simulation and real-world benchmarks, achieving 98.3% on LIBERO, a 9.3% gain on RoboTwin 2.0, and a 20.0%–21.5% absolute gain in real-world experiments, with particular benefits in behaviorally diverse and high mode-switching regimes.

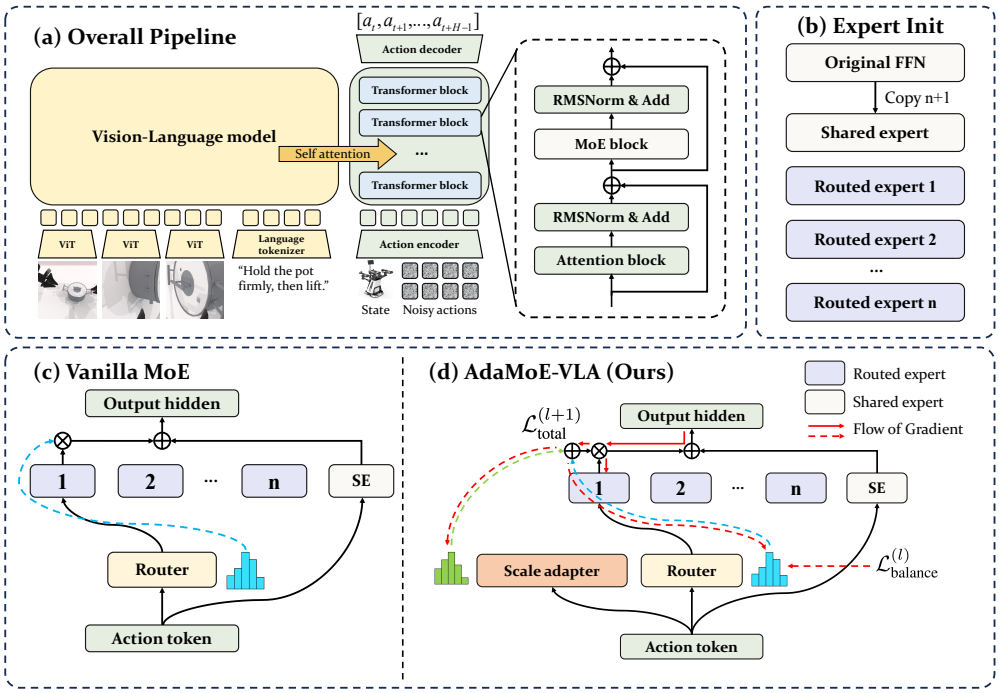


Figure 2: **AdaMoE-VLA architecture overview. (a) Overall Pipeline:** Multi-modal input processing through the VLM backbone and action expert. **(b) Expert Initialization:** The shared expert inherits the original FFN weights, while routed experts are initialized from copied FFN weights. **(c) Vanilla MoE:** A single router couples expert selection and weighting through top-k selection and softmax outputs. **(d) AdaMoE-VLA (Ours):** The router determines expert selection, while an independent *Scale Adapter* provides task-conditioned residual weighting for selected experts. This separates the gradient path for task-driven expert weighting from the local load-balancing constraint on the router.

2 METHOD

2.1 PRELIMINARIES

We build on the π-series VLA policies, including π_0 and $\pi_{0.5}$, which adopt flow-matching based action generation, and convert their action expert into a sparse mixture-of-experts module. The vision-language backbone and the policy input-output formulation remain unchanged. At each timestep t, the model combines observations $\mathbf{O}_t$ consisting of multi-view RGB images, a language instruction and the robot state, and predicts an action chunk $\mathbf{A}_t$ for high-frequency control.

Formally, the robot control problem is formulated as learning a policy that maps observations to action sequences. Following the π_0 framework, we aim to model the conditional distribution $\pi(\mathbf{A}_t \mid \mathbf{O}_t)$, where $\mathbf{A}_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}]$ denotes a chunk of H future actions, and $\mathbf{O}_t = [\mathbf{I}_1^t, \ldots, \mathbf{I}_n^t, \ell_t, \mathbf{q}_t]$ is the observation, consisting of images $\mathbf{I}_i^t$ from multiple camera views, a natural language instruction ℓ_t, and the robot's proprioceptive state $\mathbf{q}_t$ (joint angles and gripper state).

The action chunk is modeled with flow matching, following the π series. The flow matching loss is:

$$\mathcal{L}_\tau(\theta) = \mathbb{E}\Big[\|\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{O}_t) - \mathbf{u}(\mathbf{A}_t^\tau \mid \mathbf{A}_t)\|_2^2\Big] \tag{1}$$

where $\mathbf{A}_t^\tau = (1-\tau)\mathbf{A}_t + \tau\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$, and $\mathbf{u}(\mathbf{A}_t^\tau \mid \mathbf{A}_t) = \boldsymbol{\epsilon} - \mathbf{A}_t$.

During inference, we start from pure noise $\mathbf{A}_t^1 \sim \mathcal{N}(0, I)$ and partition the time interval into N equal steps with $d\tau = 1/N$. The denoising process iteratively applies:

$$\mathbf{A}_t^{\tau - d\tau} = \mathbf{A}_t^\tau - d\tau \cdot \mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{O}_t) \tag{2}$$

where $\mathbf{A}_t^\tau$ represents the noisy action at timestep t and flow time τ, and $\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{O}_t)$ is the learned velocity field that predicts the denoising direction to obtain the final action prediction $\mathbf{A}_t^0$.

2.2 MoE Action Modeling for Flow-Matching Policy

Building upon the π series, we introduce a MoE architecture specifically within the action expert, as shown in Figure 2. Specifically, our MoE action expert consists of two types of experts: (1) **Shared experts** that process common action patterns across all tasks and capture universal manipulation knowledge, and (2) **Routed experts** that provide additional action-specific capacity through sparse activation. The gating function $R(\cdot)$ routes action tokens to appropriate routed experts based on the input features, while shared experts are always activated to maintain consistent baseline performance.

Formally, for each action token x in the action sequence, the output of our MoE action expert is computed as:

$$F_{\text{MoE}}(x) = F_{\text{shared}}(x) + \sum_{i \in \text{top-}k} r_i(x) \cdot F_i(x) \tag{3}$$

where $F_{\text{shared}}(\cdot)$ represents the shared expert processing, $F_i(\cdot)$ denotes the i-th routed expert, $r_i(x)$ is the full-softmax router weight for expert i before top-k masking, and K is the total number of routed experts. The gating network employs a top-k selection strategy where only the top-k experts with highest gating scores are activated for each token, ensuring computational efficiency while maintaining the model's expressive capacity.

To prevent expert collapse and ensure uniform expert utilization, we incorporate the auxiliary load balancing loss $\mathcal{L}_{\text{balance}}$ (Fedus et al., 2022):

$$\mathcal{L}_{\text{balance}} = K \sum_{i=1}^{K} f_i P_i \tag{4}$$

where f_i is the fraction of tokens selecting expert i, and P_i is the average gating probability for expert i. This loss encourages both balanced selection frequency and gating probabilities. The total training objective combines the flow matching loss $\mathcal{L}_\tau$ with this auxiliary constraint:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_\tau + \lambda_{\text{balance}} \mathcal{L}_{\text{balance}} \tag{5}$$

where λ_{balance} controls the weight of the regularization.

2.3 Decoupled Expert Selection and Weighting

In standard MoE, the same router scores determine both the selected experts and their aggregation weights. This coupling is undesirable for VLA action generation: the load-balancing objective encourages uniform expert usage, while the flow-matching objective may require highly state-dependent and non-uniform expert contributions. Since both objectives update the same router scores, the weighting signal used for continuous action prediction can be perturbed by the balancing objective, especially when action modes switch frequently. This motivates separating expert selection, which should remain balanced, from expert weighting, which should remain task-adaptive.

We introduce a lightweight *Scale Adapter* $S(\cdot)$ to decouple these two roles. The router $R(\cdot)$ determines the top-k selected experts, while the *Scale Adapter* provides an additional task-conditioned residual weight for the selected experts. Formally, the **AdaMoE-VLA** action expert is computed as:

$$F_{\text{MoE}}(x) = F_{\text{shared}}(x) + \sum_{i \in \text{top-}k} \Big[S_i(x) + \text{softmax}(R(x))_i \Big] \cdot F_i(x) \tag{6}$$

where $S_i(x)$ is a task-conditioned residual term added to the selected router weight. The selected expert set is still determined by the router, so the load-balancing loss supervises expert selection, while the adapter adjusts the contribution magnitude of selected experts. When $k = 1$, the adapter can still modulate the routed branch relative to the always-active shared expert.

Gradient Decoupling. For layer l, let $\mathcal{L}_{\text{balance}}^{(l)}$ be the local load-balancing loss applied to the router, and let $\mathcal{L}_{\text{total}}^{(l+1)}$ denote the objective backpropagated from upper layers. Since $\mathcal{L}_{\text{balance}}^{(l)}$ depends only on the router probabilities, the gradients for *Scale Adapter* parameters θ_{S} and router parameters θ_{R} are:

$$\begin{aligned} \nabla_{\theta_S} \Big(\mathcal{L}_{\text{total}}^{(l+1)} + \lambda \mathcal{L}_{\text{balance}}^{(l)} \Big) &= \nabla_{\theta_S} \mathcal{L}_{\text{total}}^{(l+1)} \\ \nabla_{\theta_R} \Big(\mathcal{L}_{\text{total}}^{(l+1)} + \lambda \mathcal{L}_{\text{balance}}^{(l)} \Big) &= \nabla_{\theta_R} \mathcal{L}_{\text{total}}^{(l+1)} + \lambda \nabla_{\theta_R} \mathcal{L}_{\text{balance}}^{(l)} \end{aligned} \tag{7}$$



Figure 3: Empirical verification of gradient conflict mitigation.

Thus, the router remains regularized for balanced expert selection, whereas the *Scale Adapter* is optimized only through the task-driven downstream objective. This allows expert weighting to remain flexible without removing the load-balancing constraint from expert selection.

Empirical Gradient Verification. Figure 3 reports the gradient alignment angle during training. The router gradient shows larger deviation from the flow-matching gradient, while the combined router-adapter update remains better aligned, suggesting that the *Scale Adapter* reduces the interference introduced by load balancing. Together with expert-utilization statistics, these training curves can also guide the selection of λ_{balance}, favoring the smallest coefficient that prevents persistent routing concentration while maintaining a low and stable combined–FM gradient angle. More details are provided in Appendix H.

3 Experiment

3.1 Evaluation Settings

We select two simulation benchmarks to evaluate our method: (1) **LIBERO**: Four task suites including Spatial, Object, Goal, and Long (Liu et al., 2023). (2) **RoboTwin 2.0**: 19 diverse manipulation tasks. Each task dataset contains 100 expert trajectories from Clean environments and 400 from Domain Randomized environments (Chen et al., 2025a). To ensure robust statistical reliability, we conduct a comprehensive evaluation with 50 independent rollout episodes for each task, resulting in a total of 2,000 episodes for LIBERO and 950 episodes for RoboTwin 2.0.

3.2 Simulation Results

In this section, we present the main performance results of **AdaMoE-VLA** on simulation benchmarks.

Performance on LIBERO. As shown in Table 1, we evaluate **AdaMoE-VLA** on top of both π_0 and $\pi_{0.5}$ baselines. With the π_0 base, **AdaMoE-VLA** improves the average success rate from 94.2% to 96.1%, with a clear gain on the long-horizon suite (85.2% $\rightarrow$ 90.8%). With the stronger $\pi_{0.5}$ base, **AdaMoE-VLA** further improves the average success rate from 96.9% to 98.3%, achieving the best overall performance in Table 1. These results show that **AdaMoE-VLA** is compatible with both base policies and enhances overall action modeling under the same benchmark setting.

Performance on RoboTwin 2.0. The improvements are even more significant on the large-scale RoboTwin 2.0 dataset. As detailed in Table 2, we observe a substantial 9.3% performance gain (40.4% $\rightarrow$ 49.7%) across 19 manipulation tasks.

Table 1: Performance Comparison on LIBERO (Success Rate %).

Method	Spatial	Object	Goal	Long	Average
Diffusion Policy	78.5	87.5	73.5	64.8	76.1
OpenVLA	84.7	88.4	79.2	53.7	76.5
SpatialVLA	88.2	89.9	78.6	55.5	78.1
CoT-VLA	87.5	91.6	87.6	69.0	83.9
π_0-Fast	96.4	96.8	88.6	60.2	85.5
π_0	96.8	98.8	95.8	85.2	94.2
AdaMoE-VLA (π_0 base)	**99.4**	98.0	96.0	90.8	96.1
$\pi_{0.5}$	98.8	98.2	98.0	92.4	96.9
AdaMoE-VLA ($\pi_{0.5}$ base)	99.0	**99.8**	**98.2**	**96.2**	**98.3**

Table 2: Task Success Rates Comparison in RoboTwin 2.0 Domain Randomized Environments.

Task	π_0	Vanilla	Ours	Task	π_0	Vanilla	Ours	Task	π_0	Vanilla	Ours
Beat Block Hammer	88%	72%	86%	Place Can Basket	36%	50%	48%	Stack Blocks Two	58%	76%	66%
Click Bell	38%	32%	54%	Pick Dual Bottles	26%	28%	40%	Stack Bowls Three	68%	74%	80%
Click Alarmclock	24%	28%	44%	Place Cans Plasticbox	32%	44%	40%	Turn Switch	34%	28%	42%
Handover Block	24%	36%	26%	Place Object Stand	48%	62%	64%	Pick Diverse Bottles	20%	42%	34%
Move Can Pot	6%	2%	10%	Place A2B Left	26%	38%	40%	Place Dual Shoes	54%	68%	72%
Move Playingcard Away	66%	62%	68%	Place A2B Right	30%	36%	32%	**Average**	**40.4%**	**45.9%**	**49.7%**
Place Phone Stand	48%	48%	50%	Put Bottles Dustbin	42%	46%	48%				

3.3 Real-World Experiments

To validate the practical effectiveness of our approach, we conduct real-world experiments using the ALOHA platform (Zhao et al., 2023). We evaluate both short-horizon manipulation tasks and the long-horizon sequential task. For short-horizon evaluation, we fine-tune the policy on 450 real-world demonstrations across four tasks: *Place Cup*, *Stack Plate*, *Click Bell*, and *Adjust Bottle*, and conduct 50 trials for each task. For long-horizon evaluation, the robot is required to sequentially complete three stages: *open the box*, *take out and place the bag*, and *fold the box*. We evaluate each stage with 10 trials. Detailed experimental setups and object configurations are provided in the Appendix A.

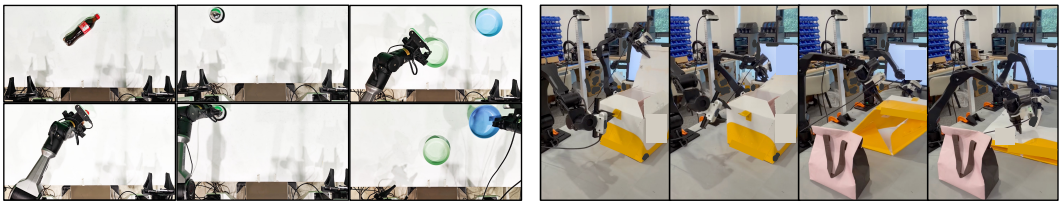


Figure 4: Qualitative results of real-world robot manipulation demonstrations. The snapshots show sequential execution processes across different real-world tasks on the Piper dual-arm (left) and the IMETA Y1 dual-arm (right).

As shown in Figure 5, **AdaMoE-VLA** achieves an average success rate of 71.5% on short-horizon tasks, outperforming the π_0 baseline by +21.5%. On the long-horizon sequential task, $\pi_{0.5}$ + **AdaMoE-VLA** consistently improves over the $\pi_{0.5}$ baseline across all three stages, increasing the average success rate from 33.3% to 53.3% (+20.0%). These results suggest that the proposed action-expert MoE design remains effective in real-world settings, especially when the task requires transitions across multiple manipulation stages.

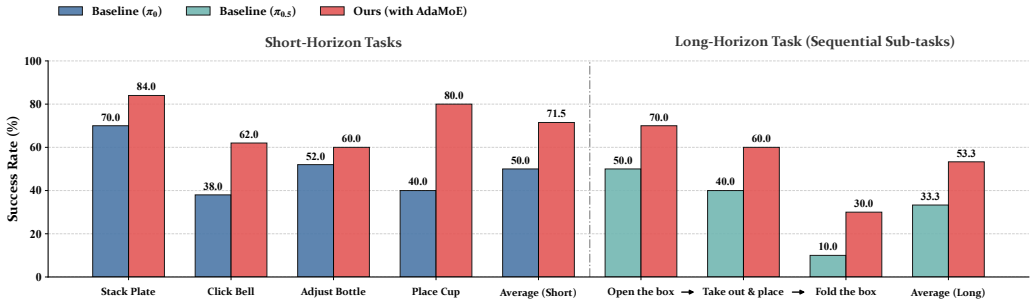


Figure 5: Real-world success rates on short-horizon and long-horizon manipulation tasks.

3.4 Ablation Studies and Analysis

3.4.1 Effectiveness of Decoupled Architecture

To validate our decoupling mechanism, we compare three architectural variants on LIBERO configured with 4 experts and top-1 selection (illustrated in Figure 6):

(a) **Vanilla MoE**: The standard coupled architecture where the router's softmax probability directly determines both expert selection and the final aggregation weight.

(b) **CSMoE (Concatenated)**: A variant where the *Scale Adapter* predicts weights by conditioning on both the input token and the router's output (via concatenation), attempting to decouple weighting from selection through an auxiliary projection.

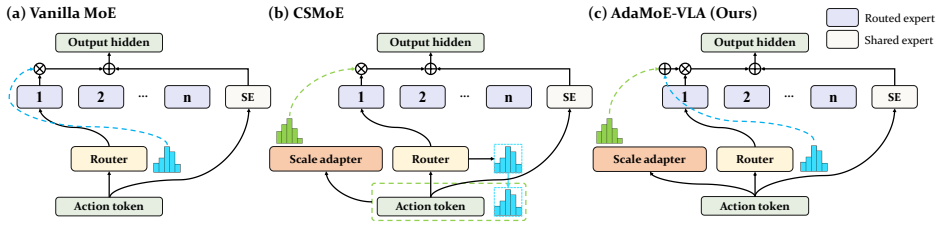


Figure 6: Architecture Variants for Decoupling Expert Selection and Weighting.

(c) **AdaMoE-VLA (Ours)**: Our proposed design where the *Scale Adapter* learns a residual term added to the selected router weight ($w = r + S$).

As shown in Table 3, our **AdaMoE-VLA** achieves the best overall performance across LIBERO task suites. The concatenated approach (CSMoE) demonstrates moderate improvements over the Vanilla MoE, effectively validating the importance of decoupling expert selection from weighting. Furthermore, our additive design proves to be the most effective mechanism by achieving a more thorough decoupling.

To verify the generalization of our approach, we also extended the comparison between Vanilla MoE and **AdaMoE-VLA** to the larger-scale RoboTwin 2.0 benchmark. As summarized in Table 2, **AdaMoE-VLA** consistently outperforms Vanilla MoE, achieving a mean success rate of 49.7% compared to 45.9%. This meaningful margin (+3.8%) further highlights the effectiveness of the decoupled architecture in handling diverse manipulation tasks.

3.4.2 Hyper-parameter Sensitivity

To understand the impact of key design choices and hyper-parameters in our **AdaMoE-VLA** architecture, we conduct comprehensive ablation studies on the LIBERO benchmark. We systematically vary different components and hyper-parameters to identify optimal configurations and understand their influence on manipulation performance.

We analyze 3 critical MoE-specific hyper-parameters that may affect model performance:

Top-k Selection and Number of Experts. The top-k selection, number of experts, and load balancing loss are highly coupled hyper-parameters in MoE architectures. Due to limited GPU resources, we cannot exhaustively search all combinations of expert numbers and top-k values with varying load balancing coefficients. We present ablation results under the setting $\lambda_{\text{balance}} = 0.01$ in Table 3.

Under this setting, we find that the configuration with 8 experts and top-2 selection achieves optimal performance (96.1%), while 4 experts with top-1 selection also achieves comparable performance

Table 3: Ablation summary on LIBERO (Success Rate %).

Ablation	Variant	#Exp.	Top-k	λ_{balance}	Spatial	Object	Goal	Long	Avg.
Router design ablation									
	π_0	–	–	–	96.8	**98.8**	95.8	85.2	94.2
	Vanilla MoE	4	1	0.01	98.6	97.0	96.0	87.4	94.8
	CSMoE	4	1	0.01	99.2	97.4	95.4	90.0	95.5
	AdaMoE-VLA	4	1	0.01	**99.6**	95.0	**97.2**	**92.0**	**96.0**
Number of experts and top-k ablation									
	AdaMoE-VLA	4	1	0.01	**99.6**	95.0	**97.2**	**92.0**	96.0
	AdaMoE-VLA	4	2	0.01	98.2	96.4	96.0	90.8	95.4
	AdaMoE-VLA	8	1	0.01	98.3	95.9	96.4	91.7	95.6
	AdaMoE-VLA	8	2	0.01	99.4	**98.0**	96.0	90.8	**96.1**
	AdaMoE-VLA	12	1	0.01	98.2	96.8	94.4	89.2	94.7
	AdaMoE-VLA	12	2	0.01	98.0	97.2	95.6	90.4	95.3
	AdaMoE-VLA	16	1	0.01	99.4	96.6	94.2	88.6	94.7
	AdaMoE-VLA	16	2	0.01	98.6	94.6	96.0	89.6	94.7
Load balancing weight ablation									
	AdaMoE-VLA	4	1	0.001	98.0	**96.0**	96.0	88.0	94.5
	AdaMoE-VLA	4	1	0.01	**99.6**	95.0	**97.2**	**92.0**	**96.0**
	AdaMoE-VLA	4	1	0.05	97.8	95.2	96.4	91.0	95.1

(96.0%). However, when increasing the number of experts to 12 or 16, performance degradation occurs. We attribute this to the relationship between optimal model capacity and available data, suggesting that the modest size of LIBERO (273K frames) limits the effective utilization of a larger expert pool. To validate this hypothesis, we conduct additional experiments on the RoboTwin dataset (see Appendix D), which contains 2.03M training frames, approximately 8 times the size of LIBERO. On RoboTwin 2.0, the optimal number of experts increases to 12. This indicates that the appropriate model size scales with the quantity of training data, and that larger, more diverse datasets are required to support a higher number of experts.

Load Balancing Loss Weight. MoE performance shows high sensitivity to the load balancing coefficient λ_{balance}. As shown in Table 3, the optimal setting ($\lambda_{\text{balance}} = 0.01$) achieves 96.0% average performance, while both insufficient balancing ($\lambda_{\text{balance}} = 0.001$, 94.5%) and excessive penalization ($\lambda_{\text{balance}} = 0.05$, 95.1%) degrade performance. The Long task suite is particularly affected, dropping from 92.0% to 88.0% with inadequate load balancing, highlighting the importance of proper expert utilization in sequential manipulation tasks.

3.5 Computational Efficiency Analysis

We evaluate the computational efficiency of our **AdaMoE-VLA** (N experts + Shared) against parameter-matched dense baselines ($N + 1$ experts). To assess performance under high-load conditions, we conduct experiments using a batch size of 32 and a sequence length of 51, simulating a standard high-throughput inference setting.

Parallelized Implementation. We implement a two-level parallelization strategy utilizing JAX's `vmap` primitive. Unlike dense models that apply uniform computation to all tokens across the batch, our architecture processes shared and routed experts via independent parallel branches. For the routed experts, we dynamically sort and reshape tokens from the batch dimension into expert-specific groups. This approach fully saturates GPU compute capacity by enabling the simultaneous execution of all experts, thereby minimizing the routing overhead during batched inference.

FLOPs Reduction. As shown in Table 4, our method maintains nearly constant inference FLOPs regardless of the number of experts, whereas dense baselines scale linearly. At the 32-expert scale, our approach achieves a **62.6% reduction in Inference FLOPs** compared to the dense baseline, demonstrating superior scalability.

Inference Latency. Table 4 highlights the latency advantage in batched settings. Specifically, we report the wall-clock time for a single inference step of the Action Expert. While dense models degrade to 129.5ms at 32 experts due to full parameter activation, our method maintains 18.28ms. This **7.1× speedup** confirms that our parallelized architecture effectively circumvents the computational bottlenecks typical of scaling dense models.

Table 4: Computational efficiency comparison. All configurations utilize N experts plus a shared expert. Inference latency are measured with batch size 32 and sequence length 51.

Experts (N)	Params (B)	Inf. FLOPs (T)			Train FLOPs (T)			Latency (ms)		Speedup	FPS (Ours)
		Dense	Ours	Red.	Dense	Ours	Red.	Dense	Ours		
4	3.84	2.14	1.82	14.8%	3.24	3.14	2.9%	21.62	13.02	1.7×	76.8
8	4.45	2.53	1.82	28.0%	3.36	3.14	6.3%	35.01	14.08	2.5×	71.0
12	5.05	2.92	1.82	37.6%	3.47	3.14	9.5%	51.15	15.69	3.3×	63.7
16	5.66	3.31	1.82	45.0%	3.59	3.14	12.5%	69.97	16.55	4.2×	60.4
32	8.07	4.88	1.82	62.6%	4.06	3.14	22.6%	129.46	18.28	7.1×	54.7

4 Related Works

4.1 Vision–Language–Action Models

Vision–Language–Action (VLA) models integrate visual perception, language understanding, and action generation for generalist robot control, often by adapting web-scale vision-language backbones to robot datasets and benchmarks (Brohan et al., 2023b;a; Liu et al., 2023; Kim et al., 2024; Black et al., 2024; Pertsch et al., 2025; Kim et al., 2025; Intelligence et al., 2025; Zhao et al., 2025;

Chen et al., 2025a). Existing methods commonly adopt either auto-regressive action decoding or diffusion/flow-matching based continuous action generation. Auto-regressive policies inherit the scaling behavior of language models but incur horizon-dependent inference costs (Brohan et al., 2023b;a; Kim et al., 2024; Pertsch et al., 2025), while diffusion/flow-matching policies generate continuous action chunks in parallel and are more suitable for real-time control (Lipman et al., 2023; Black et al., 2024; Liu et al., 2025; Wen et al., 2025; Shukor et al., 2025; Qu et al., 2025). However, real-time control constraints make dense scaling of the whole VLA backbone less practical during inference, making the action-generation module a more targeted scaling point. How to efficiently improve the capacity of this module itself remains underexplored. Our work addresses this problem by scaling the action expert of flow-matching VLA policies while preserving low-latency inference.

4.2 Action Distribution and Mode Switching in Robot Demonstrations

Robot demonstrations often contain heterogeneous behaviors, multimodal action distributions, and temporal transitions between manipulation stages. Prior studies show that robot learning is strongly affected by data diversity, embodiment heterogeneity, demonstration quality, operator strategy, and skill coverage (Dasari et al., 2020; Mandlekar et al., 2021; Walke et al., 2023; Fang et al., 2023; O'Neill et al., 2024; Khazatsky et al., 2024). Other works further analyze multimodal action distributions, behavior diversity, primitive segmentation, and temporal skill structures (Shafiullah et al., 2022; Jia et al., 2024; von Hartz et al., 2024; Chen et al., 2025b). Different from these studies, which mainly use action statistics for dataset analysis, benchmarking, or trajectory segmentation, we connect action-mode diversity directly to VLA architecture design. Our analysis across LIBERO, RoboTwin, and real-world demonstrations links more diverse and frequently switching action patterns to the capacity bottleneck of a single dense action expert, motivating action-expert scaling driven by action-mode structure.

4.3 Mixture-of-Experts for Efficient Action Modeling

Sparse Mixture-of-Experts (MoE) architectures improve model capacity through conditional computation by activating only a subset of experts for each token (Shazeer et al., 2017; Lepikhin et al., 2020; Riquelme et al., 2021; Fedus et al., 2022; Du et al., 2022). MoE has been widely used for efficient large-model scaling, dense-to-sparse conversion, and load-balancing optimization (Komatsuzaki et al., 2023; Dai et al., 2024; DeepSeek-AI et al., 2024; Jiang et al., 2024; Wang et al., 2024), and has also been explored in robotics and VLA models (Huang et al., 2025; Yu et al., 2025; Wang et al., 2025; Zhou et al., 2025a). However, these works mostly treat MoE as a general-purpose scaling or routing mechanism, whereas VLA action generation requires precise continuous control under strict latency constraints and frequent action-mode transitions. Different from prior methods, our work is motivated by the action-mode diversity and switching structure of robot demonstrations, and studies MoE as an action-expert scaling mechanism rather than a generic backbone expansion. More importantly, we identify the conflict between load-balanced expert selection and state-dependent expert weighting in continuous action generation, which motivates a VLA-specific decoupled MoE design.

5 Conclusion

We presented **AdaMoE-VLA**, a decoupled mixture-of-action-experts architecture for VLA learning. Motivated by the heterogeneous and frequently switching action modes in long-horizon manipulation, **AdaMoE-VLA** applies sparse MoE to the action expert while decoupling expert selection from expert weighting. The router is preserved for trainable top-k selection and load balancing, while a lightweight Scale Adapter provides task-conditioned residual weighting for selected experts. This separation reduces the interference between the load-balancing objective and the flow-matching action objective, enabling more flexible expert contribution under efficient sparse inference. Experiments across LIBERO, RoboTwin 2.0, and real-world manipulation tasks validate the effectiveness of this design. **AdaMoE-VLA** improves both π_0 and $\pi_{0.5}$ on LIBERO, reaching 98.3% average success rate with the $\pi_{0.5}$ base, achieves a 9.3% gain on RoboTwin 2.0, and obtains substantial improvements in real-world short-horizon and long-horizon tasks. These results suggest that decoupled MoE action experts are especially useful in behaviorally diverse settings with frequent action-mode transitions.

Reproducibility Statement

We provide the model architecture, training objectives, implementation details, and hyperparameter settings in the main paper and appendix. The appendix also describes the datasets, evaluation protocols, and additional experimental results to support independent reproduction of our findings.

Ethics Statement

This work evaluates robotic manipulation policies in simulation and controlled physical-robot experiments and does not involve human subjects or personal data. Deployment on physical robots should include task-specific safety validation and appropriate human oversight.

AI Use Statement

We used generative AI tools solely to improve the clarity and language of the manuscript. The authors reviewed and edited all AI-assisted text and take full responsibility for the final content, including its scientific claims and results.

References

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023a. URL https://arxiv.org/abs/2307.15818.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, et al. Rt-1: Robotics transformer for real-world control at scale, 2023b. URL https://arxiv.org/abs/2212.06817.

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, 2025a. URL https://arxiv.org/abs/2506.18088.

Zeren Chen, Zhelun Shi, Xiaoya Lu, Lehan He, Sucheng Qian, Zhenfei Yin, Wanli Ouyang, Jing Shao, Yu Qiao, Cewu Lu, and Lu Sheng. Rh20t-p: A primitive-level robotic dataset towards composable generalization agents, 2025b. URL https://arxiv.org/abs/2403.19622.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning, 2020. URL https://arxiv.org/abs/1910.11215.

David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979. doi: 10.1109/TPAMI.1979.4766909.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot, 2023. URL https://arxiv.org/abs/2307.00595.

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

Suning Huang, Zheyu Zhang, Tianhai Liang, Yihan Xu, Zhehao Kou, Chenhao Lu, Guowei Xu, Zhengrong Xue, and Huazhe Xu. Mentor: Mixture-of-experts network with task-oriented perturbation for visual reinforcement learning, 2025. URL https://arxiv.org/abs/2410.14972.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Xiaogang Jia, Denis Blessing, Xinkai Jiang, Moritz Reuss, Atalay Donat, Rudolf Lioutikov, and Gerhard Neumann. Towards diverse behaviors: A benchmark for imitation learning with human demonstrations, 2024. URL https://arxiv.org/abs/2402.14606.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moe++: Accelerating mixture-of-experts methods with zero-computation experts, 2024. URL https://arxiv.org/abs/2410.07348.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

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, 2023. URL https://arxiv.org/abs/2212.05055.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020. URL https://arxiv.org/abs/2006.16668.

Chengmeng Li, Junjie Wen, Yan Peng, Yaxin Peng, Feifei Feng, and Yichen Zhu. Pointvla: Injecting the 3d world into vision-language-action models. *arXiv preprint arXiv:2503.07511*, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation, 2025. URL https://arxiv.org/abs/2410.07864.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation, 2021. URL https://arxiv.org/abs/2108.03298.

Yao Mu, Tianxing Chen, Shijia Peng, Zanxin Chen, Zeyu Gao, Yude Zou, Lunkai Lin, Zhiqiang Xie, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins (early version), 2025. URL https://arxiv.org/abs/2409.02920.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. Spatialvla: Exploring spatial representations for visual-language-action model, 2025. URL https://arxiv.org/abs/2501.15830.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts, 2021. URL https://arxiv.org/abs/2106.05974.

Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022. URL https://arxiv.org/abs/2206.11251.

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. *arXiv preprint arXiv:1701.06538*, 2017.

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.

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

Jan Ole von Hartz, Tim Welschehold, Abhinav Valada, and Joschka Boedecker. The art of imitation: Learning long-horizon manipulation tasks from few demonstrations, 2024. URL https://arxiv.org/abs/2407.13432.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pp. 1723–1736. PMLR, 2023.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts, 2024. URL https://arxiv.org/abs/2408.15664.

Yixiao Wang, Mingxiao Huo, Zhixuan Liang, Yushi Du, Lingfeng Sun, Haotian Lin, Jinghuan Shang, Chensheng Peng, Mohit Bansal, Mingyu Ding, et al. Ver: Vision expert transformer for robot learning via foundation distillation and dynamic routing. *arXiv preprint arXiv:2510.05213*, 2025.

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025.

Jiawen Yu, Hairuo Liu, Qiaojun Yu, Jieji Ren, Ce Hao, Haitong Ding, Guangyu Huang, Guofan Huang, Yan Song, Panpan Cai, Cewu Lu, and Wenqiang Zhang. Forcevla: Enhancing vla models with a force-aware moe for contact-rich manipulation, 2025. URL https://arxiv.org/abs/2505.22159.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. *arXiv preprint arXiv:2503.22020*, 2025.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL https://arxiv.org/abs/2304.13705.

Zhongyi Zhou, Yichen Zhu, Junjie Wen, Chaomin Shen, and Yi Xu. Chatvla-2: Vision-language-action model with open-world embodied reasoning from pretrained knowledge, 2025a. URL https://arxiv.org/abs/2505.21906.

Zhongyi Zhou, Yichen Zhu, Minjie Zhu, Junjie Wen, Ning Liu, Zhiyuan Xu, Weibin Meng, Ran Cheng, Yaxin Peng, Chaomin Shen, et al. Chatvla: Unified multimodal understanding and robot control with vision-language-action model. *arXiv preprint arXiv:2502.14420*, 2025b.

A Details of Real-World Experiments

A.1 Hardware setting

We use an AgileX Cobot Magic mobile platform in an ALOHA configuration (Figure 7a) for short horizon tasks. Each arm is an AgileX Piper with six degrees of freedom and a one-DoF parallel gripper. The platform is equipped with three RealSense D435 RGB-D cameras: one head-mounted camera capturing RGB images at 640 $\times$ 480 resolution and 30 Hz, and two wrist-mounted cameras providing visual feedback from the end-effector perspectives.



(a) AgileX Cobot Magic.

(b) Partial experiment task-related assets.

Figure 7: Experimental setup and task assets.

A.2 Task configuration

For training VLA models, we collect 150 real world demonstrations for the place cup task, and 100 demonstrations for the other tasks.

Adjust Bottle Task Configuration: This task requires using a binary gripper to lift a horizontally placed bottle into an upright position. The task tests arm selection based on bottle orientation: when the bottle points left, the robot uses the right arm to grasp and lift it; when the bottle points right, the left arm is used. This tests the policy's ability to reason about spatial relationships and select appropriate manipulation strategies based on object orientation.

Stack Plate Task Configuration: This task requires stacking blue and green bowls in a specific sequence. It tests dual-arm coordination and fine-grained spatial control. To evaluate robustness to spatial variations and color-position associations, we randomize the task setup. The experimental setup incorporates both spatial and semantic variations. Bowl positions are randomized within the workspace to test spatial generalization. Furthermore, to rule out position-dependent bias, we test two color configurations with equal frequency: Condition A (Blue-Left, Green-Right) and Condition B (Blue-Right, Green-Left), with 25 trials dedicated to each case.

Click Bell Task Configuration: The click bell task involves pressing a bell mechanism with high spatial precision. This task presents a unique challenge: the scene looks nearly identical before and after pressing the bell. The only difference is the brief moment when the bell rings. This creates difficulty for imitation learning algorithms that rely on visual state changes. To accommodate task-specific demands, we adjust the evaluation protocol by relaxing time limits to buffer initial policy oscillations while enforcing strict spatial accuracy constraints to ensure the gripper precisely targets the actuation area (Figure 7b).

Place Cup Task Configuration: The place cup task uses a transparent cup (Figure 7b), which is visually challenging. We design three spatial configurations to evaluate spatial generalization across 150 trials: (1) coaster in the center with cup on the left or right (50 trials each, single-arm manipulation), and (2) coaster on the left or right with cup on the opposite side (25 trials each,

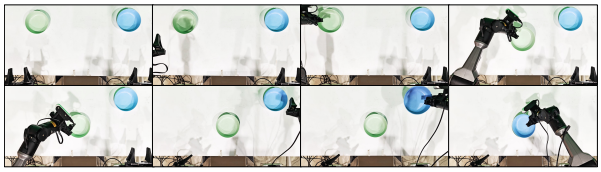
requiring bi-manual coordination as the distance exceeds single-arm reach). This tests transparent object manipulation, spatial adaptation, and multi-arm coordination.

Key Experimental Observations: During real-world deployment, we observed several important phenomena that reveal the learned behaviors of manipulation policies:

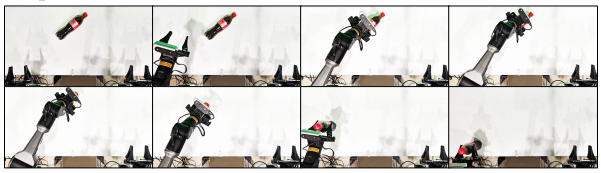
Speed matters for click bell task: The click bell task presents a unique challenge. Although our model assumes Markovian dynamics, this task is inherently non-Markovian—the scene appears nearly identical before and after the bell is pressed. Success depends entirely on the intermediate motion trajectory. We found that within a fixed action horizon, faster inference leads to higher success rates. Policies that move the end-effector to the target position more quickly complete the task more reliably. This is because the model can not only rely on visual feedback to distinguish success from failure.

Grasp location critically affects place cup task: The place cup task reveals the importance of grasp point selection. For binary grippers handling smooth, low-friction surfaces like the transparent cup shown in Figure 7b, torque balance is critical. We observed that grasping the cup sidewall (Figure 7b, red box) results in much lower success rates than grasping the bottom (Figure 7b, green box). Sidewall grasps are prone to slipping due to torque imbalance. Interestingly, **AdaMoE-VLA** shows a clear preference for bottom grasps compared to π_0. This preference directly explains why **AdaMoE-VLA** achieves substantially higher success rates in this task.

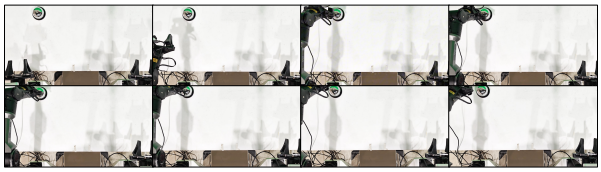
Representative rollouts from real-world experiments are shown in Figure 8.



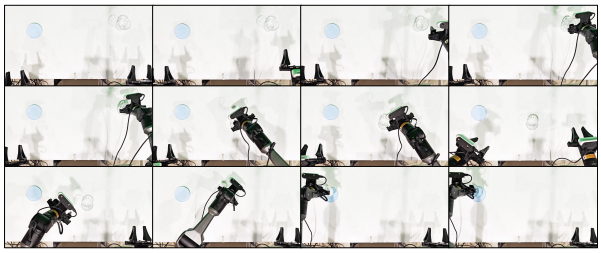
(a) Stack plate: Position the first bowl and stack the second bowl above it.



(b) Adjust bottle: Use the arm to lift the bottle head-up from the table.



(c) Click bell: Press the center top of the metal bell.



(d) Place cup: Pick up the cup and place it on the coaster.

Figure 8: Manipulation task demonstrations: bowl stacking, bottle adjustment, bell pressing, and cup placement.

Long-horizon real-world task. To assess real-world long-horizon manipulation ability, we design a sequential box-handling task on the IMETA Y1 dual-arm robotic platform, an ALOHA-style robot

platform. The robot receives only a single natural-language instruction: *"Open the box, take out the bag inside the box, place it on the right side, fold the box, place the bag on the box."* The task consists of several dependent stages, as shown in Fig. 9: the robot must first open the box, then grasp and remove the bag, place it beside the box, and finally fold the box into the target configuration. For training, we collect 50 full-task demonstrations and additional subtask demonstrations for the more challenging stages, including 50 demonstrations for *open the box* and 150 demonstrations for *fold the box*. We do not collect a separate large-scale dataset for the intermediate pick-and-place stage, as it is relatively easier and is already covered by the full-task demonstrations. This data allocation reflects the uneven difficulty of different manipulation stages, where opening and folding the box require more precise contact-rich control.

This task is substantially more challenging than isolated short-horizon manipulation, since errors made in early stages directly affect the feasibility of later stages. It also involves frequent transitions between different manipulation modes, such as reaching, opening, grasping, lifting, placing, and folding. Therefore, this setting provides a practical evaluation of **AdaMoE-VLA**'s ability to model heterogeneous action patterns and maintain robust execution over extended real-world interaction horizons.

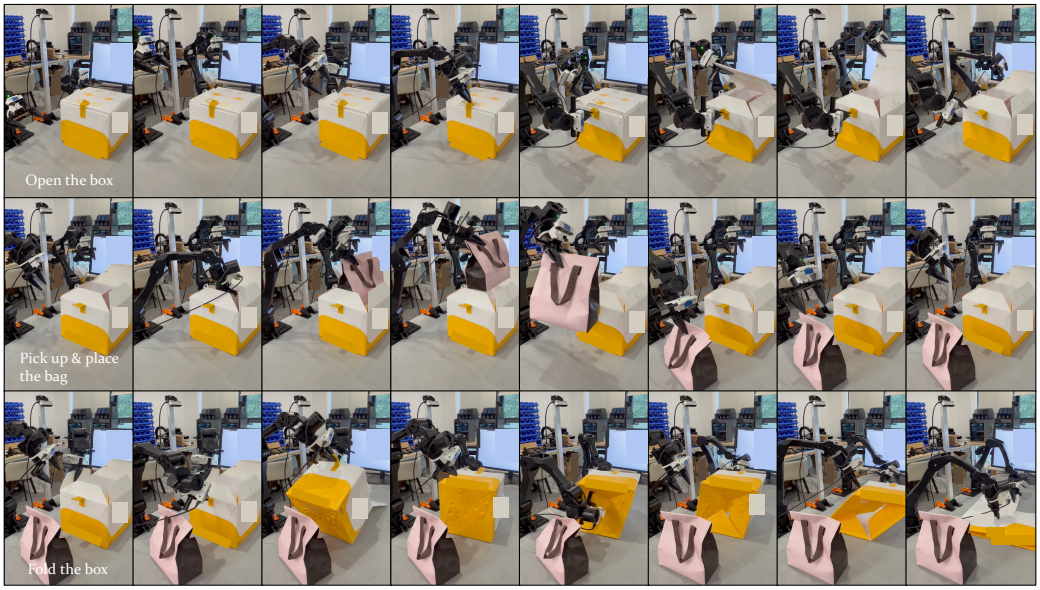


Figure 9: **Rollout visualization of the long-horizon real-world box task.** The robot sequentially completes three manipulation stages: *open the box*, *pick up and place the bag*, and *fold the box*. This task requires multiple stage transitions and long-horizon coordination, providing a real-world evaluation setting with frequent action-mode switches.

B TRAINING DETAILS

Tables 5, 6, and 7 show the detailed hyperparameters for training π_0 and **AdaMoE-VLA** across different settings. We use consistent training configurations for both models whenever possible. The main difference is that **AdaMoE-VLA** includes router-specific parameters. Both models share the same batch size (32), optimizer (AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$), gradient clipping norm (1.0), and EMA decay (0.99).

For LIBERO, we train both models for 90,000 steps. For RoboTwin, we train for 120,000 steps due to higher task complexity. Both simulation experiments use a peak learning rate of 2.5×10^{-5} and start from the same pretrained π_0 checkpoint. Regarding the MoE configuration, our main LIBERO results employ 8 experts with top-2 selection. In contrast, for LIBERO ablation studies, RoboTwin benchmarks, and real-world experiments, we utilize 4 experts with top-1 selection. A router learning rate of 5×10^{-5} is used to ensure stable sparse routing.

Table 5: Key training hyper-parameters on LIBERO.

Parameter	π_0	**AdaMoE-VLA**
Batch size	32	32
Total training steps	90,000	90,000
Peak learning rate	2.5×10^{-5}	2.5×10^{-5}
Router learning rate	–	5×10^{-5}
Number of experts	–	8
Top-k selection	–	2
λ_{balance}	–	0.01
Optimizer	AdamW	AdamW
β_1, β_2	0.9, 0.95	0.9, 0.95
Gradient clipping norm	1.0	1.0
EMA decay	0.99	0.99
Inherited weights	Pretrained π_0	Pretrained π_0

Table 6: Key training hyper-parameters on RoboTwin.

Parameter	π_0	**AdaMoE-VLA**
Batch size	32	32
Total training steps	120,000	120,000
Peak learning rate	2.5×10^{-5}	2.5×10^{-5}
Router learning rate	–	5×10^{-5}
Number of experts	–	4
Top-k selection	–	1
λ_{balance}	–	0.01
Optimizer	AdamW	AdamW
β_1, β_2	0.9, 0.95	0.9, 0.95
Gradient clipping norm	1.0	1.0
EMA decay	0.99	0.99
Inherited weights	Pretrained π_0	Pretrained π_0

For real-robot deployment, we use short fine-tuning schedules to reduce overfitting on limited real-world demonstrations. For short-horizon tasks, both π_0 and **AdaMoE-VLA** are trained for 60,000 steps, where π_0 is initialized from the RoboTwin-trained π_0 checkpoint and **AdaMoE-VLA** is initialized from the RoboTwin-trained **AdaMoE-VLA** checkpoint. This simulation-to-real initialization transfers manipulation priors from RoboTwin and accelerates real-world adaptation.

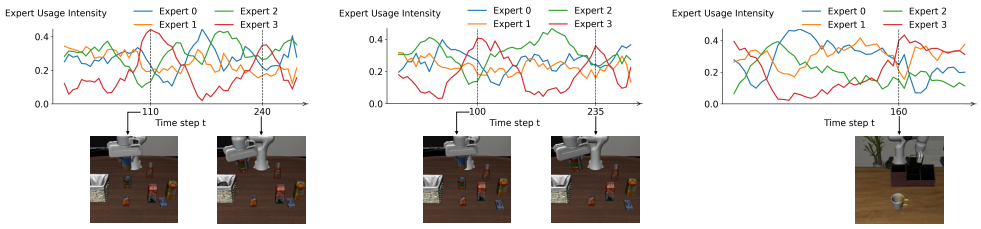
For the long-horizon task, we compare $\pi_{0.5}$ with $\pi_{0.5}$ + **AdaMoE-VLA** under a controlled setting. Both models are initialized from the same pretrained $\pi_{0.5}$ checkpoint and fine-tuned for 20,000 steps using identical training data, optimizer settings, and learning rate schedule. The only difference is the action-expert architecture: $\pi_{0.5}$ keeps the original action expert, while $\pi_{0.5}$ + **AdaMoE-VLA** equips it with **AdaMoE**.

Table 7: Key Training Hyper-parameters on Real Robot.

Parameter	**Short-horizon real-world tasks**		**Long-horizon real-world task**	
	π_0	**AdaMoE-VLA**	$\pi_{0.5}$	$\pi_{0.5}$ + **AdaMoE-VLA**
Batch size	32	32	64	64
Total training steps	60,000	60,000	20,000	20,000
Peak learning rate	2.5×10^{-5}	2.5×10^{-5}	2.5×10^{-5}	2.5×10^{-5}
Router learning rate	–	5×10^{-5}	–	5×10^{-5}
Number of experts	–	4	–	4
Top-k selection	–	1	–	1
λ_{balance}	–	0.01	–	0.01
Optimizer	AdamW	AdamW	AdamW	AdamW
β_1, β_2	0.9, 0.95	0.9, 0.95	0.9, 0.95	0.9, 0.95
Gradient clipping norm	1.0	1.0	1.0	1.0
EMA decay	0.99	0.99	0.99	0.99
Inherited weights	RoboTwin π_0	RoboTwin **AdaMoE-VLA**	Pretrained $\pi_{0.5}$	Pretrained $\pi_{0.5}$

C Analysis of Expert Activation Patterns

We analyze expert activation patterns to understand how different experts contribute to manipulation tasks. Figure 10 visualizes the usage intensity of experts at layer L, where expert usage intensity measures the accumulated contribution (sum of router and *Scale Adapter* weights) of each expert at each frame. Our **AdaMoE-VLA** uses an end-to-end architecture where MoE layers operate in high-dimensional space. This makes it difficult to find clear one-to-one matches between experts and specific actions. However, we observe some interesting patterns. For the same task "put both the alphabet soup and the tomato sauce", all experts show similar token distributions in subfigures (a) and (b). Across different tasks, some experts show consistent behavior during certain operations. For example, in subfigures (a), (b), and (c), Expert 3 shows more token usage when the policy does target positioning and gripper release. These patterns suggest possible links between expert activation and manipulation phases. More detailed statistical results are in Appendix G.



(a) put both the alphabet soup and the tomato sauce in the basket

(b) put both the alphabet soup and the tomato sauce in the basket

(c) pick up the book and place it in the back compartment of the caddy

Figure 10: Visualization of expert usage intensity.

The expert usage intensity at frame t for expert i is defined as the proportion of the combined weights assigned to that expert:

$$\text{Intensity}_i(t) = \frac{1}{T_{\text{denoise}}} \sum_{s=1}^{T_{\text{denoise}}} \frac{N_i^{(s)}(t)}{N_{\text{total}}(t)} \tag{8}$$

where $N_i^{(s)}(t)$ denotes the cumulative combined weight (sum of router probability and *Scale Adapter* value) of tokens assigned to expert i at denoising step s for frame t, $N_{\text{total}}(t)$ represents the total combined weight summed over all tokens at the corresponding step, and $T_{\text{denoise}} = 10$ represents the number of equally-spaced denoising steps in our flow matching inference process.

D Additional Ablation Study

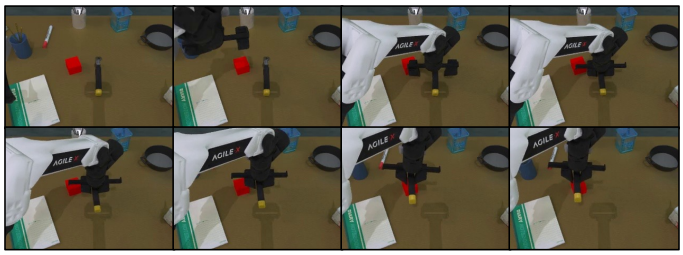
We conduct hyperparameter ablation experiments under a fixed load balancing loss weight of $\lambda_{\text{balance}} = 0.01$. Table 8 shows the results on RoboTwin benchmark with different numbers of experts and top-k values. The configuration with 12 experts and top-1 selection achieves the best performance (51.1%).

Table 8: Hyperparameter Ablation Results on RoboTwin Benchmark.

#Experts	Top-k	Success Rate (%)
8	1	45.2
8	2	47.1
12	1	**51.1**
12	2	43.8
16	1	47.8
16	2	44.1

E SIMULATION TASK DETAILS

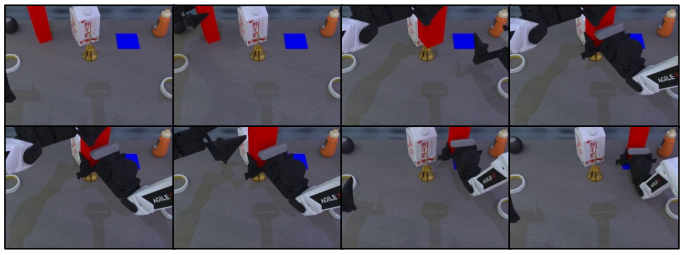
Figure 11 and Figure 12 present qualitative results of representative experiments on the RoboTwin 2.0 and LIBERO datasets, respectively. These keyframes illustrate the diversity of the evaluation tasks, ranging from precise dual-arm manipulation to long-horizon household activities.



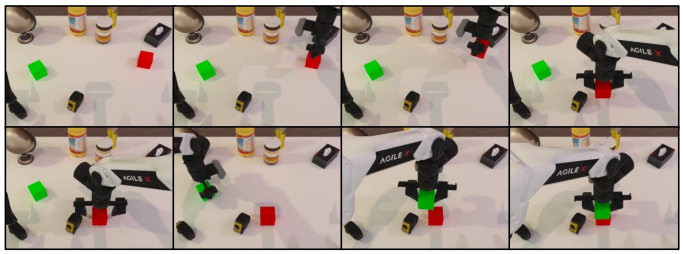
(a) Beat block hammer: Grab the grippy handle hammer, then strike the block.



(b) Place dual shoes: Pick up two the shoe for walking, tips left, and set them in the orange shoe-box.

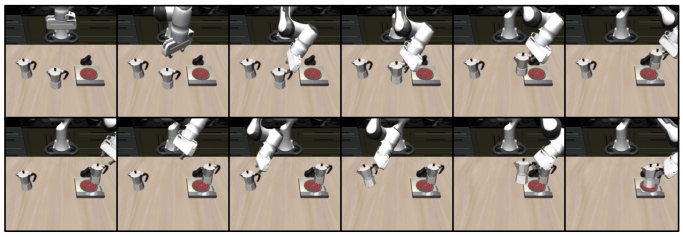


(c) Handover block: With the left arm, grab the red block, pass it to the right, and set it on the blue pad.

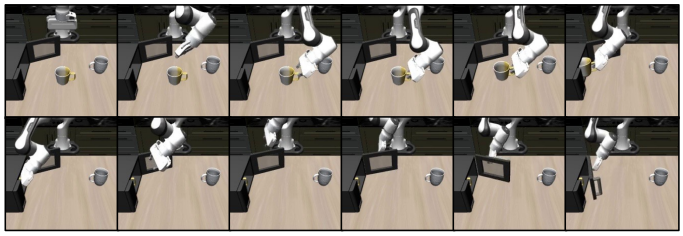


(d) Stack blocks two: Set red block in the center, then position green block on top of it.

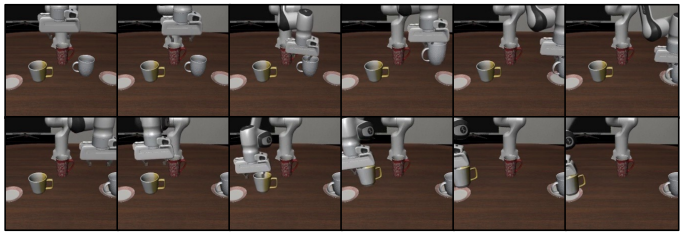
Figure 11: RoboTwin 2.0 manipulation task demonstrations (from top to bottom): (a) beat block hammer, (b) place dual shoes, (c) handover block, and (d) stack blocks two.



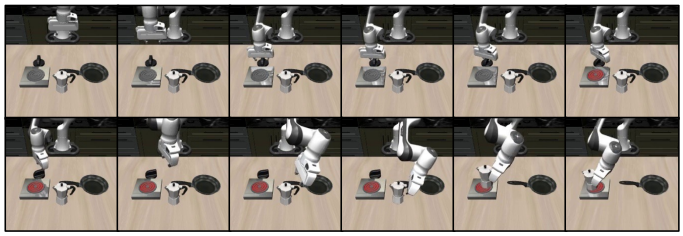
(a) Put both moka pots on the stove.



(b) Put the yellow and white mug in the microwave and close it.



(c) Put the white mug on the left plate and put the yellow and white mug on the right plate.



(d) Turn on the stove and put the moka pot on it.

Figure 12: LIBERO manipulation task demonstrations.

F COMPUTATIONAL EFFICIENCY ANALYSIS

Table 9 compares two models with approximately equal parameter counts: our efficient parallel MoE approach (N expert + Shared) and the traditional dense-matched baseline (N+1 expert matched Dense). Our MoE architecture only modifies the action expert module, implemented with a two-level parallelization strategy using JAX's vmap primitive. In the table, (B, S, D) denotes input dimensions where B is batch size, S is sequence length, and D is hidden dimension. Note that the computational overhead of the router and *Scale Adapter* is negligible compared to the expert computations.

Implementation details: Our architecture processes shared and routed experts in two parallel branches. The shared expert operates on all tokens independently. For routed experts, we sort tokens by expert assignment and reshape them into batches of size [E, T, D], where E is the number of experts and T is tokens per expert. We then use `jax.vmap` with `in_axes=(0,0,0,0)` to force parallel execution across the expert dimension. Each expert processes its assigned tokens simultaneously, and we merge results after reordering tokens back to their original positions. For the router, we use a random assignment strategy that distributes tokens uniformly across experts, which matches the balanced load distribution observed in our trained routers. This design ensures that both branches and all routed experts compute in parallel, maximizing GPU utilization.

Table 9: Model Performance Comparison: MoE vs Dense Models.

(a) Model Parameters, FLOPs, and Memory Usage.

Model	**Params**	**Inference FLOPs**	**Training FLOPs**	**Memory**
π_0	3.24B	1.75 TFLOPs	3.12 TFLOPs	6.60GB
4 expert + Shared	3.84B	1.82 TFLOPs ↓14.8%	3.14 TFLOPs ↓2.9%	7.87GB
5 expert matched Dense	3.84B	2.14 TFLOPs	3.24 TFLOPs	7.87GB
8 expert + Shared	4.45B	1.82 TFLOPs ↓28.0%	3.14 TFLOPs ↓6.3%	9.16 GB
9 expert matched Dense	4.45B	2.53 TFLOPs	3.36 TFLOPs	9.16 GB
12 expert + Shared	5.05B	1.82 TFLOPs ↓37.6%	3.14 TFLOPs ↓9.5%	10.44 GB
13 expert matched Dense	5.05B	2.92 TFLOPs	3.47 TFLOPs	10.44 GB
16 expert + Shared	5.66B	1.82 TFLOPs ↓45.0%	3.14 TFLOPs ↓12.5%	11.73 GB
17 expert matched Dense	5.66B	3.31 TFLOPs	3.59 TFLOPs	11.73 GB
32 expert + Shared	8.07B	1.82 TFLOPs ↓62.6%	3.14 TFLOPs ↓22.6%	16.85 GB
33 expert matched Dense	8.07B	4.88 TFLOPs	4.06 TFLOPs	16.85 GB

(b) Inference Latency Comparison.

Model (B, S, D)	**Action Expert Latency (ms)**			**Total Latency (ms) (1, 51, 1024)**
	(1, 51, 1024)	**(1, 101, 1024)**	**(32, 51, 1024)**	
π_0	1.44	2.08	5.82	71.10
4 expert + Shared	3.99	4.12	13.02	92.90
5 expert matched Dense	3.53	3.91	21.62	91.30
8 expert + Shared	4.87	5.61	14.08	104.84
9 expert matched Dense	4.83	5.88	35.01	109.84
12 expert + Shared	5.52	6.42	15.69	116.02
13 expert matched Dense	6.10	6.83	51.15	124.41
16 expert + Shared	7.18	8.02	16.55	125.55
17 expert matched Dense	7.51	8.80	69.97	131.80
32 expert + Shared	12.11	12.17	18.28	174.16
33 expert matched Dense	12.21	14.38	129.46	177.43

Computational efficiency (Table 9a): Our method maintains nearly constant inference FLOPs regardless of expert count. From 4 to 32 experts, inference FLOPs stay at 1.82 TFLOPs. In contrast, the dense baseline scales linearly from 2.14 to 4.88 TFLOPs. At 32 experts, our approach saves 62.6% inference FLOPs and 22.6% training FLOPs compared to the dense baseline.

Inference speed (Table 9b): Our parallel strategy significantly accelerates expert computation. For example, with 32 experts and configuration (32, 51, 1024), our method achieves 18.28ms action expert latency versus 129.46ms for the dense baseline, representing over 7× speedup. The speedup advantage is consistent across different expert counts, demonstrating the effectiveness of our parallelization approach.

Why our method is faster: First, shared and routed experts execute in parallel through separate computation branches. Second, vmap-based parallelization within routed experts fully utilizes GPU resources by processing all experts simultaneously. The efficiency advantage grows with expert count, showing that our two-level parallel architecture is a promising direction for scaling MoE models. This approach scales better than traditional dense implementations while maintaining competitive performance.

Comparison with Standard 1:4 Dense Baseline. To further validate the efficiency of our architecture, we conducted an additional comparison against a standard Dense baseline where the MLP expansion ratio is fixed at 1:4 ($d_{\text{mlp}} = 4 \times d_{\text{model}}$), representing the conventional Transformer configuration. As shown in Table 10, while the standard dense baseline exhibits lower latency than the parameter-matched dense baseline (due to fewer total parameters), it still suffers from linear scaling costs. In contrast, our **AdaMoE-VLA** maintains a flat latency profile. At the 32-expert scale, our method achieves a **5.2× speedup** (18.28ms vs. 95.24ms) compared to the standard dense baseline, reinforcing the suitability of our approach for real-time control tasks even against streamlined dense architectures.

Table 10: Inference Latency comparison with Standard Dense Baseline ($d_{\text{model}} : d_{\text{mlp}} = 1 : 4$). Comparison is conducted with Batch Size=32 and Sequence Length=51.

Experts (N)	Latency (ms)		Speedup	Δ Time (ms) (Saved)
	Std. Dense	Ours		
4	18.74	13.02	1.4×	5.72
8	26.78	14.08	1.9×	12.70
12	39.12	15.69	2.5×	23.43
16	48.90	16.55	3.0×	32.35
32	95.24	18.28	5.2×	76.96

G *Scale Adapter* IMPACT ANALYSIS

G.1 *Scale Adapter* METRICS

Scale Magnitude. Measures the average absolute scale adjustment for the activated expert:

$$\text{Scale Magnitude} = \mathbb{E}_{t,j}[|s_{t,j,e^*}|] \tag{9}$$

where s_{t,j,e^*} is the scale value for the top-1 expert e^*. Our observed range (0.181–0.210) indicates consistent adaptation magnitude across suites.

Positive/Negative Ratio. Compares the frequency of positive versus negative scale adjustments:

$$\text{Pos/Neg Ratio} = \left(\frac{\#\{s_{e^*} > 0\}}{\text{total}} \Big/ \frac{\#\{s_{e^*} < 0\}}{\text{total}} \right) \tag{10}$$

We observe a 77%/23% split. This systematic amplification bias suggests the adapter primarily reinforces confident expert predictions rather than performing symmetric corrections.

Relative Impact. Quantifies the adapter's contribution relative to the router's gating score:

$$\text{Relative Impact} = \frac{|s_{t,j,e^*}|}{|g_{t,j,e^*}|} \times 100\% \tag{11}$$

A range of 61–68% indicates that scale adjustments are comparable in magnitude to the router outputs, confirming the adapter's critical role in task-specific adaptation rather than merely fine-tuning.

Within-Task CV. Assesses consistency across executions:

$$\text{CV} = \frac{\sigma(\text{scale})}{\mu(\text{scale})} \times 100\% \tag{12}$$

Low CVs (average 3.11%) across all tasks confirm high reproducibility, indicating the adapter learns stable task-specific patterns rather than random noise.

G.2 Expert Router Metrics

Gini Coefficient. Measures expert specialization ($K = 4$):

$$\text{Gini} = \frac{\sum_{i=1}^{K} \sum_{j=1}^{K} |a_i - a_j|}{2K^2 \bar{a}} \tag{13}$$

Low observed values (0.029–0.061) indicate effective load balancing across experts without over-specialization.

Entropy. Evaluates routing diversity:

$$\text{Entropy} = -\sum_{i=1}^{K} p_i \log p_i \tag{14}$$

Observed values of 1.379–1.385 (near the maximum of ≈ 1.39) demonstrate near-optimal utilization of all experts.

Variance Ratio. Our primary metric for task differentiation, comparing between-task variance to within-task execution noise:

$$\text{Variance Ratio} = \frac{\sigma_{\text{between-task}}}{\sigma_{\text{within-task}}} \tag{15}$$

where $\sigma^2_{\text{between}}$ is the variance of task mean scales, and σ^2_{within} averages the variance across 50 trajectories per task.

G.3 Suite-Level Analysis (Table 11)

Table 11: Comprehensive analysis of MoE and *Scale Adapter* across LIBERO.

Suite	*Scale Adapter* Metrics			Expert Router Metrics		Task Analysis
	Scale Magnitude	Pos/Neg Ratio (%)	Relative Impact (%)	Gini Coeff.	Entropy	Variance Ratio
Spatial	0.210 ± 0.017	76.9 / 23.1	67.8 ± 4.2	0.029 ± 0.006	1.385 ± 0.001	2.86
Goal	0.205 ± 0.016	79.1 / 20.9	66.9 ± 4.1	0.050 ± 0.023	1.381 ± 0.004	2.53
Object	0.181 ± 0.006	73.9 / 26.1	60.8 ± 1.7	0.047 ± 0.011	1.382 ± 0.002	0.95
LIBERO-10	0.186 ± 0.011	77.3 / 22.7	61.9 ± 3.2	0.061 ± 0.015	1.379 ± 0.003	1.75
Average	0.196	76.8 / 23.2	64.4	0.047	1.382	2.02

***Scale Adapter* Behavior.** Scale magnitude ranges from 0.181 (Object) to 0.210 (Spatial), showing consistent adaptation strength across suites. Positive scales dominate (77% average), indicating the adapter primarily amplifies expert outputs rather than suppressing them. Relative impact averages 64.4%, demonstrating scale adjustments contribute substantially to final outputs—comparable in magnitude to router predictions themselves.

Expert Router Patterns. Gini coefficients range from 0.029 (Spatial) to 0.061 (LIBERO-10). Low values across all suites indicate balanced expert utilization without over-specialization. Entropy values of 1.379–1.385 approach the maximum $\log(4) \approx 1.39$ (99% utilization), showing all four experts contribute effectively without collapse.

Task Differentiation and Performance Correlation. Variance ratio predicts MoE effectiveness. Object suite's ratio of 0.95 falls below 1.0, meaning within-task variation exceeds between-task differences. These 10 tasks differ only in object appearance while sharing identical pick-and-place actions. This explains the -0.8% performance drop (98.0% vs. π_0's 98.8% in Table 1): our action-level MoE cannot differentiate visually distinct but action-identical tasks, creating architectural overhead without specialization benefits. In contrast, Goal (2.53) and Spatial (2.86) achieve good differentiation through diverse action patterns. LIBERO-10 shows moderate ratio (1.75) but delivers

the largest gain (+5.6%, 90.8% vs. 85.2%), as complex multi-step tasks benefit most from action-level specialization. This correlation validates our design: MoE excels when tasks differ in actions, not just visual context.

G.4 Task-Level Analysis (Table 12)

Scale Magnitude Consistency. Object suite shows the smallest range (0.173–0.195, Δ=0.022), while Goal suite spans 0.174–0.230 (Δ=0.056). This 2.5$\times$ difference confirms Object tasks require nearly identical adjustments while Goal tasks need diverse adaptations.

Table 12: Per-Task Analysis: *Scale Adapter* and Expert Router Metrics (40 LIBERO Tasks).

Suite	Task	Scale Mag.	Rel. Impact (%)	Gini Coeff.	Entropy	Within CV (%)	Pos Ratio (%)	Neg Ratio (%)	Trajs
Spatial	Bowl: between plate & ramekin	0.248	76.8	0.033	1.384	3.54	78.7	21.3	50
	Bowl: next to ramekin	0.204	66.2	0.035	1.384	2.07	75.9	24.1	50
	Bowl: from table center	0.195	64.3	0.035	1.384	2.43	76.1	23.9	50
	Bowl: on cookie box	0.223	72.1	0.018	1.386	4.56	77.5	22.5	50
	Bowl: in top drawer	0.205	66.6	0.033	1.385	2.80	76.8	23.2	50
	Bowl: on ramekin	0.223	70.8	0.021	1.385	2.52	77.0	23.0	50
	Bowl: next to cookie box	0.199	65.6	0.026	1.385	2.51	75.4	24.6	50
	Bowl: on stove	0.186	61.5	0.028	1.385	1.80	75.8	24.2	50
	Bowl: next to plate	0.216	69.0	0.030	1.384	2.17	75.8	24.2	50
	Bowl: on wooden cabinet	0.200	65.5	0.026	1.385	3.70	76.3	23.7	50
	Suite Average	*0.210*	*67.8*	*0.029*	*1.385*	*2.81*	*76.9*	*23.1*	*500*
Goal	Open middle drawer	0.174	58.7	0.073	1.378	3.11	75.6	24.4	50
	Bowl on stove	0.210	69.3	0.029	1.384	1.72	80.5	19.5	50
	Wine bottle on cabinet top	0.205	66.2	0.044	1.382	2.13	81.7	18.3	50
	Open drawer + bowl inside	0.209	67.9	0.052	1.380	2.83	80.8	19.2	50
	Bowl on cabinet top	0.214	70.5	0.045	1.382	2.36	81.0	19.0	50
	Push plate to stove front	0.203	65.3	0.027	1.384	4.69	77.0	23.0	50
	Cream cheese in bowl	0.220	70.8	0.038	1.382	4.65	79.3	20.7	50
	Turn on stove	0.199	65.4	0.076	1.377	2.66	76.6	23.4	50
	Bowl on plate	0.230	73.2	0.024	1.385	2.90	80.6	19.4	50
	Wine bottle on rack	0.185	62.2	0.096	1.374	2.72	77.9	22.1	50
	Suite Average	*0.205*	*66.9*	*0.050*	*1.381*	*2.98*	*79.1*	*20.9*	*500*
Object	Alphabet soup $\rightarrow$ basket	0.178	59.7	0.051	1.381	4.24	73.1	26.9	50
	Cream cheese $\rightarrow$ basket	0.186	62.1	0.062	1.379	5.92	73.6	26.4	50
	Salad dressing $\rightarrow$ basket	0.184	61.5	0.054	1.380	2.41	73.5	26.5	50
	BBQ sauce $\rightarrow$ basket	0.181	61.0	0.062	1.379	3.80	73.3	26.7	50
	Ketchup $\rightarrow$ basket	0.173	58.5	0.038	1.383	1.91	73.5	26.5	50
	Tomato sauce $\rightarrow$ basket	0.195	64.8	0.044	1.382	5.25	73.7	26.3	50
	Butter $\rightarrow$ basket	0.177	59.6	0.045	1.382	2.18	73.9	26.1	50
	Milk $\rightarrow$ basket	0.182	60.6	0.025	1.385	3.90	74.1	25.9	50
	Chocolate pudding $\rightarrow$ basket	0.177	59.7	0.041	1.383	2.08	73.0	27.0	50
	Orange juice $\rightarrow$ basket	0.180	60.3	0.052	1.381	1.59	73.3	26.7	50
	Suite Average	*0.181*	*60.8*	*0.047*	*1.382*	*3.33*	*73.9*	*26.1*	*500*
LIBERO-10	Soup & tomato sauce $\rightarrow$ basket	0.192	64.0	0.043	1.382	5.05	77.3	22.7	50
	Cheese box & butter $\rightarrow$ basket	0.190	63.4	0.066	1.378	1.67	76.7	23.3	50
	Stove on + moka pot	0.173	58.2	0.073	1.377	3.02	75.9	24.1	50
	Bowl in drawer + close	0.180	60.7	0.043	1.382	3.11	76.5	23.5	50
	White & yellow mugs on plates	0.195	65.2	0.059	1.380	2.82	78.0	22.0	50
	Book in caddy compartment	0.169	57.2	0.043	1.382	2.83	76.3	23.7	50
	Mug + pudding arrangement	0.192	63.3	0.070	1.377	4.19	78.1	21.9	50
	Soup & cheese box $\rightarrow$ basket	0.198	65.7	0.056	1.380	1.73	77.3	22.7	50
	Both moka pots on stove	0.171	57.2	0.093	1.373	5.08	75.9	24.1	50
	Mug in microwave + close	0.197	64.5	0.061	1.379	3.68	76.5	23.5	50
	Suite Average	*0.186*	*61.9*	*0.061*	*1.379*	*3.32*	*77.3*	*22.7*	*500*
Overall Average (40 tasks)		**0.196**	**64.4**	**0.047**	**1.382**	**3.11**	**76.8**	**23.2**	**2000**

Relative Impact Patterns. Impact ranges from 57–77% across tasks, confirming scale adjustments are comparable in magnitude to router outputs. Object suite shows tighter clustering (58.5–64.8%,

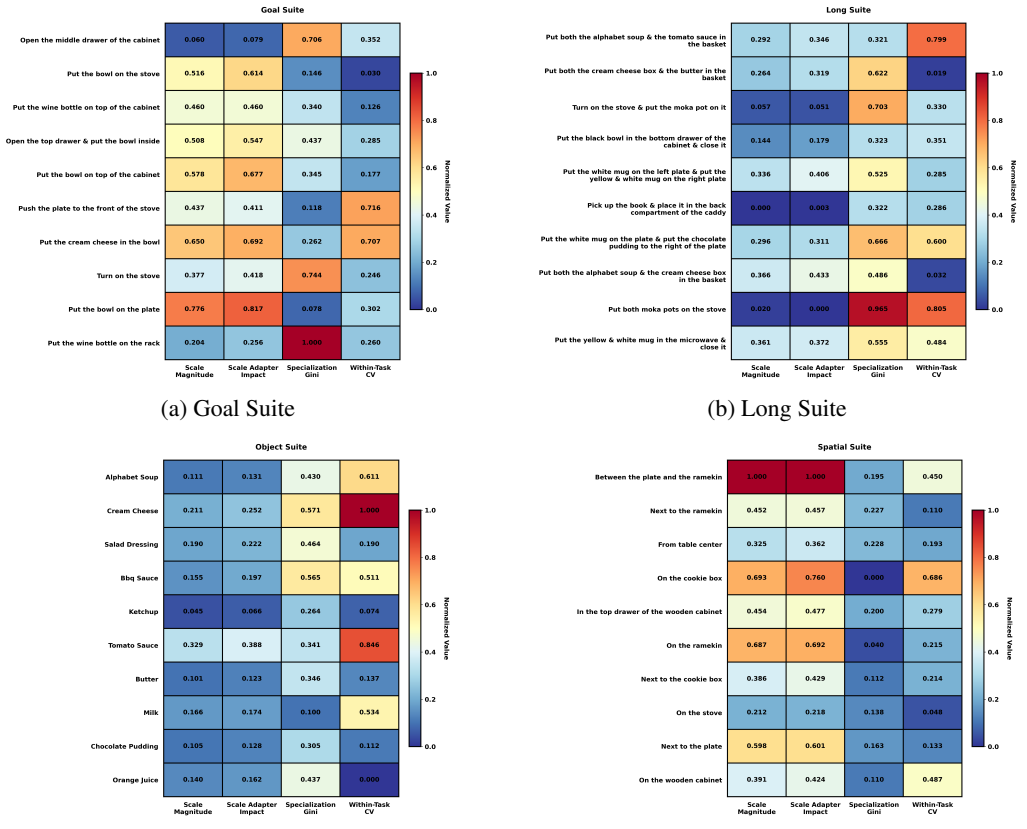
Δ=6.3%), while Goal suite spans wider (58.7–73.2%, Δ=14.5%). Tasks like "Bowl on plate" achieve 73.2% impact, meaning scale nearly matches gating strength. This high contribution explains why *Scale Adapter* is critical for task-specific adaptation.

Positive/Negative Distribution. All tasks show 70–80% positive scales, confirming systematic amplification rather than balanced bidirectional adjustment. This pattern suggests the adapter learns to boost confident expert predictions. The 3:1 ratio indicates scale primarily reinforces rather than corrects router decisions.

Specialization Patterns. Object suite's Gini coefficients cluster tightly (0.025–0.062), showing uniform expert usage. Goal suite spans 0.024–0.096, with "Wine bottle on rack" achieving highest specialization. LIBERO-10's "Both moka pots" reaches 0.093, indicating multi-object tasks benefit most from expert differentiation.

Within-Task Stability. CV values stay below 6% for all tasks, confirming reproducibility. Object suite's "Cream cheese" shows highest CV (5.92%), possibly from object detection noise. Goal suite's "Both moka pots" (5.08%) and "Push plate" (4.69%) reflect task complexity rather than instability.

Figure 13 visualizes task-level metrics across all 40 LIBERO tasks, with each metric normalized to [0,1] for cross-suite comparison. Object suite's uniform blue-green coloring confirms task similarity (variance ratio 0.95), while Long suite's structured color diversity correlates with the largest gain (+5.6%). The visual correspondence between color patterns and performance outcomes validates our variance-ratio framework.



(a) Goal Suite

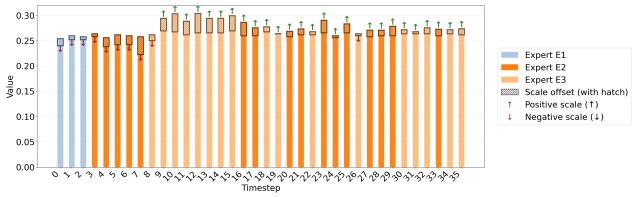
(b) Long Suite

(c) Object Suite

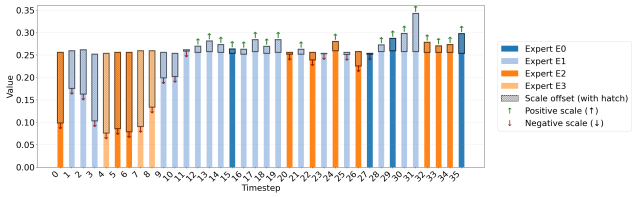
(d) Spatial Suite

Figure 13: Expert activation pattern analysis across LIBERO benchmark suites. Each heatmap visualizes normalized metrics quantifying task-specific specialization.

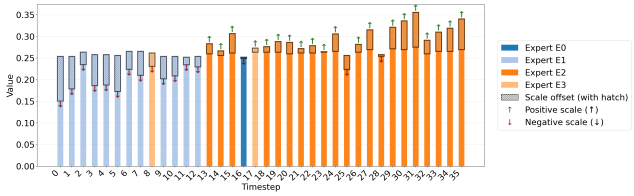
We additionally visualize the outputs of the *Scale Adapter* and router for individual tokens under an **AdaMoE** policy with number of experts = 4 and top-k = 1. We present two representative examples. The first example shows the *Scale Adapter* and router outputs across all denoising steps for the 4th token at layer 5 in the task "open the middle drawer of the cabinet". This token corresponds to inferring the action at time step 3, where the time step represents the physical action execution sequence in the manipulation task. The results are shown in Figures 14 and 15. The second example shows the *Scale Adapter* and router outputs across all denoising steps for the 8th token at layer 17 in the task "pick up the orange juice and place it in the basket". This token corresponds to inferring the action at time step 7, representing the 7th action in the physical execution sequence. The results are shown in Figures 16 and 17.



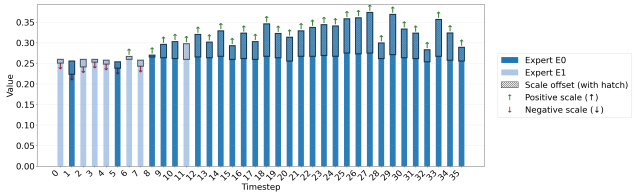
(a) Denoise Step 0.



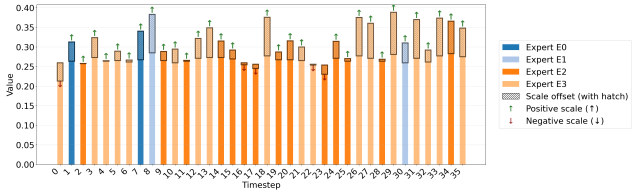
(b) Denoise Step 1.



(c) Denoise Step 2.

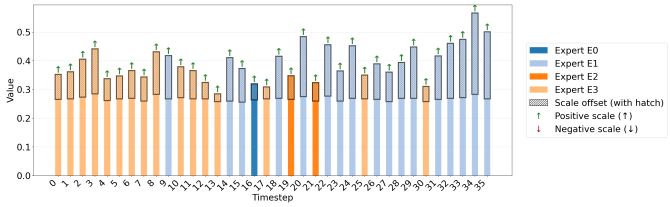


(d) Denoise Step 3.

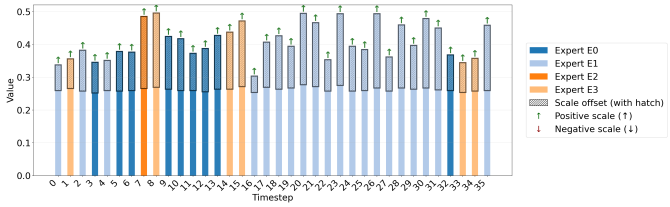


(e) Denoise Step 4.

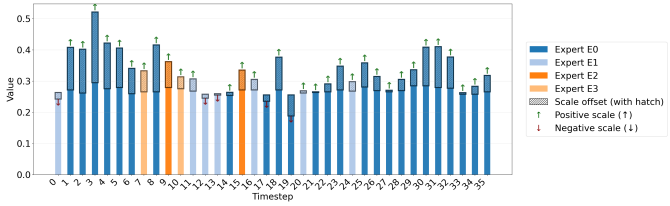
Figure 14: Expert selection and *Scale Adapter* values across denoising steps (Steps 0-4) for Layer 5, Token 3. Each subplot shows the top-1 expert and scale values. Green arrows (↑): positive scales; red arrows (↓): negative scales. Task: "open the middle drawer of the cabinet".



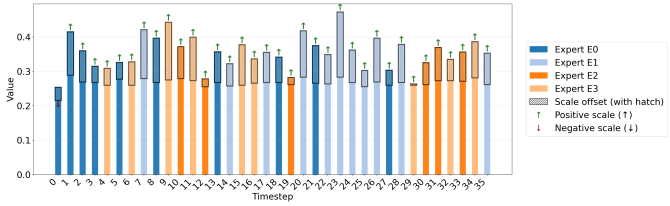
(a) Denoise Step 5.



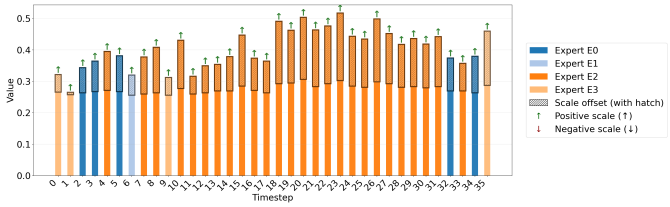
(b) Denoise Step 6.



(c) Denoise Step 7.

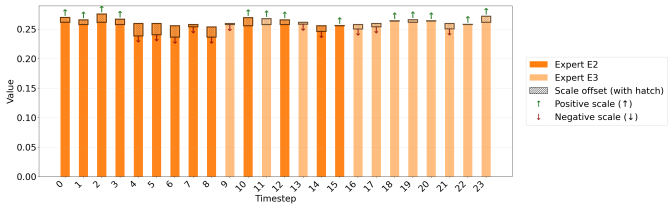


(d) Denoise Step 8.

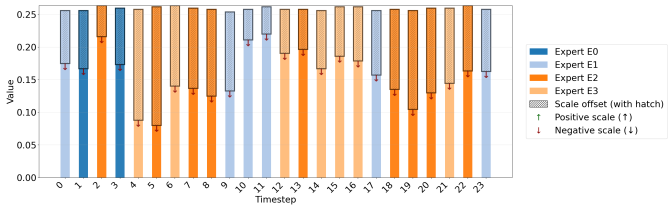


(e) Denoise Step 9.

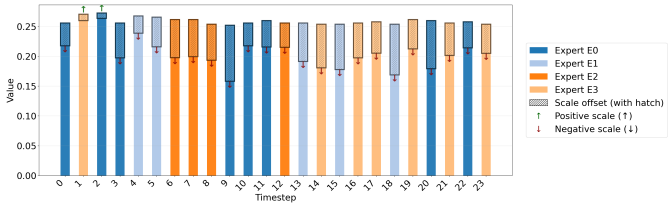
Figure 15: Expert selection and *Scale Adapter* values across denoising steps (Steps 5-9) for Layer 5, Token 3. Each subplot shows the top-1 expert and scale values. Green arrows (↑): positive scales; red arrows (↓): negative scales. Task: "open the middle drawer of the cabinet".



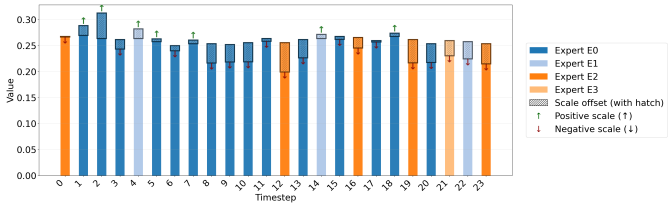
(a) Denoise Step 0.



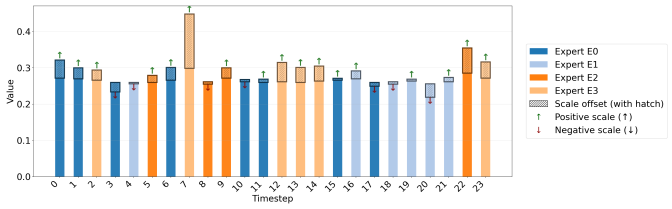
(b) Denoise Step 1.



(c) Denoise Step 2.

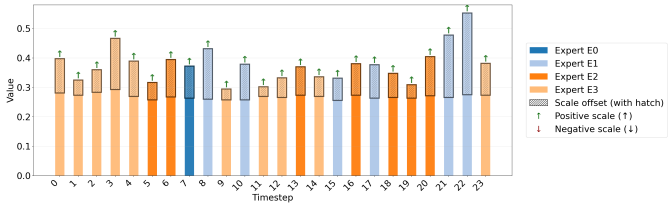


(d) Denoise Step 3.

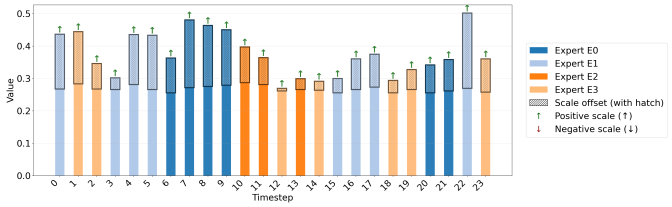


(e) Denoise Step 4.

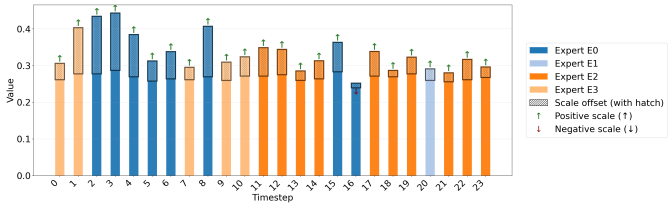
Figure 16: Expert selection and *Scale Adapter* values across denoising steps (Steps 0-4) for Layer 17, Token 7. Each subplot shows the top-1 expert and scale values. Green arrows (↑): positive scales; red arrows (↓): negative scales. Task: "pick up the orange juice and place it in the basket".



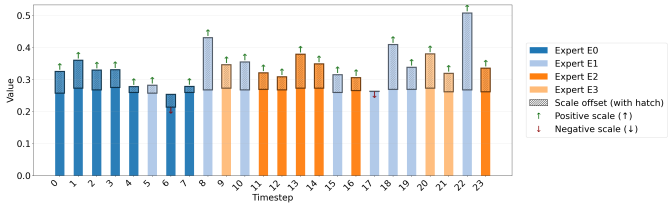
(a) Denoise Step 5.



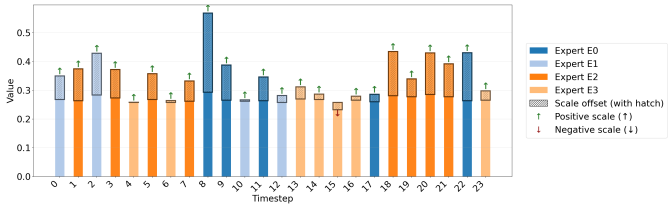
(b) Denoise Step 6.



(c) Denoise Step 7.



(d) Denoise Step 8.



(e) Denoise Step 9.

Figure 17: Expert selection and *Scale Adapter* values across denoising steps (Steps 5-9) for Layer 17, Token 7. Each subplot shows the top-1 expert and scale values. Green arrows (↑): positive scales; red arrows (↓): negative scales. Task: "pick up the orange juice and place it in the basket".

H GRADIENT ANALYSIS

To empirically validate our hypothesis that **AdaMoE-VLA** mitigates gradient interference, we conducted a granular gradient analysis on the router and *Scale Adapter* parameters across all 18 MoE layers. This analysis was performed using a configuration of 4 experts with top-1 selection. We utilized a fixed load balancing coefficient of $\lambda_{\text{balance}} = 0.01$ and extracted the gradients derived from both the flow-matching loss ($\mathcal{L}_\tau$) and the load balancing loss ($\mathcal{L}_{\text{balance}}$) for both components.

We specifically computed two angular metrics to measure the alignment with the primary task optimization direction:

1. **Router Misalignment:** The angle between the router's total gradient (computed as $\nabla_{\theta_R}\mathcal{L}_\tau + 0.01 \cdot \nabla_{\theta_R}\mathcal{L}_{\text{balance}}$) and the pure task gradient ($\nabla_{\theta_R}\mathcal{L}_\tau$).
2. **Combined System Alignment:** The angle between the effective combined gradient (the sum of router and *Scale Adapter* gradients) and the pure task gradient.

As visualized in Figure 3, the *Router Misalignment* (top) consistently fluctuates between 3° and 6°, indicating that the auxiliary balance loss indeed pulls the router away from the optimal task trajectory.

In contrast, the *Combined System Alignment* (bottom) remains significantly lower (approximately 0.6° to 0.8°). This demonstrates that the *Scale Adapter* effectively acts as a correction term, compensating for the deviation caused by the balance loss and ensuring that the final parameter updates remain highly aligned with the primary flow-matching objective.

I WHY APPLY MOE TO THE ACTION HEAD RATHER THAN THE VLA BACKBONE?

We apply MoE to the action head rather than the vision-language backbone for two reasons: stronger task-aligned supervision and lower inference overhead. This analysis is conducted on the $\pi_{0.5}$ base model, where we compare inserting the same MoE configuration into either the vision-language backbone or the action head.

First, the action head is directly optimized by the flow-matching objective, so experts inserted into this module receive more explicit specialization signals for action generation. In contrast, MoE experts in the vision-language backbone are only indirectly supervised through the downstream action loss. Under robot manipulation data alone, this indirect supervision may be insufficient to induce reliable vision-language expert specialization. Backbone-level MoE may require additional vision-language supervision, such as VQA-style data, which is outside the scope of our robot manipulation training setting.

Second, action-head MoE is more efficient than backbone MoE. Using the $\pi_{0.5}$ base model, we measure the inference time of dense and MoE variants separately for the vision-language backbone and the action head under the same MoE configuration with 4 experts and top-1 routing. As shown in Table 13, inserting MoE into the backbone roughly doubles the backbone latency, while inserting MoE into the action head incurs a smaller absolute overhead.

Table 13: Inference latency comparison for applying MoE to different modules on the $\pi_{0.5}$ base model. We report mean $\pm$ standard deviation in milliseconds.

Module	**Dense (ms)**	**+MoE (ms)**
VL backbone	49.41 ± 1.88	103.76 ± 4.89
Action head	21.81 ± 1.17	38.22 ± 2.22

For the action head, the additional latency introduced by **AdaMoE** can also be reduced by using fewer denoising steps. As shown in Table 14, reducing the number of denoising steps from 10 to 4 substantially decreases latency while maintaining comparable LIBERO performance. This makes action-head MoE more compatible with real-time control constraints.

Overall, the action head provides a better trade-off between expert specialization and inference efficiency. It receives direct action-level supervision from the flow-matching loss, while also preserving

Table 14: Effect of denoising steps on LIBERO performance and action-head latency using the $\pi_{0.5}$ + **AdaMoE-VLA** model.

Denoising Steps	LIBERO Avg. ↑	Latency (ms) ↓
10	98.30	38.22
8	98.65	32.76
6	98.55	28.59
4	98.75	21.44

practical real-time inference through sparse activation and adjustable denoising steps. We therefore apply **AdaMoE** to the action head rather than the VLA backbone.

J Negative Transfer in Low-Diversity Data Regimes

Although **AdaMoE** generally improves VLA action modeling, we observe that the benefit of MoE can be smaller in low-diversity data regimes. For example, LIBERO-Object contains relatively homogeneous action patterns compared with more diverse benchmarks such as RoboTwin 2.0 and real-world long-horizon manipulation. In such settings, the need for action-level expert specialization is naturally limited. Introducing additional routed experts may therefore provide limited benefit and can sometimes introduce extra variance, leading to mild negative transfer on specific task suites.

This behavior is consistent with our main hypothesis: **AdaMoE** is most useful when the action distribution contains heterogeneous and frequently switching modes. In low-diversity regimes, however, the model may not need to activate additional experts for every token. To improve robustness in such cases, we explore a null expert mechanism inspired by prior MoE variants such as MoE++ (Jin et al., 2024). The null expert allows the model to suppress unnecessary routed-expert activation when additional specialization is not beneficial.

Table 15 shows that adding a null expert removes the performance drop on LIBERO-Object while preserving the overall improvement over the base π_0 policy. In particular, **AdaMoE-VLA** with a null expert improves the Object suite from 98.8% to 99.2%, and improves the average success rate from 94.2% to 96.3%.

Table 15: Effect of adding a null expert on LIBERO. The null expert helps suppress unnecessary expert activation in low-diversity regimes.

Model	Spatial	Object	Goal	Long	Avg.
π_0	96.8	98.8	95.8	85.2	94.2
AdaMoE-VLA + null expert (base π_0)	99.2	99.2	96.2	90.6	96.3

These results suggest that negative transfer in low-diversity regimes is not an intrinsic instability of **AdaMoE**, but rather a capacity-allocation issue. When action patterns are homogeneous, the model benefits from the ability to avoid unnecessary expert usage. In contrast, in behaviorally diverse settings with frequent action-mode transitions, **AdaMoE** provides clearer gains, as reflected by the larger improvements on RoboTwin 2.0 and real-world experiments.

K Action-Mode Clustering and Switching Statistics

To analyze the action diversity of different manipulation datasets, we perform an unsupervised action-mode clustering analysis based on short-horizon robot-configuration transitions. For each trajectory, we first extract the robot configuration sequence in terms of joint positions (`qpos`). For bimanual tasks, the joint positions of both arms are concatenated into a single configuration vector. Instead of directly clustering single-timestep configurations, we compute configuration-to-configuration deltas over a short temporal window, which better capture local motion patterns.

Specifically, we construct a local temporal descriptor using four consecutive configuration transitions:

$$\mathbf{z}_t = [\Delta\mathbf{q}_t, \Delta\mathbf{q}_{t+1}, \Delta\mathbf{q}_{t+2}, \Delta\mathbf{q}_{t+3}], \tag{16}$$

where each configuration delta is defined as:

$$\Delta \mathbf{q}_t = \mathbf{q}_{t+1} - \mathbf{q}_t. \tag{17}$$

Here, $\mathbf{q}_t$ denotes the robot joint-position vector (`qpos`) at timestep t. Thus, each descriptor $\mathbf{z}_t$ summarizes local robot motion over four consecutive state transitions. This short-horizon qpos-delta descriptor captures local manipulation patterns such as reaching, opening, grasping, lifting, placing, and folding, while being more informative than a single-frame configuration or a single-timestep action.

We apply K-means clustering to these qpos-delta descriptors to obtain discrete action modes. The number of clusters is selected using the Davies–Bouldin Index (DBI). Specifically, we sweep over candidate cluster numbers and choose the value that achieves the lowest DBI score, corresponding to compact and well-separated action clusters. After clustering, each short-horizon segment $\mathbf{z}_t$ is assigned a mode label m_t according to its nearest cluster center.

Based on the resulting mode labels, we compute the mode-switching frequency for each trajectory. Given a sequence of assigned action-mode labels $\{m_1, m_2, \ldots, m_T\}$, the number of mode switches is defined as:

$$N_{\text{switch}} = \sum_{t=1}^{T-1} \mathbb{I}\left[m_t \neq m_{t+1}\right], \tag{18}$$

where $\mathbb{I}[\cdot]$ is the indicator function. This statistic measures how frequently the trajectory transitions between different action modes within an episode.

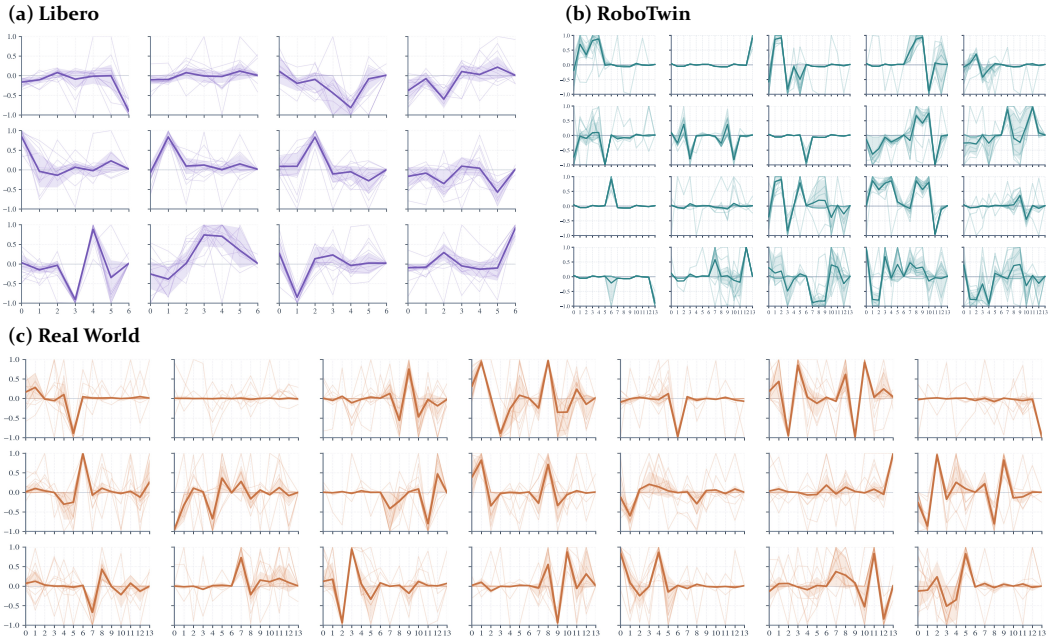


Figure 18: **Visualization of discovered action-mode clusters.** We construct local motion descriptors by concatenating robot qpos deltas over four consecutive state transitions, and then perform K-means clustering with the number of clusters selected by the Davies–Bouldin Index (DBI). Each subplot represents one discovered action mode. Thin curves show individual qpos-delta segments assigned to the cluster, while the bold curve shows the cluster-average pattern. The resulting mode labels are further used to compute the mode-switching frequency of each trajectory.

Figure 18 visualizes the discovered action-mode clusters on LIBERO, RoboTwin, and real-world data. Each subplot corresponds to one cluster, where thin curves denote individual four-step qpos-delta segments and the bold curve denotes the cluster-average pattern. The clustering results show that RoboTwin and real-world demonstrations contain more diverse and rapidly changing motion patterns than LIBERO. This supports our observation that more complex manipulation settings involve richer action-mode structures and more frequent transitions, motivating the use of adaptive mixture-of-action experts for VLA action modeling.

L Comparison with Activated-Parameter-Matched Dense Baselines

To further examine whether the gains of **AdaMoE-VLA** simply come from increasing the number of activated parameters, we compare against activated-parameter-matched dense baselines on LIBERO. Specifically, we construct dense action experts whose activated parameter count is matched to **AdaMoE-VLA** with 4 routed experts and top-1 routing. This comparison is intended to test whether directly enlarging the dense action MLP can provide similar benefits to sparse multi-expert action modeling.

However, this dense baseline differs from **AdaMoE-VLA** in its initialization path. The enlarged dense MLP has a different hidden dimension from the original pretrained action expert, and therefore cannot directly inherit the original pretrained MLP weights. As a result, the expanded dense MLP is randomly initialized. In contrast, **AdaMoE-VLA** follows a sparse upcycling strategy: the shared expert and routed experts are initialized from the pretrained action expert, preserving compatibility with the original VLA initialization. Therefore, this comparison should be interpreted as an activated-parameter-matched dense expansion baseline, rather than as a fully initialization-matched baseline.

Table 16 and Table 17 report the results for π_0 and $\pi_{0.5}$, respectively. For both base models, the dense expansion baseline underperforms the original base policy. With the π_0 base, the dense baseline achieves 90.9% average success rate, lower than the original π_0 model at 94.2%. With the stronger $\pi_{0.5}$ base, the dense baseline achieves 93.6%, also lower than the original $\pi_{0.5}$ model at 96.85%. These results suggest that simply increasing activated dense capacity is not sufficient in this setting, especially when the expansion breaks compatibility with pretrained action-expert weights.

Table 16: Comparison with an activated-parameter-matched dense baseline on LIBERO using the π_0 base. The dense baseline is matched to **AdaMoE-VLA** with 4 experts and top-1 routing, but its expanded MLP is randomly initialized because the changed hidden dimension prevents direct inheritance of pretrained MLP weights.

Model	Spatial	Object	Goal	Long	Avg.
Base π_0	96.8	98.8	95.8	85.2	94.2
Dense	95.6	97.2	90.8	80.0	90.9

Table 17: Comparison with an activated-parameter-matched dense baseline on LIBERO using the $\pi_{0.5}$ base. The dense baseline is matched to **AdaMoE-VLA** with 4 experts and top-1 routing, but its expanded MLP is randomly initialized because the changed hidden dimension prevents direct inheritance of pretrained MLP weights.

Model	Spatial	Object	Goal	Long	Avg.
Base $\pi_{0.5}$	98.8	98.2	98.0	92.4	96.9
Dense	97.0	96.8	93.6	87.0	93.6

Overall, the dense baseline results indicate that the improvements of **AdaMoE-VLA** should not be attributed merely to a larger activated parameter count. A key practical advantage of our sparse action-expert design is that it increases action-modeling capacity while preserving compatibility with pretrained initialization. This initialization compatibility is important for VLA fine-tuning, where robot data is limited and randomly initialized enlarged dense layers can degrade the pretrained action-generation capability.

M Selection of the Load-Balancing Coefficient

The load-balancing coefficient λ_{balance} controls the trade-off between expert utilization and task-oriented optimization:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\tau} + \lambda_{\text{balance}} \mathcal{L}_{\text{balance}}. \tag{19}$$

We do not treat $\lambda_{\text{balance}} = 0.01$ as a universal constant. Our selection principle is to use the smallest coefficient that prevents persistent routing concentration while maintaining a low and stable deviation

Table 18: Effect of the Load-Balancing Coefficient on LIBERO (Success Rate %). All configurations use the π_0 backbone, four routed experts, and top-1 selection.

Coefficient	Spatial	Object	Goal	Long	Average
$\lambda_{\text{balance}} = 0.001$	98.0	96.0	96.0	88.0	94.5
$\lambda_{\text{balance}} = 0.01$	**99.6**	95.0	**97.2**	**92.0**	**96.0**
$\lambda_{\text{balance}} = 0.05$	97.8	95.2	96.4	91.0	95.1

from the flow-matching optimization direction. We assess these requirements using expert-utilization statistics and gradient-alignment diagnostics.

For the controlled setting with $K = 4$ routed experts and top-1 selection, we monitor the activation concentration score

$$q_e = \frac{n_e(\mathcal{W})}{\sum_{j=1}^{K} n_j(\mathcal{W})}, \qquad S_{\text{act}} = \sum_{e=1}^{K} q_e^2, \tag{20}$$

where $n_e(\mathcal{W})$ denotes the number of tokens dispatched to routed expert e within a sliding training window $\mathcal{W}$. The always-active shared expert is excluded from this statistic. Uniform utilization gives $S_{\text{act}} = 0.25$, whereas complete concentration on a single expert gives $S_{\text{act}} = 1$. In this setting, we empirically use a persistent value above 0.35 as an indicator of substantial routing concentration, suggesting that a stronger balancing coefficient should be considered. This threshold is specific to the evaluated routing configuration.

Balanced utilization alone is insufficient for selecting λ_{balance}, since excessive regularization can interfere with task-dependent expert weighting. We therefore also inspect the Combined–FM gradient angle analyzed in Appendix H. An increased or unstable angle indicates that the combined Router–Scale-Adapter update deviates more strongly from the flow-matching direction. Among candidate coefficients that avoid persistent routing concentration, we favor the smallest coefficient that maintains low and stable Combined–FM angles.

For the selected coefficient $\lambda_{\text{balance}} = 0.01$, Figure 3 shows router-only angles of approximately $3°$–$6°$, while the combined Router–Scale-Adapter angles remain approximately $0.6°$–$0.8°$. These observations support the role of the Scale Adapter in reducing the interference associated with load balancing.

Table 18 reports the controlled LIBERO comparison using the π_0 backbone, four routed experts, and top-1 selection, with all other training and evaluation settings held fixed. Among the tested coefficients, $\lambda_{\text{balance}} = 0.01$ achieves the highest average success rate of 96.0%, compared with 94.5% for 0.001 and 95.1% for 0.05. The effect is particularly visible on LIBERO-Long, where success increases from 88.0% at 0.001 to 92.0% at 0.01.

These results corroborate our diagnostic-based choice: both insufficient balancing and overly strong regularization reduce performance in the evaluated setting. For a new task or routing configuration, we apply the same selection principle rather than assuming that either the coefficient 0.01 or the concentration threshold 0.35 transfers unchanged.

N Comparison with DeepSeek-Style Loss-Free Balancing

We evaluate a DeepSeek-style auxiliary-loss-free balancing baseline to examine whether removing the auxiliary load-balancing loss can reproduce the improvements of AdaMoE-VLA. This comparison adapts the bias-based balancing strategy to our VLA action head rather than reproducing the complete DeepSeek language-model architecture.

The loss-free baseline selects experts using routing scores modified by expert-wise biases. These biases are updated according to observed expert loads:

$$b_e \leftarrow b_e + \gamma \, \text{sign}(\bar{c} - c_e), \qquad \bar{c} = \frac{1}{K} \sum_{j=1}^{K} c_j, \tag{21}$$

where c_e is the observed token count for expert e and γ is the bias-update rate. The biases affect expert selection but are not added to the weights multiplying the selected expert outputs.

Table 19: Comparison with DeepSeek-Style Loss-Free Balancing on LIBERO (Success Rate %). "Freeze Nk" means that bias updates are frozen during the final Nk training steps. All MoE configurations use four routed experts, one shared expert, and top-1 routing.

Method	Spatial	Object	Goal	Long	Average
Loss-free, $\gamma = 10^{-2}$	78.8	49.0	56.6	56.6	60.3
Loss-free, $\gamma = 10^{-3}$	99.2	98.6	95.6	83.8	94.3
Loss-free, $\gamma = 10^{-4}$	98.0	97.0	**97.6**	85.2	94.5
Loss-free, $\gamma = 10^{-5}$	79.4	88.2	58.8	72.2	74.7
Loss-free, $\gamma = 10^{-4}$, freeze 10k	99.4	97.6	96.8	79.8	93.4
Loss-free, $\gamma = 10^{-4}$, freeze 20k	**99.6**	97.0	96.4	76.2	92.3
Loss-free, $\gamma = 10^{-4}$, freeze 30k	99.0	**98.8**	96.8	84.4	94.8
Vanilla MoE	98.6	97.0	96.0	87.4	94.8
AdaMoE-VLA	**99.6**	95.0	97.2	**92.0**	**96.0**

We use the same π_0 backbone, Gemma action expert, copied expert initialization, four routed experts, one shared expert, and top-1 routing as the controlled AdaMoE-VLA comparison. The optimizer, learning rates, batch size, training data, training budget, and rollout evaluation protocol are matched within each benchmark. We sweep the bias-update rate and additionally evaluate whether freezing the bias updates near the end of training improves performance.

On LIBERO, all compared configurations are trained for 90k steps. We evaluate $\gamma \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ without bias-update freezing. For $\gamma = 10^{-4}$, we additionally freeze the bias updates during the final 10k, 20k, or 30k training steps. Freezing applies only to the load-driven bias updates; ordinary model optimization continues. Table 19 reports the complete results.

The strongest loss-free configuration by average success uses $\gamma = 10^{-4}$ with the final 30k bias updates frozen. It achieves 94.8%, matching Vanilla MoE and remaining 1.2 percentage points below AdaMoE-VLA. The effect of freezing is non-monotonic: freezing the final 10k, 20k, and 30k steps yields 93.4%, 92.3%, and 94.8%, respectively.

The largest remaining suite-level gap occurs on LIBERO-Long. The configuration with the highest average loss-free performance obtains 84.4% on this suite. Even when selecting the best loss-free configuration specifically for LIBERO-Long, its highest success rate is 85.2%, obtained with $\gamma = 10^{-4}$ and no freezing. AdaMoE-VLA reaches 92.0%, exceeding this suite-specific best result by 6.8 percentage points.

Across the tested bias-update rates and freezing schedules, loss-free performance ranges from 60.3% to 94.8%. By comparison, the three tested AdaMoE-VLA balancing coefficients span a $50\times$ range and achieve 94.5%–96.0%. These observations indicate sensitivity to the loss-free controller settings in the evaluated fine-tuning regime.

On RoboTwin 2.0, we additionally evaluate $\gamma \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ without bias-update freezing. Table 20 reports the complete task-level results. The strongest tested loss-free configuration uses $\gamma = 10^{-5}$ and achieves 38.3% average success, compared with 36.7% and 35.5% for $\gamma = 10^{-4}$ and 10^{-3}, respectively.

Table 20: DeepSeek-Style Loss-Free Balancing on RoboTwin 2.0. Each group reports success rates for three bias-update rates γ, without bias-update freezing.

Task	10^{-5}	10^{-4}	10^{-3}	Task	10^{-5}	10^{-4}	10^{-3}	Task	10^{-5}	10^{-4}	10^{-3}
Beat Block Hammer	82%	86%	74%	Place Can Basket	44%	54%	46%	Stack Blocks Two	18%	20%	22%
Click Bell	24%	30%	30%	Pick Dual Bottles	44%	24%	28%	Stack Bowls Three	60%	50%	46%
Click Alarmclock	36%	48%	40%	Place Cans Plasticbox	64%	16%	66%	Turn Switch	26%	24%	34%
Handover Block	14%	12%	14%	Place Object Stand	58%	66%	56%	Pick Diverse Bottles	42%	14%	36%
Move Can Pot	4%	4%	0%	Place A2B Left	22%	34%	28%	Place Dual Shoes	28%	46%	14%
Move Playingcard Away	46%	52%	40%	Place A2B Right	24%	28%	26%	**Average**	**38.3%**	**36.7%**	**35.5%**
Place Phone Stand	58%	44%	50%	Put Bottles Dustbin	34%	46%	24%				

Compared with the baseline results in Table 2, the best tested loss-free configuration remains below π_0 at 40.4%, Vanilla MoE at 45.9%, and AdaMoE-VLA at 49.7%. AdaMoE-VLA exceeds this

configuration by 11.4 percentage points on average and performs better on 15 of the 19 evaluated tasks. Thus, changing the bias-update rate improves the loss-free baseline but does not close the overall performance gap within the tested range.

The loss-free baseline uses load-driven selection biases to balance expert utilization, whereas AdaMoE-VLA retains explicit load-balancing regularization and introduces input-conditioned residual expert weights:

$$w_e(x) = \left[\mathrm{softmax}(R(x))\right]_e + S_e(x), \qquad e \in \mathcal{I}(x), \tag{22}$$

where $\mathcal{I}(x)$ denotes the selected expert set. The Scale Adapter therefore provides an additional task-supervised pathway for adjusting the contributions of selected experts.

Together with the gradient-alignment analysis, these results support the benefit of preserving task-adaptive expert composition while maintaining balanced utilization in VLA fine-tuning. Under the evaluated configurations, tuning load-driven selection biases alone does not reproduce AdaMoE-VLA's overall or long-horizon performance.

O Compute Resources

All training experiments were conducted on 8 NVIDIA A800 GPUs with 80GB memory per GPU. We used JAX for training, and GPU memory followed JAX's preallocation behavior. In our experiments, the preallocation fraction was set to 90%, corresponding to approximately 72GB reserved memory per A800 GPU. This preallocation does not necessarily indicate the exact peak tensor memory usage, but reflects the GPU memory reserved by the JAX runtime during training.

Table 21 summarizes the wall-clock training time for each type of experiment. On LIBERO, training one model takes approximately 18 hours. On RoboTwin 2.0, training one model takes approximately 24 hours. For real-world experiments, training one model takes approximately 10 hours for the short-horizon tasks and approximately 10 hours for the long-horizon task. For evaluation, LIBERO experiments were evaluated on NVIDIA A800 GPUs, while RoboTwin 2.0 evaluation and real-world robot inference were conducted on a single NVIDIA RTX 4090 GPU.

Table 21: Compute resources used in our experiments. Training time is reported as the approximate wall-clock time for training one model. All GPUs used are NVIDIA GPUs.

Experiment setting	Training GPUs	GPU mem.	Training time	Evaluation GPU
LIBERO	8 × A800	80GB	∼18 hours	A800
RoboTwin 2.0	8 × A800	80GB	∼24 hours	RTX 4090
Real-world short-horizon	8 × A800	80GB	∼10 hours	RTX 4090
Real-world long-horizon	8 × A800	80GB	∼10 hours	RTX 4090