CAPVECTOR: LEARNING TRANSFERABLE CAPABILITY VECTORS IN PARAMETRIC SPACE FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposes a novel approach to address the challenge that pretrained vision-language-action models (VLAs) often fail to effectively improve performance and reduce adaptation costs during standard supervised finetuning (SFT). Some advanced finetuning methods with auxiliary training objectives can improve performance and reduce the number of convergence steps. However, they typically incur significant computational overhead due to the additional losses from auxiliary objectives. To simultaneously achieve the enhanced capabilities of auxiliary training with the simplicity of standard SFT, we propose a new paradigm for steering the behavior of VLAs, centered around *capability vectors*. Capability vectors specify the beneficial properties induced by auxiliary-objective SFT and are given by element-wise arithmetic operations between the auxiliary-objective SFT models and regular SFT models. The capability vectors are then merged with pretrained parameters to form a capability-enhanced meta model. With a regular SFT integrated with a lightweight orthogonal regularization loss, the capability-enhanced meta model attains performance comparable to auxiliary finetuned baselines with reduced computational overhead and higher training efficiency. Internal and external experiments demonstrate that our capability vectors (1) are effective and versatile across diverse models, (2) can generalize to novel environments and embodiments out of the box.

1 INTRODUCTION

Vision–Language–Action (VLA) models have become a dominant paradigm in current research on robotic foundation models. They map multimodal perception into executable robotic control, exhibiting a certain degree of language following and visual generalization ability. Similar to Large Language Models (LLMs) (Agarwal et al., 2025; Yang et al., 2025), training VLAs typically consists of two processes: (1) A pre-training process that allows the model to learn the mapping relation between multimodal input and action output. This process is conducted on large-scale robotic datasets and costs thousands of GPU hours. (2) A finetuning process that allows the model to fit the specific task structure.

However, recent studies have revealed that pre-trained models do not exhibit the expected strong generalization capability on certain complex downstream tasks. That is, merely collecting a small number of demonstrations and performing standard supervised finetuning (SFT) is often insufficient for the model to quickly adapt to the task and achieve performance significantly superior to training from scratch (Kim et al., 2025b;a; Black et al., 2024). Several approaches aim to augment the standard SFT with an auxiliary objective. By designing auxiliary training objectives (Zheng et al., 2025; Song et al., 2025; Li et al., 2026; Bai et al., 2026; Liu et al., 2026) aimed at enhancing specific foundational capabilities, this paradigm enables the model to not only fit the target task's action distribution but also strengthen the corresponding foundational abilities (*e.g.*, spatial perception and multimodal reasoning). With appropriately designed auxiliary objectives, models can significantly reduce the number of training steps required for convergence and achieve downstream performance that surpasses that of standard SFT.

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.

Despite the above strengths, these approaches have obvious drawbacks: auxiliary objectives often introduce extra modules and additional forward passes. For example, the 3D Foundation Model in Spatial Forcing (Li et al., 2026) requires additional computation to obtain aligned targets during training, and LaRA-VLA (Bai et al., 2026) requires training of the latent chain-of-thought tokens, which incurs extra computational overhead. As the number of downstream tasks and the scale of data grow, this overhead gradually becomes prohibitive (Appendix Section D.2). This naturally motivates the following question: **Can the beneficial properties s, induced by carefully designed finetuning procedures, be transferred into the pretrained model itself, such that the model inherently possesses s?** If so, one could rely solely on standard SFT to inherit the same training efficiency and performance improvements, without incurring additional overhead.

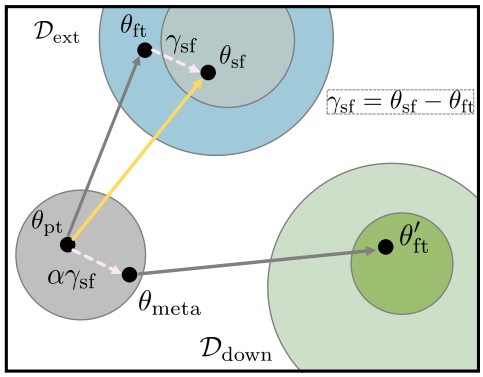

Figure 1: **Illustration of our `CapVector`.** The **grey line** indicates the regular finetuning from pretrained models to finetuned models, the **yellow line** indicates the specifically designed finetuning with auxiliary objectives, and the **pink dashed line** denotes the capability vectors. The outer circle denotes the region of the parameter space that yields workable performance on the data manifold, while the inner circle represents the subregion associated with superior performance. We observe that by extracting capability vectors from $\mathcal{D}_{\text{ext}}$ and merging them to obtain a capability-enhanced meta model θ_{meta}, the performance on $\mathcal{D}_{\text{down}}$ is improved. Variables are defined in Section 2.

We show that the properties can be extracted through arithmetic operations in parameter space and subsequently transferred. Drawing inspiration from the concept of task vectors (Ilharco et al., 2022), we treat the two gains obtained during the training process—namely, the improvement of general capabilities and the enhancement of task-specific action fitting accuracy—as two separable parameter vectors.

Specifically, we acquire two sets of finetuned model parameters by applying the auxiliary-objective SFT and the standard SFT method to the same downstream task, respectively. The difference between these two sets of parameters is named as *capability vectors* (`CapVector`). They can then be integrated into the pretrained backbone through arithmetic operations, thereby achieving model merging. The whole process is shown in Figure 1. While prior work in this field has primarily focused on obtaining an off-the-shelf specialist model via merging (Chen et al., 2025a; Fu et al., 2025; Yadav et al., 2025), it remains unclear whether such techniques can be employed to produce a better generalist model that is more suitable for arbitrary downstream finetuning while also delivering superior performance. After capability extraction, a lightweight orthogonal regularization loss is needed during downstream finetuning to prevent forgetting of capability vectors.

In experiments, we focus on investigating the extractable capabilities and the underlying training mechanism of this approach. Extensive experiments demonstrate that the merged meta model can achieve performance and training efficiency comparable to SFT methods with auxiliary objectives across multiple downstream tasks. Furthermore, we validate the versatility of `CapVector` through the experiments on diverse VLA architectures and SFT strategies. Moreover, we demonstrate that different types of capability vectors can be combined additively to produce synergistic effects. Finally, internal and external experiments in the real world demonstrate its practicality and generalization to novel environments and embodiments out of the box.

In summary, our core contributions are as follows:

- We introduce the concept of the capability vector, which represents the gain in general capabilities acquired during finetuning with auxiliary objectives in the form of model parameters. By merging these capability vectors with the pretrained model, we obtain a capability-enhanced meta model.
- Based on the meta model, we only need to make minimal modifications to standard SFT by introducing an orthogonal regularization loss to mitigate forgetting. This achieves both the simplicity of standard SFT and the high performance of auxiliary-objective finetuning during downstream training.

- Extensive experiments demonstrate our `CapVector`'s effectiveness and efficiency as a general learning strategy on various tasks, environments, and models.

2 CAPABILITY VECTORS (CAPVECTOR)

Our method consists of two stages. Before training, we transfer the capability vectors derived from the auxiliary-objective SFT, thereby obtaining an enhanced meta model that inherits the desired properties. During training, to adapt the model to downstream tasks without degrading these properties, we introduce a regularization strategy in orthogonal subspaces.

2.1 PROBLEM FORMULATION

Assume we have a pretrained VLA model $\theta_{\text{pt}} \in \mathbb{R}^d$ and a multi-task extensive dataset $\mathcal{D}_{\text{ext}}$, whose included tasks are referred to as capability extraction tasks. These tasks are specifically designed not for downstream performance, but to induce and expose particular model capabilities through parameter variations during finetuning. We denote the extracted *capability vectors* as $\gamma_{\text{ao}} \in \mathbb{R}^d$. Our goal is to obtain a more capable meta model $\theta_{\text{meta}} = \theta_{\text{pt}} + \gamma_{\text{ao}}$ that is superior to the pretrained model by acquiring general capability vectors on a small-scale set of capability extraction tasks. That is, given a downstream task dataset $\mathcal{D}_{\text{down}}$ for evaluation, under consistent training settings, the model obtained by finetuning θ_{meta} on $\mathcal{D}_{\text{down}}$ achieves better performance than the model obtained by finetuning θ_{pt}.

2.2 BEFORE TRAINING: CAPABILITY VECTORS TRANSFERRING

First, we consider employing standard SFT on the data in $\mathcal{D}_{\text{ext}}$, resulting in the finetuned model θ_{ft}:

$$\theta_{\text{ft}} = \theta_{\text{pt}} + \Delta_{\text{ft}}. \tag{1}$$

We denote Δ_{ft} as the parameter difference between the pretrained and the finetuned model.

Next, we consider the scenario of extracting capability vectors from SFT methods with auxiliary objectives, such as Spatial Forcing (Li et al., 2026) that aligns intermediate visual embeddings of VLAs with geometric representations produced by pretrained 3D foundation models to enhance spatial perception, and LaRA-VLA (Bai et al., 2026) that internalises multimodal chain-of-thought into continuous latent representations to enhance long-horizon reasoning capabilities. We denote the model θ_{ao} finetuned by these auxiliary-objective SFT methods as $\theta_{\text{ao}} = \theta_{\text{pt}} + \Delta_{\text{ao}}$.

We employ the same finetuning recipe to get θ_{ft} and θ_{ao}. Then, the capability vector γ_{ao} is given by models' element-wise difference

$$\gamma_{\text{ao}} = \theta_{\text{ao}} - \theta_{\text{ft}}. \tag{2}$$

Next, to achieve our goal of transferring the properties of θ_{ao} to θ_{pt}, we merge the capability vectors γ_{ao} and θ_{pt} and get the capability-enhanced meta model with properties:

$$\theta_{\text{meta}} = \theta_{\text{pt}} + \alpha\gamma_{\text{ao}}, \tag{3}$$

where α denotes vector weights. This provides a better initialization for further performing finetuning on any new tasks:

$$\theta'_{\text{ft}} = \theta_{\text{meta}} + \Delta'_{\text{ft}}. \tag{4}$$

2.3 DURING TRAINING: REGULARIZATION IN ORTHOGONAL SUBSPACES

While we have transferred the properties to the pretrained model, there is an obvious question: *how to retain the properties during regular finetuning?*

Because the capability vectors and the obtained meta model share the same parametric space, the parameters of the meta model undergo updates within the shared parametric space. Without the auxiliary supervision, the standard SFT can harm the properties, and this phenomenon can be more harmful with more training steps.

Some previous work (Wang et al., 2023) utilizes orthogonal regularization to maintain the model's performance and continue to learn new tasks. In our case, we aim to keep the orthogonality between

the capability vectors γ_{ao} and Δ'_{ft} to prevent interference. Our fundamental insight is rooted in the nature of finetuning: the parameter changes are not mere numerical adjustments but encapsulate crucial model update directions. Thus, orthogonality needs to satisfy:

$$\langle {\gamma_{\text{ao}}}^{(p)}, {\Delta'_{\text{ft}}}^{(p)} \rangle = 0, \forall {\gamma_{\text{ao}}}^{(p)} \in \gamma_{\text{ao}}, {\Delta'_{\text{ft}}}^{(p)} \in \Delta'_{\text{ft}}, \tag{5}$$

where p denotes a parameter in the capability vectors and task vectors. Therefore, our orthogonal regularization loss is defined as:

$$\mathcal{L}_{\text{orth}}(\gamma_{\text{ao}}, \Delta'_{\text{ft}}) = \sum_{p} \sum_{i,j} |{\gamma_{\text{ao}}}_{ij}^{(p)} {\Delta'_{\text{ft}}}_{ij}^{(p)}| \tag{6}$$

where i, j denote the element at the i-th row and j-th column of the matrix. The total training loss is:

$$\mathcal{L} = \mathcal{L}_{\text{action}} + \lambda \mathcal{L}_{\text{orth}}, \tag{7}$$

where λ is the weight of the orthogonality loss. Please note that the extra overhead induced by orthogonality loss is slight, as quantified in Appendix Section D.1. For Low-Rank Adaptation (LoRA) tuning, we only calculate $\mathcal{L}_{\text{orth}}$ between the matrix A in LoRA. This is because they represent the updating direction of the model, and the matrix B serves as linear weighting coefficients for matrix A (Büyükakyüz, 2024).

3 Experiments

In this section, we evaluate the effectiveness of our `CapVector` and offer several findings by investigating the following research questions (RQs):

- **RQ1:** Can `CapVector` effectively transfer capabilities in the domain? How does the design of the loss function and the choice of hyperparameters contribute to the performance? ***(In-distribution Effectiveness)***
- **RQ2:** Are the extracted capability vectors task-irrelevant? Do they exhibit out-of-domain transferability? ***(Out-of-distribution Effectiveness & Generalization)***
- **RQ3:** Is `CapVector` consistently effective and efficient on various VLA architectures? Can it transfer diverse capabilities (*e.g.*, spatial perception and multimodal reasoning) of different auxiliary-objectives SFT? ***(Versatility)***
- **RQ4:** Can `CapVector` realize sim-to-real transfer, *i.e.*, are the capability vectors obtained from simulated environments still effective in the real world? Can `CapVector` work across robot embodiments and real-world scenes out of the box? ***(Real-world Performance & Practicality)***
- **RQ5:** Can the superposition of distinct capability vectors enable capability composition? ***(Composition Capability)***
- **RQ6:** How do capability vectors improve training efficiency? ***(Efficiency)***

3.1 Experimental Settings

Simulated Environments. We evaluate our method on two representative simulated benchmarks, LIBERO (Liu et al., 2023) and RoboTwin 2.0 (Chen et al., 2025b). LIBERO is a widely used benchmark built on Robosuite (Zhu et al., 2020). It consists of four suites (Spatial, Object, Goal, Long), each comprising 10 tasks. Success rates are reported with 500 rollouts per suite across 3 random seeds. RoboTwin 2.0 (Chen et al., 2025b) is a bimanual manipulation benchmark built on Sapien (Xiang et al., 2020). In this paper, we focus on 10 tasks with clean backgrounds as target datasets and run 100 rollouts per task to calculate success rates. We also utilize another 5 tasks with clean and randomized backgrounds individually as capability extraction tasks in Section 3.3.

Base Models. We choose three representative VLAs, OpenVLA-OFT (Kim et al., 2025a), StarVLA (Community, 2026), and $\pi_{0.5}$ (Black et al., 2025) as our regular SFT backbones. We choose two auxiliary-objective SFT methods, Spatial Forcing (Li et al., 2026) and LaRA-VLA (Bai et al., 2026), as introduced in Section H. Following official settings, we use LoRA tuning for OpenVLA-OFT and full tuning for StarVLA and $\pi_{0.5}$.

Table 1: **In-distribution Comparison in LIBERO under various training iterations based on Spatial Forcing.** θ_{ft}: OpenVLA-OFT. θ_{ao}: Spatial Forcing. $\mathcal{D}_{\text{ext}}$: {LIBERO-Spatial}. $\mathcal{D}_{\text{down}}$: {LIBERO-Spatial, Object, Goal, Long}.

Progress	Method	Spatial	Object	Goal	Long	Average
5k Steps	Spatial Forcing (θ_{ao})	93.8%	94.8%	94.6%	66.6%	87.5%
	OpenVLA-OFT (θ_{ft})	87.0%	**99.8%**	92.8%	48.8%	82.1%
	`CapVector` w/o orthogonal loss	96.0%	99.0%	**97.4%**	68.0%	90.1%
	`CapVector` (ours)	**98.0%**	99.2%	96.6%	**73.0%**	**91.7%**
1 Epoch	Spatial Forcing (θ_{ao})	98.4%	99.6%	97.8%	84.8%	95.2%
	OpenVLA-OFT (θ_{ft})	97.0%	99.8%	96.4%	70.4%	90.9%
	`CapVector` w/o orthogonal loss	98.4%	**100.0%**	**98.0%**	86.0%	95.6%
	`CapVector` (ours)	**98.6%**	99.8%	97.6%	**90.0%**	**96.5%**
8 Epochs	Spatial Forcing (θ_{ao})	93.0%	**98.2%**	**98.4%**	87.2%	94.2%
	OpenVLA-OFT (θ_{ft})	92.8%	**98.2%**	97.8%	87.0%	93.9%
	`CapVector` w/o orthogonal loss	97.6%	97.8%	96.6%	92.2%	96.1%
	`CapVector` (ours)	**98.0%**	98.0%	96.8%	**93.6%**	**96.6%**
150k Steps	Spatial Forcing (θ_{ao})	97.2%	**99.2%**	96.8%	94.2%	96.9%
	OpenVLA-OFT (θ_{ft})	96.8%	94.8%	92.8%	86.2%	92.7%
	`CapVector` w/o orthogonal loss	97.4%	99.0%	**97.2%**	91.2%	96.2%
	`CapVector` (ours)	**98.4%**	98.4%	96.8%	**94.8%**	**97.1%**

Training Details. All experiments are conducted on NVIDIA H100 GPUs, with 1 GPU used for OpenVLA-OFT, 8 GPUs for StarVLA, and 4 GPUs for $\pi_{0.5}$. Per-device batch size is set to 8 for OpenVLA-OFT, 16 for StarVLA, and 32 for $\pi_{0.5}$. Training step is set to 150k for OpenVLA-OFT, 20k for StarVLA, and 60k for $\pi_{0.5}$. As shown in Section 2.1, we denote the training set of θ_{ao} and θ_{ft} as $\mathcal{D}_{\text{ext}}$, and denote the training set of θ'_{ft} as $\mathcal{D}_{\text{down}}$.

3.2 In-distribution (ID) Study (RQ1)

Settings. The following settings are considered for comparison: $\mathcal{D}_{\text{ext}}$ ={LIBERO-Spatial} and $\mathcal{D}_{\text{ext}}$ ={LIBERO-Spatial, Object, Goal, Long}. We compare our `CapVector` with θ_{ft} (OpenVLA-OFT) and θ_{ao} (Spatial Forcing). Please note that `CapVector` is trained from θ_{meta} through standard SFT, identical to that applied to θ_{ft}.

Finding 1: `CapVector` ***inherits the efficiency and effectiveness from*** θ_{ao}**.** For ID transferring, Table 1 shows that our `CapVector` yields comparative or even higher success rates over Spatial Forcing on all training steps and all tasks. This indicates that the capability vectors are implicit representations of the extra spatial capabilities of Spatial Forcing, and simply merging parameters successfully transfers these capabilities. Additionally, with only 5k training steps, our `CapVector` achieves a substantially higher success rate than OpenVLA-OFT, despite both models being trained with regular finetuning, indicating that `CapVector` inherits the training efficiency of Spatial Forcing.

Finding 2: The orthogonal loss is critical for maintaining the capability of the capability vectors. As shown in Figure 2, while the performances of `CapVector` w/o orthogonal loss are consistently over Spatial Forcing on 5k steps, 1 epoch, and 8 epochs, it can not match Spatial Forcing on the 150k steps, which represents abundant training steps. This indicates that the pre-injected capabilities are updated and reduced during the regular finetuning process, finally resulting in capability degradation.

When the orthogonal loss is incorporated to retain the injected capabilities and constrain the model updating on the new direction, the capability degradation is largely mitigated. Table 1 shows that `CapVector` with orthogonal loss has a clear performance improvement over the baseline without it, and is still superior to Spatial Forcing with 150k training steps.

Ablation of Hyperparameters. λ is used to control the weight of the orthogonal loss in Equation (7). As shown in Figure 2, the model performs best under λ = 1e-4, which is the default setting in all other experiments. Ablation of vector weights α is shown in Section C.2.

3.3 Out-of-distribution (OOD) Study (RQ2)

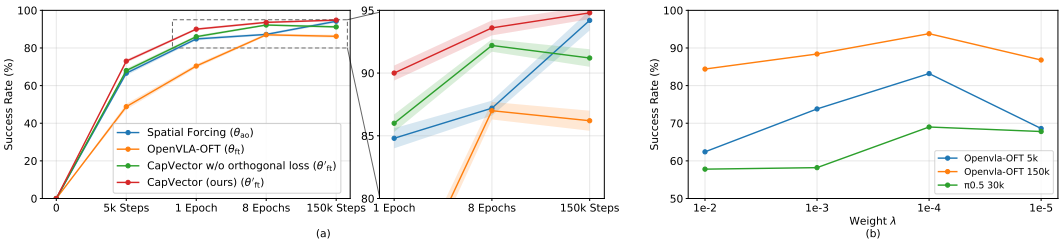

Figure 2: **(a) Success rates vs. training iterations** on LIBERO-Long for Spatial Forcing (θ_{ao}), OpenVLA-OFT (θ_{ft}), and `CapVector` (θ'_{ft}). The shadow represents the standard deviation across 3 seeds. **(b) Ablation of the orthogonal regularization weight λ.**

Table 2: **Comparison in RoboTwin 2.0 benchmark.** θ_{ft}: {OpenVLA-OFT}. θ_{ao}: {Spatial Forcing}. $\mathcal{D}_{ext}$: {LIBERO-Spatial}. $\mathcal{D}_{down}$: {RoboTwin 2.0 10 tasks with clean settings.}

Method	Turn switch	Hand-over block	Hand-over mic	Place shoe	Pick dual bottles	Place object basket	Put bottles dustbins	Place phone stand	Put object cabinet	Stack bowls two	Avg.
OpenVLA-OFT	33.0%	1.0%	4.0%	2.0%	1.0%	7.0%	1.0%	1.0%	9.0%	8.0%	6.7%
+ Spatial Forcing	**47.0%**	**23.0%**	**100.0%**	2.0%	17.0%	24.0%	15.0%	**17.0%**	**23.0%**	**63.0%**	**33.1%**
`CapVector` ($\mathcal{D}_{ext}$ = Spatial) (ours)	33.0%	1.0%	99.0%	**13.0%**	**23.0%**	**39.0%**	**22.0%**	12.0%	17.0%	59.0%	31.8%
`CapVector` ($\mathcal{D}_{ext}$ = Long)	18.0%	12.0%	99.0%	13.0%	6.0%	33.0%	4.0%	5.0%	21.0%	26.0%	23.7%
`CapVector` ($\mathcal{D}_{ext}$ = 90)	36.0%	1.0%	0.0%	2.0%	1.0%	11.0%	3.0%	0.0%	9.0%	7.0%	9.4%
$\pi_{0.5}$	3.0%	**44.0%**	9.0%	3.0%	27.0%	7.0%	12.0%	1.0%	34.0%	11.0%	15.1%
+ Spatial Forcing	**5.0%**	0.0%	**15.0%**	19.0%	29.0%	**18.0%**	**32.0%**	27.0%	72.0%	19.0%	**23.6%**
`CapVector` (Merged VLM)	4.0%	1.0%	14.0%	19.0%	27.0%	**18.0%**	**32.0%**	19.0%	**73.0%**	**22.0%**	22.9%
`CapVector` (VLM + Expert) (ours)	**5.0%**	0.0%	12.0%	**20.0%**	**32.0%**	15.0%	31.0%	**34.0%**	64.0%	**22.0%**	23.5%

Finding 3: The capability vectors can be seen as task-irrelevant, thus `CapVector` exhibits out-of-distribution transfer ability. To evaluate the transferring feasibility across domains, we conduct experiments with $\mathcal{D}_{ext}$ and $\mathcal{D}_{down}$ in different simulated environments. Table 2 shows that for different architectures, capability extraction datasets, and merging strategies, our `CapVector` realizes capability transferring to unseen distribution. Specifically, with the capability vectors γ_{ao} obtained from LIBERO, our `CapVector` always outperforms base models on most tasks of RoboTwin by a clear margin, especially improving success rates from 6.7% to 31.8% with OpenVLA-OFT as θ_{ft}. Moreover, it achieves performance comparable to Spatial Forcing. Furthermore, Figure 3 also validates the OOD transferring in the setting that $\mathcal{D}_{ext}$ is RoboTwin 2.0 and $\mathcal{D}_{down}$ is LIBERO-Long. The observed improvements in cross-domain success rates provided by capability vectors demonstrate their task-agnostic nature and capacity to facilitate generalized model performance enhancement.

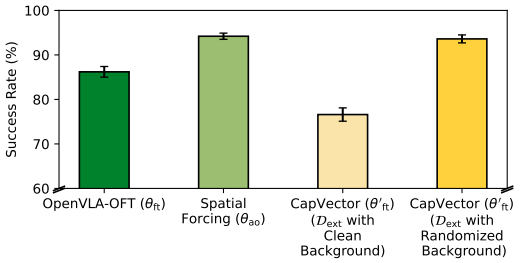

Figure 3: **Influence of the visual richness of $\mathcal{D}_{ext}$.** $\mathcal{D}_{ext}$: {RoboTwin 2.0 five capability extraction tasks}. $\mathcal{D}_{down}$: {LIBERO-Long}. We report the success rates on LIBERO-Long and consider the capability vectors from data with different visual richness. The error bars represent the standard error across 3 seeds.

3.4 Versatility Study (RQ3)

***Finding 4:* `CapVector` *demonstrates versatility across pretrained models* θ_{pt} *with different architectures and diverse auxiliary-objective SFT methods* θ_{ao}.** While the previous experiments have demonstrated the effectiveness of `CapVector` with the OpenVLA-OFT as θ_{ft} and Spatial Forcing as θ_{ao}, we further consider other θ_{ft} and θ_{ao}. Given LIBERO-Spatial as $\mathcal{D}_{ext}$, we validate the versatility of `CapVector` on two settings: (1) θ_{ft}: {StarVLA}, θ_{ao}: {LaRA-VLA}, and $\mathcal{D}_{down}$: {LIBERO}. (2) θ_{ft}: {$\pi_{0.5}$}, θ_{ao}: {Spatial Forcing}, and $\mathcal{D}_{down}$: {RoboTwin}.

Tables 1 and 3 shows that `CapVector` is effective across distinct auxiliary-objective methods, validating its capacity to extract and transfer diverse foundational capabilities. Specifically, while

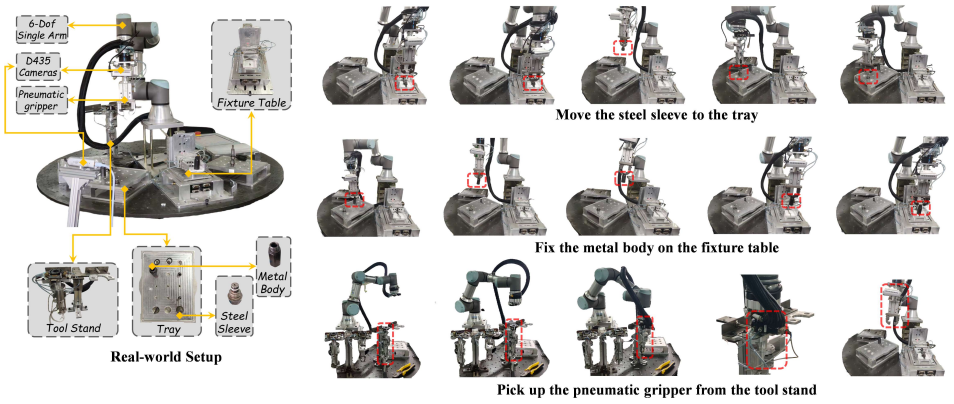

Figure 4: **Real-world setup for industrial tasks on UR3 robot.**

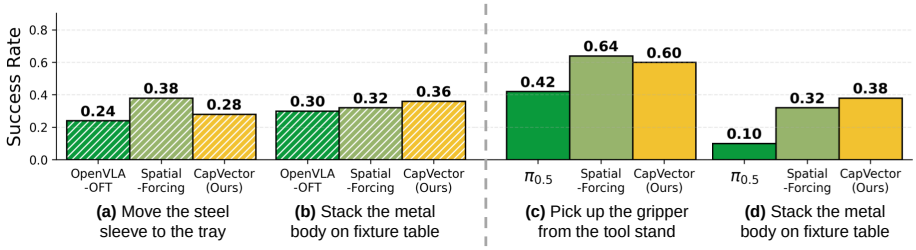

Figure 5: **Real-world experiments on industrial tasks on UR3 robots.** θ_{ft}: {$\pi_{0.5}$ and OpenVLA-OFT}. θ_{ao}: {Spatial Forcing}. $\mathcal{D}_{\text{ext}}$: {LIBERO-Spatial}. $\mathcal{D}_{\text{down}}$: {3 real-world industrial tasks}.

Table 1 highlights its success in extracting geometric comprehensions from Spatial Forcing, Table 3 proves that it can just as effectively capture the multimodal chain-of-thought reasoning abilities internalized by LaRA-VLA. When applied to the StarVLA backbone, `CapVector` achieves an impressive average success rate on LIBERO, outperforming the standard StarVLA baseline and performing comparably to the full LaRA-VLA method. These results indicate that `CapVector` can effectively transfer various capabilities while avoiding the extra computational costs associated with auxiliary SFT methods.

Table 3: **LIBERO in-distribution comparison based on LaRA-VLA.**

Method	Spatial	Object	Goal	Long	Avg.
LaRA-VLA (θ_{ao})	96.4%	**98.6%**	**99.8%**	**96.6%**	**97.9%**
StarVLA (θ_{ft})	**96.8%**	86.6%	98.2%	96.4%	94.5%
`CapVector` (ours)	96.6%	98.2%	99.2%	94.4%	97.1%

Tables 2 and 3 shows that `CapVector` is effective for θ_{pt} with both autoregressive architectures (*e.g.,* OpenVLA) and flow-matching architectures (*e.g.,* StarVLA and $\pi_{0.5}$), obtaining consistent improvements of success rates and achieving similar performance as θ_{ao}. Given that the flow-matching expert is typically initialized prior to finetuning, we evaluate two variants: one that merges only the parameters of the Vision-Language Model (VLM), and another that merges both the VLM and the action expert. Table 2 shows that both variants reach higher success rates over the regularly finetuned $\pi_{0.5}$, and merging both the parameters of VLM and the action expert yields better performance.

3.5 Real-world Study (RQ4)

Settings. We verify the effectiveness of our `CapVector` on the real-world hardware platform. Figure 4 shows that the platform serves as a flexible assembly testbed designed for industrial applications. We design a comprehensive suite of tasks that encompasses common robotic manipulation scenarios in industrial production. We collect 100 episodes per task and finetune on all tasks together. During evaluation, models are tested over 100 rollouts per task to obtain reliable success rates.

Finding 5: `CapVector` ***can realize sim-to-real transferring and is robust and general for diverse embodiments and scenes.*** To verify whether the extra spatial capabilities extracted from simulation can directly generalize to the physical world, we utilized the capability vectors merged solely from LIBERO-Spatial and applied them to our real-world training. The quantitative results shown in

Figure 5 demonstrate that `CapVector` yields substantial improvements over the standard baselines across all tasks, notably even surpassing Spatial Forcing on some tasks.

These results suggest that the spatial perception capabilities inherent in the capability vectors are environment-agnostic to a certain degree. These vectors capture fundamental geometric cues rather than overfitting to the domain-specific visual textures. The finding highlights the practical value of `CapVector`, as it enables the utilization of scalable simulation datasets to extract the transferable capabilities, while avoiding the complexity of applying extra auxiliary objectives in the real world.

To evaluate system robustness and cross-embodiment performance, we share the same capability vectors' weight with 2 external collaborators, allow them to fine-tune the θ_{meta} on 2 embodiments (ARX Lift 2 and AgileX Cobot) and 4 complex tasks, and get performance numbers from them. Figure 6 shows that our method consistently outperforms the base model to varying degrees. In particular, Figure 6 (a) requires the model to sequentially transfer four test tubes, posing a significant challenge to its long-horizon precise manipulation capability. For this task, our method improves the success rate from 0.36 to 0.53. This demonstrates that our `CapVector` has superior out-of-the-box performance and can be a general strategy to improve pretrained models and fine-tuning processes. Detailed setup of these 2 embodiments is shown in Section E.

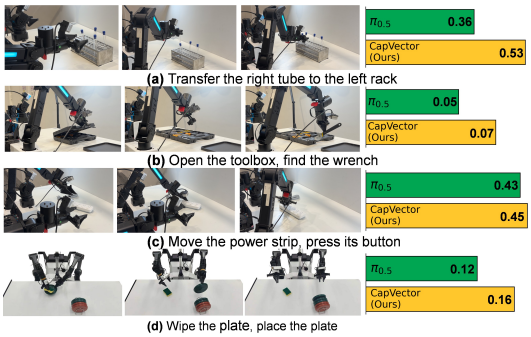

Figure 6: **Cross-embodiment deployment of capability vectors on ARX Lift 2 and AgileX Cobot out of the box.** θ_{ft}: $\{\pi_{0.5}\}$. θ_{ao}: {Spatial Forcing}. $\mathcal{D}_{\text{ext}}$: {LIBERO-Spatial}. $\mathcal{D}_{\text{down}}$: {4 external real-world tasks}.

3.6 Combining Multiple Capability Vectors (RQ5)

Finding 6: Different capability vectors can be combined additively to produce synergistic effects. Motivated by the parameter-space analysis above, we conduct an additional capability-composition experiment using the capability vectors extracted from Spatial Forcing and LaRA-VLA, as shown in Table 4. In all experiments, we use $\pi_{0.5}$ as the base model and collect 3 spatial tasks and 3 reasoning tasks in the real world, which are detailed in the Appendix B. We construct the composed model as $\theta_{\text{comp}} = \theta_{\text{base}} + \gamma_{\text{SF}} + \gamma_{\text{LaRA}}$.

Table 4: Success rates (%) of models with different capability vectors. θ_{ft}: $\{\pi_{0.5}\}$. $\mathcal{D}_{\text{ext}}$: {LIBERO-Spatial}. $\mathcal{D}_{\text{down}}$: {6 real-world spatial and reasoning tasks}. All results are averaged over three random seeds, with standard deviations reported.

Model	Spatial tasks	Reasoning tasks	Average
$\pi_{0.5}$	61.2 ± 0.7	58.4 ± 0.8	59.8 ± 0.6
`CapVector` (γ_{SF})	68.8 ± 0.5	59.9 ± 0.7	64.4 ± 0.5
`CapVector` (γ_{LaRA})	63.0 ± 0.6	65.5 ± 0.6	64.3 ± 0.5
`CapVector` ($\gamma_{\text{SF}} + \gamma_{\text{LaRA}}$)	$\mathbf{70.0 \pm 0.5}$	$\mathbf{66.7 \pm 0.5}$	$\mathbf{68.4 \pm 0.4}$

The individual vectors exhibit clear capability-specific effects. On spatially demanding tasks, injecting γ_{SF} capability vectors improves the average success rate by 7.6%, whereas γ_{LaRA} provides a smaller improvement of 1.8%. Conversely, on reasoning-intensive tasks, γ_{LaRA} improves performance by 7.1%, while γ_{SF} provides an improvement of only 1.5%. More importantly, the composed model preserves and further strengthens the benefits of both individual capability vectors. Compared with the base model, joint injection improves spatial performance by 8.8% and reasoning performance by 8.3%. It therefore achieves the highest average success rate of 68.4%, outperforming the γ_{SF} and γ_{LaRA} variants by 4.0% and 4.1%, respectively.

The additional parameter-space analysis in Appendix G shows that two capability vectors exhibit low cosine similarity, indicating largely distinct update directions. Meanwhile, γ_{SF} primarily introduces updates in perception-related intermediate Transformer blocks, whereas γ_{LaRA} places greater update intensity in deeper blocks associated with multimodal reasoning. This structural separation provides a plausible explanation for why the two capabilities can be composed with limited interference. Overall, these results demonstrate that capability vectors are not restricted to isolated single-capability transfer. Combining distinct capability vectors can yield synergistic rather than conflicting interactions.

3.7 Efficiency Analysis (RQ6)

Finding 7: `CapVector` ***improves training efficiency by reusing capability vectors from a single extraction across multiple downstream finetuning tasks.*** Although `CapVector` improves downstream finetuning efficiency, its overall efficiency also depends on the one-time capability extraction cost and how this cost is amortized across downstream tasks. To quantify this trade-off, we normalize the cost of regular SFT on one downstream task to 1; auxiliary-objective SFT costs approximately 2 due to additional modules and forward passes. Assuming that training on $\mathcal{D}_{\text{ext}}$ costs the same as training on one downstream task, extraction requires

$$\mathcal{C}_{\text{ext}} = \mathcal{C}(\theta_{\text{ft}}) + \mathcal{C}(\theta_{\text{ao}}) = 1 + 2 = 3. \tag{8}$$

The extraction is reusable, and each downstream task requires only regular SFT. Thus, for N tasks:

$$\mathcal{C}_{\text{CapVector}}(N) = 3 + N, \qquad \mathcal{C}_{\text{aux}}(N) = 2N. \tag{9}$$

Under this estimate, `CapVector` breaks even at three tasks and reduces total training cost thereafter. Its average cost per task, $1 + 3/N$, approaches that of regular SFT as N increases. The above experiments demonstrate this reuse: a single meta model enhanced using capability vectors extracted from LIBERO-Spatial benefits LIBERO-Object, Goal, Long, RoboTwin 2.0, and real-world manipulation.

4 Related Work

SFT Strategies for VLAs. Recent works have increasingly focused on advancing generalist VLAs (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2024; 2025; Wen et al., 2025; Bjorck et al., 2025) trained on large-scale robot data (O'Neill et al., 2024; Khazatsky et al., 2024) based on VLMs (Beyer et al., 2024; Yang et al., 2025). However, standard SFT on these foundation models is often insufficient for the model to quickly achieve high performance on new tasks. Therefore, many auxiliary-objective SFT strategies are proposed. OpenVLA-OFT (Kim et al., 2025a) improves performance and inference efficiency via optimized action decoding strategies and learning objectives, while FLARE (Zheng et al., 2025) introduces future latent representation alignment for implicit world modeling. Spatial Forcing (Li et al., 2026) enhances spatial abilities by imposing spatial alignment constraints, and LaRA-VLA (Bai et al., 2026) and LaST_0 (Liu et al., 2026) internalize multi-modal CoT into continuous latents for efficient reasoning. Although finetuning with auxiliary objectives provides enhanced performance and efficiency, they often introduce additional forward passes. In this paper, we avoid the overhead through capability extraction and model merging.

Model Merging. Model merging combines parameters of two distinct models to reuse knowledge and improve robustness. Prior work in LLMs and VLMs (Wang et al., 2024; Yadav et al., 2023; 2024; Nasery et al., 2025; Lu et al., 2025; Jang et al., 2024) shows that simple weight interpolation or averaging can effectively combine task-specific skills and mitigate distribution shifts. O-LoRA (Wang et al., 2023) further mitigates catastrophic forgetting by orthogonal low-rank updates. However, extending model merging to VLA models has received limited attention. ReVLA (Dey et al., 2025) alleviates visual catastrophic forgetting via vision backbone reversal. RETAIN (Yadav et al., 2025) enables robust adaptation in low-data regimes by interpolating pretrained and finetuned VLA checkpoints. MergeVLA (Fu et al., 2025) further proposes a cross-skill composition strategy. Unlike simple model ensembling, we extract capability vectors via arithmetic operations on model parameters, followed by model merging to achieve capability transfer across models.

5 Conclusion

We introduce the concept of capability vectors to represent the gains in general capabilities acquired during finetuning, and propose a pipeline that integrates these vectors. By merging capability vectors with pretrained models, we construct a capability-enhanced meta model. `CapVector` achieves the simplicity and computational efficiency of standard SFT with the high performance of auxiliary-objective SFT. Through a series of experiments, we systematically analyze the effectiveness, versatility, and mechanism of capability vectors. Finally, real-world experiments validate the practical applicability and generalization. Overall, this work provides a novel strategy for extracting and transferring gains from finetuning into pretrained models, offering a feasible solution for more efficient and broadly applicable VLA training.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Shuanghao Bai, Jing Lyu, Wanqi Zhou, Zhe Li, Dakai Wang, Lei Xing, Xiaoguang Zhao, Pengwei Wang, Zhongyuan Wang, Cheng Chi, et al. Latent reasoning vla: Latent thinking and prediction for vision-language-action models. *arXiv preprint arXiv:2602.01166*, 2026.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

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.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025.

Kerim Büyükakyüz. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.

Shiqi Chen, Jinghan Zhang, Tongyao Zhu, Wei Liu, Siyang Gao, Miao Xiong, Manling Li, and Junxian He. Bring reason to vision: Understanding perception and reasoning through model merging. In *Forty-second International Conference on Machine Learning*, 2025a.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025b.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025c.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

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

Yuxia Fu, Zhizhen Zhang, Yuqi Zhang, Zijian Wang, Zi Huang, and Yadan Luo. Mergevla: Cross-skill model merging toward a generalist vision-language-action agent. *arXiv preprint arXiv:2511.18810*, 2025.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. Model stock: All we need is just a few fine-tuned models. In *European Conference on Computer Vision*, pp. 207–223. Springer, 2024.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning*, pp. 2679–2713. PMLR, 2025b.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

Fuhao Li, Wenxuan Song, Han Zhao, Jingbo Wang, Pengxiang Ding, Donglin Wang, Long Zeng, and Haoang Li. Spatial forcing: Implicit spatial representation alignment for vision-language-action model. 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

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. *arXiv preprint arXiv:2410.07864*, 2024.

Zhuoyang Liu, Jiaming Liu, Hao Chen, Jiale Yu, Ziyu Guo, Chengkai Hou, Chenyang Gu, Xiangju Mi, Renrui Zhang, Kun Wu, et al. Last _{0}: Latent spatio-temporal chain-of-thought for robotic vision-language-action model. *arXiv preprint arXiv:2601.05248*, 2026.

Wei Lu, Rachel K Luu, and Markus J Buehler. Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities. *npj Computational Materials*, 11(1):84, 2025.

Anshul Nasery, Jonathan Hayase, Pang Wei Koh, and Sewoong Oh. Pleas-merging models with permutations and least squares. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 30493–30502, 2025.

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.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Wenxuan Song, Ziyang Zhou, Han Zhao, Jiayi Chen, Pengxiang Ding, Haodong Yan, Yuxin Huang, Feilong Tang, Donglin Wang, and Haoang Li. Reconvla: Reconstructive vision-language-action model as effective robot perceiver. *arXiv preprint arXiv:2508.10333*, 2025.

Ke Wang, Nikolaos Dimitriadis, Guillermo Ortiz-Jiménez, François Fleuret, and Pascal Frossard. Localizing task information for improved model merging and compression. In *ICML*, 2024.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, 2023.

Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. Diffusionvla: Scaling robot foundation models via unified diffusion and autoregression. In *Forty-second International Conference on Machine Learning*, 2025.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023.

Prateek Yadav, Tu Vu, Jonathan Lai, Alexandra Chronopoulou, Manaal Faruqui, Mohit Bansal, and Tsendsuren Munkhdalai. What matters for model merging at scale? *arXiv preprint arXiv:2410.03617*, 2024.

Yajat Yadav, Zhiyuan Zhou, Andrew Wagenmaker, Karl Pertsch, and Sergey Levine. Robust finetuning of vision-language-action robot policies via parameter merging. *arXiv preprint arXiv:2512.08333*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, et al. Flare: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

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

A ABSTRACT

In this supplementary material, we provide the following supplementary information:

- Section B describes the spatial and reasoning tasks used to evaluate capability composition.
- Section C presents the comparisons with TIES-style merging methods and the ablations on the merging weight.
- Section D analyzes the computational and memory overhead of orthogonal regularization and auxiliary-objective finetuning.
- Section E details real-world data collection, robot platforms, and task setups.
- Section F discusses limitations and future directions.
- Section G analyzes the parameter-space structure and functional specificity of capability vectors.
- Section H introduces the baseline models and auxiliary-objective finetuning methods.

B TASK DETAILS FOR EVALUATING CAPABILITY COMPOSITION

To evaluate the individual effects of capability vectors and their composition, we construct three spatial tasks and three reasoning tasks. The spatial tasks emphasize spatial relation understanding and geometric alignment, whereas the reasoning tasks emphasize conditional decision-making, attribute relations, and action dependencies. This categorization reflects the primary demands of each task rather than a strict separation between spatial and reasoning capabilities.

Spatial Tasks. (1) Spatial Relation Placement. The robot follows an instruction to place a red block between a blue block and a yellow block. The positions of the two reference blocks vary across trials to evaluate whether the model can interpret relative spatial relations in the observed scene and determine an appropriate placement location.

(2) Orientation-Constrained Insertion. The robot grasps a rectangular block, adjusts its position and orientation, and inserts it into a matching rectangular slot. The initial block pose and slot orientation vary across trials to evaluate relative-pose understanding and precise alignment.

(3) Constrained Placement. The robot places a target object into a designated region surrounded by obstacles while avoiding contact with or displacement of the surrounding obstacles. Obstacle layouts vary across trials to evaluate the model's understanding of available space and geometric constraints.

Reasoning Tasks. (1) Conditional Sorting. The robot places a block into the appropriate tray according to the cue card's color and a conditional rule specified in the instruction. The tray assignments for blue and yellow cue cards vary across instructions to prevent reliance on a fixed color-to-location mapping. This task evaluates whether the model can combine visual observations with the current rule to make the correct decision.

(2) Shared-Attribute Inference. The robot observes two reference objects, identifies their shared color or shape, and places the candidate object with the same attribute into a tray. Exactly one candidate satisfies the requirement in each trial. This task evaluates whether the model can infer attribute relations between objects and use them to select the correct target.

(3) Prerequisite Action Planning. The robot retrieves an object from a container in response to the instruction to place the target object into a tray. When a transparent cover blocks access to the container, the robot must first remove it; when the container is open, the robot can grasp the object directly. This task evaluates whether the model can identify action dependencies from the scene state and execute the necessary action sequence.

All model variants are evaluated using the same task configurations and evaluation conditions. Spatial and reasoning success rates are computed by averaging over the three tasks in each category, and overall success is averaged over all six tasks. All results are reported as the mean and standard deviation across three random seeds.

C MORE EXPERIMENTS

We conduct more extensive experiments to investigate the following: **(RQ7)** How does our feature extraction and model merging approach compare with other model merging methods? **(RQ8)** How to choose the merging weight α?

C.1 COMPARISON WITH A TIES-STYLE MERGING OPERATOR (RQ7)

To examine how a standard merging operator compares with training-time capability preservation, we evaluate a TIES-style merging baseline (Yadav et al., 2023) adapted to `CapVector`. Unlike the direct addition used by `CapVector` without orthogonal loss, this baseline applies magnitude trimming, coordinate-wise sign election, and aggregation of sign-consistent updates to the capability vector and downstream-task update. Since $\gamma_{\text{ao}} = \theta_{\text{ao}} - \theta_{\text{ft}}$ is a difference between two finetuned models rather than a conventional task vector relative to the pretrained checkpoint, we refer to this variant as TIES-style merging adapted to `CapVector`.

Table 5: **Comparison with a TIES-style merging operator on LIBERO.** We report success rates (%). The TIES-style variant adapts TIES-Merging (Yadav et al., 2023) to `CapVector`. Bold denotes the highest value in each column, including ties.

Method	Spatial (%)	Object (%)	Goal (%)	Long (%)	Average (%)
OpenVLA-OFT (θ_{ft})	96.8	94.8	92.8	86.2	92.7
`CapVector` w/o orthogonal loss	97.4	**99.0**	**97.2**	91.2	96.2
`CapVector` + TIES-style merging	97.8	**99.0**	97.0	92.6	96.6
`CapVector` (ours)	**98.4**	98.4	96.8	**94.8**	**97.1**

As shown in Table 5, TIES-style merging improves the average success rate from 96.2% to 96.6% over `CapVector` without orthogonal loss. However, our complete `CapVector` achieves a higher average success rate of 97.1%, with a larger advantage on LIBERO-Long (94.8% versus 92.6%). These results indicate that our improvement does not arise solely from additive model merging. While TIES resolves coordinate-wise conflicts after training, our orthogonal regularization reduces interference between the downstream update and the capability vector during optimization.

C.2 MORE ABLATIONS (RQ8)

α is utilized to control the merging weight of the capability vectors during the model merging phase, as defined in Equation (3). This weight dictates the degree to which the extracted general capabilities γ_{ao} are integrated into the pretrained backbone θ_{pt} to construct the capability-enhanced meta-model θ_{meta}. As shown in Table 6, the model achieves its optimal performance when α = 1.1, which is chosen as the default setting in all other experiments.

Table 6: **Ablation study for the merging weight α of the capability vectors.** We report the Success rates (%). $\mathcal{D}_{\text{ext}}$: {LIBERO-Spatial}. $\mathcal{D}_{\text{down}}$: {LIBERO-Long}.

VLA Backbone	Weight α					
	0.5	0.7	0.9	1.1	1.3	1.5
OpenVLA-OFT	91.2	91.8	92.8	**94.8**	90.6	92.4

D ANALYSIS OF TRAINING OVERHEAD

D.1 EXTRA OVERHEAD INDUCED BY ORTHOGONAL LOSS

To maintain the capability of the capability vectors during downstream finetuning, we incorporate the orthogonal loss, as illustrated in Equation (6). We evaluate its computational overhead by

Table 7: **Extra overhead of orthogonal loss.** We report the training computational cost (FLOPs) and GPU memory based on the LoRA tuning of OpenVLA-OFT.

Orthogonal Loss	FLOPs	GPU Memory
✘	17.9T	62.8G
✔	17.9T + 0.3G	62.8G + 0.5G

measuring the computational FLOPs and GPU memory usage, as summarized in Table 7. The orthogonal loss introduces a negligible computational overhead, increasing total training FLOPs by merely **0.3G (<0.002%)** and GPU memory usage by approximately **0.5G (<0.8%)**. Consequently, `CapVector` achieves the superior performance improvement through our orthogonal loss while maintaining simplicity and low resource consumption.

D.2 Extra Overhead Comparison between Auxiliary-objective SFT Methods and Ours

Table 8: **Extra overhead of Auxiliary-objective SFT.** We report the training computational cost (FLOPs) and GPU memory based on the LoRA tuning of OpenVLA-OFT. We take Spatial Forcing as the example of auxiliary-objective SFT methods.

Method	FLOPs	GPU Memory
OpenVLA-OFT	17.9T	62.8G
+ Spatial Forcing	17.9T + 5.0T	62.8G + 10.9G
+ `CapVector` (Ours)	17.9T + 0.3G	62.8G + 0.5G

As summarized in Table 8, the auxiliary-objective SFT method (*e.g.*, Spatial Forcing) incurs substantial computational costs because it introduces additional forward passes and auxiliary modules to align the targets. Specifically, adding Spatial Forcing to the OpenVLA-OFT baseline increases training FLOPs by **5.0T (28%)** and GPU memory usage by **10.9G (17%)**. By contrast, our `CapVector` introduces a negligible overhead while achieving performance comparable to, or even exceeding, full auxiliary-objective finetuned methods across diverse tasks.

E Real-world Setup.

Data Collection. During the data collection phase, we first manually designate a diverse set of intermediate waypoints for each task. Subsequently, the robot autonomously generated a wide variety of trajectories by randomly selecting from these waypoints. Finally, we collected a total of 100 episodes per task.

Robot and Task Setup. In Section 3.5, we deploy our models across three distinct hardware platforms to evaluate both specific real-world performance and cross-embodiment generalization. Here we show those two external collaborators' real-world robot and task setups in Figure 7.

- ARX Lift 2: This platform utilizes a 6-DoF dual-arm system with D405 cameras. The task workspace is configured with objects like a test tube rack, test tubes, a toolbox, a tray, and a power strip to evaluate complex, multi-stage manipulations.
- AgileX Cobot: This setup consists of a teleoperation-ready system featuring master and puppet 6-DoF dual arms. The workspace includes a dish rack, plate, and sponge, designed to evaluate everyday dexterous tasks.

F Limitations.

Overall, post-training for robot foundation models can be broadly categorized into two paradigms: supervised fine-tuning (SFT) and reinforcement learning (RL). In this work, we primarily focus on

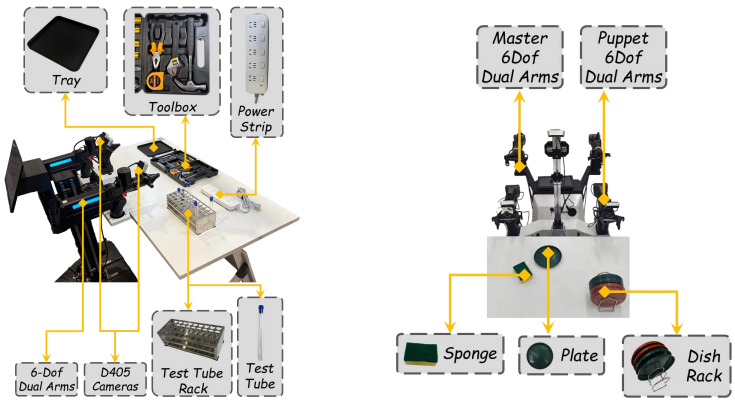

Figure 7: **Real-world setup of the cross-embodiment deployment tasks on ARX Lift2 and Agilex Cobot robots.**

capability vectors obtained through SFT, while leaving the acquisition of capability vectors in RL settings to future work.

G Structural and Functional Analysis of Capability Vectors

We conduct parameter-level and functional analyses of the capability vectors extracted from Spatial Forcing and LaRA-VLA to examine their structural characteristics in the parameter space. For these analyses, we use StarVLA as the base model and collect three spatial tasks and three reasoning tasks in the real world, as mentioned in Section B.

G.1 Parameter-Space Distinguishability

We first examine whether capability vectors extracted from different auxiliary objectives correspond to distinguishable parameter directions. Specifically, we compute the cosine similarity between the capability vector extracted from Spatial Forcing, γ_{SF}, and that extracted from LaRA-VLA, γ_{LaRA}:

$$\mathrm{CosSim}\left(\gamma_{\mathrm{SF}}, \gamma_{\mathrm{LaRA}}\right) = \frac{\langle \gamma_{\mathrm{SF}}, \gamma_{\mathrm{LaRA}} \rangle}{\|\gamma_{\mathrm{SF}}\|_2 \, \|\gamma_{\mathrm{LaRA}}\|_2}. \tag{10}$$

The resulting cosine similarity is only 0.067, indicating that the spatial-perception capability extracted from Spatial Forcing and the multimodal-reasoning capability extracted from LaRA-VLA correspond to substantially different directions in the parameter space.

G.2 Layer-Wise and Module-Wise Parameter Distributions

We further analyze where the capability vectors are concentrated in the model. Because different layers and modules contain different numbers of parameters, directly comparing their vector norms may bias the analysis toward larger modules. We therefore calculate the parameter-count-normalized magnitude for module or layer l as

$$r_l(\gamma) = \frac{\|\gamma_l\|_2 / \sqrt{n_l}}{\sum_j \|\gamma_j\|_2 / \sqrt{n_j}}, \tag{11}$$

where γ_l denotes the capability-vector component associated with module or layer l, and n_l denotes the number of parameters in that module or layer.

The two capability vectors exhibit clearly different distribution patterns. For Spatial Forcing, 57.6% of the normalized vector magnitude is concentrated in intermediate Transformer blocks. In comparison, the LaRA-VLA capability vector places 60.9% of its normalized magnitude in the deeper Transformer blocks. This is consistent with findings in VLM research: early-to-middle layers focus on local visual perception, while later layers focus on information fusion and reasoning (Chen et al., 2025a).

G.3 Functional Specificity of Different Capability Vectors

In addition to parameter-level analysis, we examine whether the differences in representational structures between the vectors correspond to different downstream behaviors. Under the same merging coefficient and downstream finetuning configuration, we independently transfer the Spatial Forcing and LaRA-VLA capability vectors to tasks emphasizing spatial perception and long-horizon multimodal reasoning.

On spatial tasks, transferring the Spatial Forcing vector improves the average success rate by 7.6 ± 0.5 percentage points, whereas transferring the LaRA-VLA vector produces an improvement of only 1.8 ± 0.6 percentage points. In contrast, on reasoning tasks, the LaRA-VLA vector improves performance by 7.1 ± 0.6 percentage points, while the Spatial Forcing vector provides an improvement of only 1.5 ± 0.7 percentage points.

These capability-specific transfer results indicate that the two vectors do not merely represent different parameter directions. Instead, their parameter-space differences are associated with distinguishable functional effects: the Spatial Forcing vector predominantly transfers spatial-perception capability, whereas the LaRA-VLA vector predominantly transfers multimodal and long-horizon reasoning capability.

Together, these analyses provide structural and functional evidence that the capability vectors extracted from Spatial Forcing and LaRA-VLA are not only distinguishable in the parameter space, but also systematically associated with their respective spatial-perception and multimodal-reasoning capabilities.

H Baselines.

OpenVLA-OFT (Kim et al., 2025a). OpenVLA-OFT proposes an Optimized finetuning (OFT) strategy aimed at improving both the performance and inference efficiency of vision-language-action (VLA) models when adapted to specific robotic tasks. This approach combines parallel decoding, action chunking, continuous action representations, and a straightforward L1 regression loss to accelerate inference. For tasks demanding fine-grained language comprehension, OFT is enhanced with FiLM (Feature-wise Linear Modulation) (Perez et al., 2018) to reinforce language grounding. On the LIBERO simulation benchmark (Liu et al., 2023), OpenVLA-OFT achieves a 97.1% success rate while increasing action generation speed by 26 times. In real-world experiments with a bimanual ALOHA robot (Zhao et al., 2024), it surpasses strong finetuned VLAs such as π_0 (Black et al., 2024) and RDT-1B (Liu et al., 2024), as well as policies trained from scratch, achieving up to a 15% absolute improvement in average success rate on dexterous manipulation tasks.

$\pi_{0.5}$ (Black et al., 2025). $\pi_{0.5}$ is a vision-language-action (VLA) model designed to achieve open-world generalization for real-world robotic manipulation. Building on π_0, $\pi_{0.5}$ adopts a co-training framework that leverages highly heterogeneous data sources, including demonstrations from multiple robot platforms, high-level semantic prediction tasks, web-scale multimodal data, and language supervision. The model follows a hierarchical architecture that first infers high-level semantic subtasks and subsequently predicts low-level action chunks, enabling long-horizon and multi-stage manipulation. Despite relying predominantly on non-mobile-manipulation data during training, $\pi_{0.5}$ demonstrates strong generalization to unseen homes and complex real-world tasks, such as household cleaning and dexterous object rearrangement.

StarVLA (Community, 2026). StarVLA introduces a modular, Lego-like unified framework to address the severe fragmentation across architectures, codebases, and evaluation protocols in VLA research. It implements a shared "backbone-action-head" abstraction that allows researchers to seamlessly interchange diverse vision-language or world-model backbones (*e.g.*, Qwen-VL (Yang et al., 2025) and Cosmos (Kim et al., 2026)) with four representative action-decoding paradigms (*e.g.*, autoregressive tokenization, flow-matching). This unified architecture is further strengthened by reusable, paradigm-agnostic training strategies for multimodal co-training and a standardized server-client interface for cross-benchmark evaluation. Across major benchmarks including LIBERO, SimplerEnv, and RoboTwin 2.0, StarVLA provides fully reproducible single-benchmark training recipes that match or surpass prior state-of-the-art methods, significantly lowering the barrier for principled comparison and practical real-robot deployment.

Spatial Forcing (Li et al., 2026). Spatial Forcing (SF) introduces a straightforward yet effective alignment strategy to enhance spatial reasoning in VLA models. SF implicitly guides VLAs to acquire 3D spatial comprehension by aligning intermediate visual embeddings with geometric representations extracted from pretrained 3D foundation models. This alignment improves action precision without requiring explicit 3D sensor inputs or depth estimators. On the LIBERO (Liu et al., 2023) and RoboTwin (Chen et al., 2025c) benchmarks, SF surpasses strong 2D- and 3D-based VLA baselines, achieving state-of-the-art performance while accelerating training by up to 3.8× and demonstrating improved data efficiency across diverse robotic tasks.

LaRA-VLA (Bai et al., 2026). Latent Reasoning VLA (LaRA-VLA) introduces a unified framework that internalizes multi-modal chain-of-thought (CoT) reasoning into continuous latent representations for embodied action. LaRA-VLA implicitly guides the model to reason in latent space via a curriculum-based training paradigm, progressively transitioning from explicit textual and visual CoT supervision to pure latent reasoning. This alignment resolves the representational mismatch between discrete reasoning tokens and continuous control, effectively eliminating explicit CoT generation overhead during inference. On LIBERO (Liu et al., 2023) simulation benchmarks and long-horizon real-robot manipulation tasks, LaRA-VLA consistently outperforms state-of-the-art VLA baselines, achieving superior performance while reducing inference latency by up to 90% for efficient real-time control.