NestedVLA: Learning to Consolidate and Generate Skills for Vision-Language-Action Models

Guodong Du **Wanyu Lin***
Department of Computing, The Hong Kong Polytechnic University
guodong.du@connect.polyu.hk wan-yu.lin@polyu.edu.hk

Abstract

Vision–language–action (VLA) models can learn a wide range of robotic skills, but combining separately trained skills remains challenging. The original training data may be unavailable for joint retraining, while directly merging model weights can cause interference. We propose **NestedVLA**, which consolidates independently fine-tuned VLA policies without access to their original training data. Instead of merging all updates into one checkpoint, a hypernetwork embedded in the VLA generates action-expert weight updates for a selected task. Each task's compressed vision–language model update is stored separately, and both updates are added to a frozen base VLA to instantiate the corresponding policy. We evaluate NestedVLA by consolidating three policies fine-tuned with reinforcement learning on MetaWorld, ManiSkill, and CALVIN. It achieves 66.7% average success, improving on the strongest model-merging baseline by 41.1 percentage points. It performs within 1 percentage point of per-task fine-tuning on ManiSkill and substantially narrows the gap on CALVIN. In this three-policy setting, total storage is $1.5\times$ that of a single full checkpoint. Code is available at `https://github.com/WanyuGroup/NeurIPS2026-NestedVLA`.

1 Introduction

Vision-language-action (VLA) models combine visual and language understanding with embodied control [2, 74, 27, 1, 43]. Independently fine-tuning a common backbone produces policies specialized for different robotic benchmarks, but storing one full checkpoint per benchmark duplicates most parameters. Joint multi-task training can avoid this duplication when data from all benchmarks can be pooled. We instead study post-hoc consolidation of policies independently fine-tuned from the same backbone. We seek to consolidate their task-specific parameter updates into a compact representation without pooling the original training data.

A natural post-hoc approach is to merge the parameter updates of the fine-tuned policies into a single checkpoint. In VLA models, however, independently learned updates can interfere strongly when combined: in our experiments, simple averaging loses most of the performance of the separate policies. Our weight-space analysis finds weak alignment across major parameter groups, an incompatibility that is particularly consequential in the action expert because it directly governs control (Appendix E). This observation is consistent with MergeVLA [17], which likewise identifies dependencies distributed across action-expert layers as a major obstacle to cross-skill merging. These findings motivate a consolidation strategy that preserves policy-specific parameter structure rather than collapsing all updates into one set of weights.

We propose **NestedVLA**, a framework that shifts the paradigm from parameter aggregation to parameter generation. A hypernetwork embedded in the VLA generates sparse action-expert updates

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

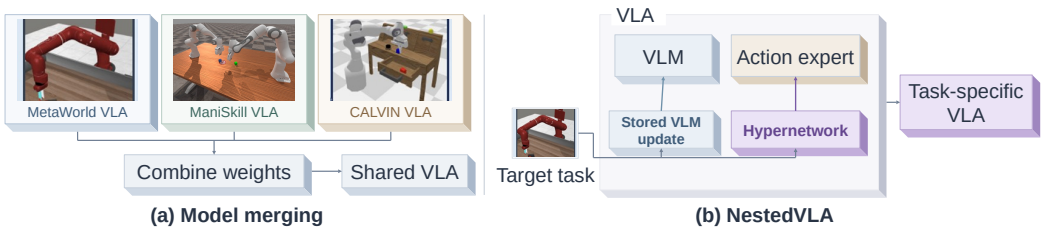


Figure 1: **Motivation and overview of NestedVLA.** (a) Model merging combines independently fine-tuned VLA weights into a shared model, where incompatible updates can impair the source tasks. (b) NestedVLA retains a compressed task-specific VLM update and uses an embedded task-conditioned hypernetwork to generate the action-expert update. Both updates are applied to a shared base to materialize the selected VLA.

conditioned on the target task, while compressed task-specific VLM residuals are retained separately. The selected updates compose with a shared base to instantiate a task-specific VLA without changing its forward architecture or introducing routing supervision. We evaluate NestedVLA on MetaWorld [38], ManiSkill [48], and CALVIN [39] using the open-source $\pi_{0.5}$ [43] backbone. NestedVLA outperforms the strongest model-merging baseline by 41 pp on average, matches per-task fine-tuning within 1 pp on ManiSkill, and substantially closes the gap on CALVIN.

2 Related Work

Vision-Language-Action models. Recent VLAs cast robotic control as instruction-following on top of pretrained vision-language backbones [2, 74, 27, 25, 1, 43]. OpenVLA [27] discretizes actions into rare vocabulary tokens for autoregressive prediction; π_0 [1] pairs the VLM with a separate *action expert* trained via flow matching for continuous action chunks; $\pi_{0.5}$ [43] extends this design to interleave sub-task narration and low-level control. Lighter variants such as VLA-Adapter [54] preserve multi-task competence at sub-billion-parameter scales. VLAct uses representation-centric continued pre-training to preserve VLM priors and promote shared action semantics across embodiments [66]. Across this family, multi-task ability is learned from pooled data, requiring co-located datasets and predefined skills. We target the regime where these assumptions break: skills are acquired *independently* and consolidated *after the fact*.

Parameter-efficient visual adaptation. Prompt- and adapter-based methods offer lightweight alternatives to full fine-tuning of vision backbones. Recent work makes prompts instance-aware, injects structured visual semantics, or searches for layer-specific prompt fusion rules [60–62]. Structured low-rank adaptation further filters task-irrelevant directions and coordinates updates across layers [64], while visual alignment methods address over-reliance on language priors in multimodal models [63]. Meng et al. [41] propose AHRA, a hierarchical shared–private expert framework that aligns semantic levels while preserving modality-specific evidence. Such methods adapt or organize multimodal representations; NestedVLA instead starts from independently trained VLA policies and consolidates their parameter updates post hoc.

Hypernetworks. Using one network to predict the parameters of another dates back to HyperNetworks [21], with later applications to continual learning [49] and multi-task NLP [26]. Recent parameter generators use latent diffusion to synthesize neural-network weights [51], while graph-relational hypernetworks generate personalized client models by exploiting relations among heterogeneous clients [34]. Other parameter generators scale this process to full models with recurrent generation [52], map task prompts directly to LoRA weights [33], or produce instance-conditioned weight updates as functional memory [59]. NestedVLA inherits this task-conditioned generation premise, but uses patch-based tokenization to target a sparse top-K delta on a frozen billion-scale VLA backbone for post-hoc RL-policy consolidation.

Reusable task updates. Related work on language models compresses full-model task updates into modular SkillPacks for heterogeneous knowledge fusion and multi-domain specialization [14, 32]. HoE instead extracts LoRA-form experts from task vectors and routes them for multi-

objective alignment [31]. These studies concern text-only language models and retain or select task modules. NestedVLA studies closed-loop VLA policies and generates the action-expert update with an embedded hypernetwork. Complementary LLM studies optimize resource-aware subtask routing or analyze how post-training objectives reshape confidence during reasoning [7, 30]; these studies address deployment and reasoning behavior rather than parameter consolidation.

Model merging. Model merging combines independently trained checkpoints into a single deployable model. Recent surveys organize model fusion across parameter, representation, and behavior levels [3, 72, 73]. **Static merging** produces one fixed delta via weighted averaging [58, 37, 23, 12], conflict-handling [65, 68, 5, 11, 15], or low-rank sub-spaces [18, 47, 36, 35, 13], with adapter counterparts [69, 71]. **Dynamic merging** retains per-task masks, scalars, residuals, or frequency-domain corrections at inference [50, 22, 70, 28, 8]. Recent adaptive methods build reusable component-level expert libraries for input-aware recombination [45] or evolve experts with manifold-aware, training-free routing for continual merging [46]. Across both regimes the operative verb is *select*: every output value is an interpolation, mask, or projection of values already present in $\{\theta_m - \theta_0\}_m$. Complementary studies make expert training merge-aware [20], calibrate or quantize post-merge models [19, 53], and analyze scaling laws or expert-read budgets for weight-space merging [56, 55]. NestedVLA *generates* the per-skill delta from a single shared hypernetwork instead. Embodied applications of merging remain largely unexplored; ReVLA [6] mitigates single-skill domain forgetting rather than multi-skill consolidation.

3 NestedVLA

3.1 Problem Formulation

We consider M task-specific datasets $\mathfrak{D} = \{\mathcal{D}_m\}_{m=1}^{M}$, each corresponding to a manipulation task in a benchmark-specific setting. Fine-tuning a pretrained VLA backbone Θ_0 on each $\mathcal{D}_m$ yields task-specific parameters $\{\Theta_m\}_{m=1}^{M}$ and the corresponding *task vector* [23]

$$\tau_m \;=\; \Theta_m - \Theta_0, \tag{1}$$

which captures all parameter changes induced by task m. We treat each τ_m as the parametric encoding of a *skill*, a reusable manipulation capability acquired in isolation. The goal of skill consolidation is to construct a unified model that leverages the entire skill set $\{\tau_m\}_{m=1}^{M}$ across diverse downstream scenarios without storing full per-task checkpoints or using expert routing.

A common approach is to merge $\{\tau_m\}$ into a single static delta [23, 65, 58, 37, 5]. However, parameter aggregation in the VLA setting suffers from severe inter-task conflicts (Sec. 1), as different tasks impose contradictory updates on the same parameters. We instead propose to *generate* parameters. Write $\Theta_m = (\Theta_m^V, \Theta_m^A)$ for the VLM and action-expert parameters. In the evaluated v1 system, a generator $\mathcal{G}_\phi$, conditioned on task identifier m, produces the action-expert delta, while a compressed VLM delta z_m^V is retained for the same task:

$$\hat{\tau}_m^A = \mathcal{G}_\phi(m), \qquad \hat{\Theta}_m^A = \Theta_0^A + \hat{\tau}_m^A, \qquad \hat{\Theta}_m^V = \Theta_0^V + z_m^V. \tag{2}$$

Both updates compose with the frozen base at inference. This shifts the action-side problem from *parameter aggregation* to *parameter generation*, avoiding full per-task checkpoints and a single conflicting merge.

3.2 Overview

NestedVLA realizes the generator $\mathcal{G}_\phi$ as a hypernetwork *nested* inside the VLA pipeline: it shares the backbone's parameter layout as its output space, is trained against targets derived from the backbone's task vectors, and is queried jointly with the backbone at inference. The framework operates in three stages (Fig. 2):

1. **Skill Acquisition** (§3.3): fine-tune Θ_0 on each $\mathcal{D}_m$, then compress the resulting task vector τ_m into a sparse skill z_m via drop-and-rescale.
2. **Skill Consolidation** (§3.4): train one nested hypernetwork $\mathcal{G}_\phi$ to absorb all M action-expert targets $\{z_m^A\}_{m=1}^{M}$ into shared parameters ϕ with task embeddings.

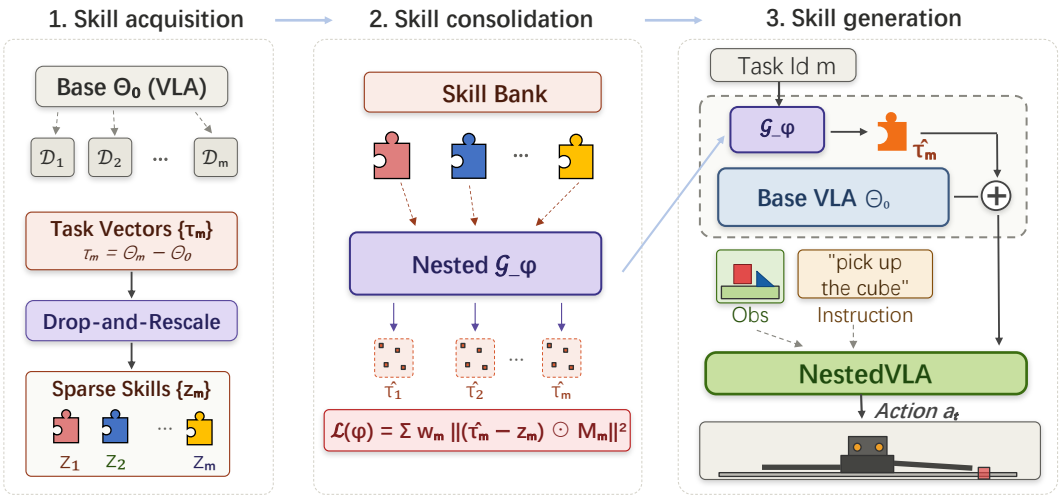


Figure 2: **NestedVLA pipeline.** (1) *Skill acquisition*: each task is fine-tuned independently from the same base Θ_0 and the resulting task vector τ_m is compressed into a sparse skill $z_m = \alpha \cdot \tau_m \odot M_m$ via drop-and-rescale. (2) *Skill consolidation*: a single nested hypernetwork $\mathcal{G}_\phi$ is trained against the action-expert portions of $\{z_m\}$ with a multi-task masked MSE loss. (3) *Skill generation*: at inference, querying $\mathcal{G}_\phi$ with task id m generates the action-expert update; the corresponding compressed VLM update is retrieved and both compose with the frozen base.

3. **Skill Generation** (§3.5): at inference, query $\mathcal{G}_\phi$ with the target task identity to generate the action-expert delta, retrieve the compressed VLM delta, and compose both updates with the frozen base.

3.3 Skill Acquisition

The acquisition stage produces, for each task, a sparse skill representation z_m that the hypernetwork can later be trained against. It comprises two steps: independent fine-tuning to obtain τ_m, and drop-and-rescale compression to turn τ_m into the compact z_m.

Independent fine-tuning. For each $\mathcal{D}_m$, we fine-tune Θ_0 to obtain Θ_m and the task vector $\tau_m = \Theta_m - \Theta_0$. We make no assumption on the training signal: Θ_m may be obtained via supervised fine-tuning, reinforcement learning, or any combination thereof, as long as all $\{\Theta_m\}$ are derived from the same Θ_0. This decoupling matches the data-silo realities of robotics, where specialized skills are often trained independently in their native pipelines.

Drop-and-rescale compression. A naive option is to use the dense τ_m as the hypernetwork target, but this is prohibitive in both memory and learnability for VLA backbones with billions of parameters. We observe that VLA fine-tuning deltas exhibit a *sparse-magnitude-dominated* structure: a small fraction of elements concentrate the bulk of the delta's ℓ_2 energy, while the rest behave like low-amplitude noise; in contrast, LLM SFT deltas are typically denser and benefit from quantization-plus-low-rank schemes such as D-QRELO [29]. We compared low-bit quantization, low-rank decomposition, and drop-and-rescale on VLA deltas and found drop-and-rescale dominates the fidelity-versus-compression trade-off (§D); we adopt it throughout NestedVLA.

Operator definition. Let $\tau_m^{(p)} \in \mathbb{R}^{d_p}$ denote the slice of τ_m for parameter tensor p. For a module budget $K \in (0, 1]$, we compute a per-module top-K binary mask $M_m^{(p)}$ that retains the elements with the largest magnitude within the module p belongs to. The skill representation is

$$z_m^{(p)} = \alpha \cdot \tau_m^{(p)} \odot M_m^{(p)}, \tag{3}$$

where α is a global rescale factor that compensates for the energy lost during sparsification. Module-level (rather than per-tensor) top-K allocates the budget adaptively across layers, which improves

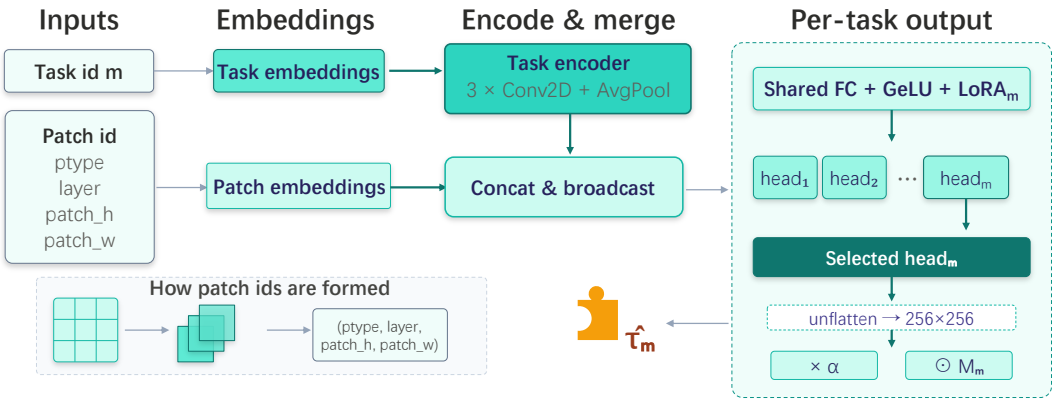


Figure 3: **Architecture of the nested hypernetwork $\mathcal{G}_\phi$.** Patches are addressed by $(\texttt{ptype}, \texttt{layer}, \texttt{patch_h}, \texttt{patch_w})$ and mapped to four per-task patch embeddings (*patch path*). The task id m is mapped to a task embedding and processed by a lightweight *task encoder* (2D convolutions plus average pooling) into a single task feature f_{task}. f_{task} is broadcast across all patches and concatenated with the patch embeddings, then passed through a *shared FC trunk* augmented with a per-task low-rank LoRA path (*LoRA path*). The trunk output is decoded by a *multi-head output* module that allocates one fully-connected projection per task (*multi-head path*) into a flat patch prediction. The precomputed action-expert mask M_m^A and global rescale α produce $\hat{\tau}_m^A = \alpha \cdot \mathcal{G}_\phi(m) \odot M_m^A$; the compressed VLM update is retained separately.

downstream fidelity (§D). The mask is deterministic given τ_m: it is precomputed once and applied multiplicatively to the loss at training (Eq. 4) and to the head output at inference (Eq. 5). The hypernetwork only learns the dense values on $M_m{=}1$ positions.

3.4 Skill Consolidation

Given the action-expert targets $\{z_m^A\}_{m=1}^M$, the consolidation stage trains a single hypernetwork $\mathcal{G}_\phi$ that absorbs the entire skill set into one set of shared parameters ϕ. We describe the parameter tokenization that exposes the backbone layout to the hypernetwork (§3.4.1), the architecture of $\mathcal{G}_\phi$ (§3.4.2), and the training objective (§3.4.3).

3.4.1 Patch-based Parameter Tokenization

A direct regression $\mathcal{G}_\phi(m) \to z_m$ over a billion-element output is infeasible. We tokenize the parameter space into a 2D grid of *patches*. Each weight tensor in the backbone is split into fixed-size $H_{\text{patch}} \times W_{\text{patch}}$ tiles (large tensors span multiple patches; small tensors are zero-padded into one). Every patch is uniquely addressed by four indices: parameter type (`ptype`, e.g. self-attention $Q/K/V/O$, FFN gate/up/down, layer norm), `layer` (transformer block index), and `patch_h`, `patch_w` (tile coordinates within the source tensor). This factorization exposes a structured spatial layout to the generator while keeping the per-token output dimension bounded. Concrete patch counts and tile sizes are reported in §4.1.

3.4.2 Hypernetwork Architecture

$\mathcal{G}_\phi$ has five trainable components, illustrated in Fig. 3: per-task *patch embeddings*, a *task encoder*, a *concat-and-broadcast* step, a *shared FC trunk* with a per-task LoRA path, and a *multi-head output* module. In our ablations (§4.3), the three task-conditioned components (patch embeddings, LoRA, multi-head output) are called the *patch path*, *LoRA path*, and *multi-head path*.

Per-task patch embeddings. Each of the four patch indices (`ptype`, `layer`, `patch_h`, `patch_w`) is mapped through a learnable embedding table indexed by both the patch index and the task id. The four resulting vectors are concatenated to form a per-patch feature. Per-task indexing (as opposed to a single shared table) lets the hypernetwork use different effective patch addresses for different tasks, which gives a small but consistent gain in our ablations.

Task encoder. The task identifier m is mapped to a learnable task embedding task_emb(m), reshaped onto a small spatial grid, and processed by a lightweight 2D convolutional encoder followed by global average pooling, yielding a single task feature $f_{\text{task}} \in \mathbb{R}^{D_t}$. This vector is the only carrier of task identity entering the rest of the network. The 2D-conv design is empirical; we did not observe meaningful gains from increasing its depth or width.

Concatenate and broadcast. f_{task} is broadcast across all patches and concatenated with the four patch embeddings to form a per-patch input $[f_{\text{task}}; \texttt{ptype}; \texttt{layer}; \texttt{patch_h}; \texttt{patch_w}]$. This is the only point where task and patch information meet; everything before it is either purely task-side or purely patch-side, and everything after it is patch-wise.

Shared FC trunk and LoRA path. A shared fully-connected layer of hidden dimension h with non-linearity provides task-invariant feature transport across patches. A per-task low-rank LoRA path of rank r runs in parallel to allow each task to slightly perturb the trunk's representation without a separate trunk per task. We ablate both the shared FC and the LoRA path in §4.3.

Multi-head output. The trunk output is decoded by a multi-head module: each task is served by a dedicated fully-connected projection that maps the trunk representation to a flat patch prediction, which is then unflattened into an $H_{\text{patch}} \times W_{\text{patch}}$ tile. Replacing the shared output (with optional low-rank task adapter) by per-task heads substantially mitigates multi-task interference at the cost of additional parameters in the output module (§4.3).

External rescale. The factor α in Eq. 3 is precomputed from $\{\tau_m\}$ and stored externally rather than predicted by the hypernetwork. This avoids polluting the regression objective with a small number of high-leverage scalars whose gradient signal is otherwise dominated by the dense main-value targets.

3.4.3 Training Objective

We train $\mathcal{G}_\phi$ with a multi-task masked regression loss

$$\mathcal{L}(\phi) \;=\; \sum_{m=1}^{M} w_m \cdot \frac{1}{|M_m^A|} \left\| \left(\mathcal{G}_\phi(m) - z_m^A\right) \odot M_m^A \right\|_2^2, \tag{4}$$

where $|M_m^A|$ counts the kept action-expert positions and $\{w_m\}$ are task-balancing weights that compensate for the heterogeneous learnability of different benchmarks (§4.1). Loss is computed only on $M_m{=}1$ positions; the base VLA Θ_0 is frozen throughout training, and the only trainable parameters are ϕ (patch embeddings, task encoder, shared FC, LoRA, multi-head output).

3.5 Skill Generation at Inference

At test time, given a target task m, we query the hypernetwork once, apply the precomputed action-expert mask and rescale, retrieve the compressed VLM update, and add both to the frozen base:

$$\hat{\Theta}_m^A = \Theta_0^A + \alpha\, \mathcal{G}_\phi(m) \odot M_m^A, \tag{5}$$

$$\hat{\Theta}_m^V = \Theta_0^V + z_m^V. \tag{6}$$

The resulting $\hat{\Theta}_m$ is a complete VLA model that can be deployed without any architectural modification, expert routing, or runtime mixture computation. A single trained $\mathcal{G}_\phi$ replaces the action-expert portions of M task-specific checkpoints with one shared generator, while retaining compressed VLM updates and small task-specific metadata.

4 Experiments

We evaluate NestedVLA on three closed-loop manipulation benchmarks and ask three questions: (i) can a single nested hypernetwork consolidate a heterogeneous skill set without storing a full checkpoint per skill (§4.2)? (ii) which design choices matter (§4.3)? (iii) where does the remaining gap to per-task fine-tuning come from (§4.4)?

Table 1: **Main result.** Multi-task hypernet on three RL benchmarks. We report `success_once` (%) for MetaWorld and ManiSkill, and `success_at_end` (%) for CALVIN, averaged over 3 evaluation seeds (typical std. $\pm 2.7\%$). *Params* is the total checkpoint storage relative to a single fine-tuned model ($1.0\times \approx 3.61$ B for $\pi_{0.5}$). NestedVLA outperforms every static and dynamic merging baseline by 41 pp on average at the lowest storage cost among methods that condition on task identity. Best per column in **bold**.

Method	Params	MetaWorld	ManiSkill	CALVIN	Avg.
Reference upper bounds					
Per-task FlowSDE (no merge)	3.0×	72.3	88.1	81.3	80.5
Static merging					
Averaging	1.0×	3.1	41.3	0.0	14.8
TIES [65]	1.0×	2.0	37.2	0.0	13.1
WUDI [5]	1.0×	2.7	32.8	0.0	11.8
TSV-Merging [18]	1.00×	3.9	36.3	0.0	13.4
Dynamic merging					
TALL-Mask [50]	2.09×	8.1	53.8	6.0	22.6
EMR-Merging [22]	2.89×	8.2	54.4	6.3	23.0
Twin-Merging [35]	2.25×	10.4	56.9	8.7	25.3
Free-Merging [70]	2.16×	9.4	58.3	9.1	25.6
NestedVLA (ours)					
NestedVLA	**1.5×**	**37.9**	**87.2**	**75.0**	**66.7**

4.1 Setup

Backbone and skills. We instantiate Θ_0 as $\pi_{0.5}$ [43] and collect three skills by independently fine-tuning with FlowSDE [4] on MetaWorld [38], ManiSkill [48], and CALVIN [39], yielding task vectors $\{\tau_m\}_{m=1}^{3}$. The backbone factorizes into a VLM half ($\sim$3.21 B params) and an action expert ($\sim$0.40 B params); we confine the hypernetwork to the action expert and keep per-task VLM storage, motivated by a sanity test in which TIES- and WUDI-merged VLMs (with GT action experts) drop ≥ 60 pp on MetaWorld and collapse CALVIN to 0% chain completion.

Benchmarks and metrics. **MetaWorld** (32 tasks, 256 trajectories) and **ManiSkill** (25 tasks, 320 trajectories) report `success_once`: a trajectory counts as successful if it reaches the goal at any point during rollout. **CALVIN ABC-D** (96 rollouts) is a long-horizon split where each rollout is a chain of *five* language-conditioned sub-tasks; we report `success_at_end`, the fraction of rollouts that complete the entire chain. For diagnostic purposes we also evaluate the *ideal* top-K reconstruction $\Theta_0 + \tau_m \odot M_m$, i.e. the GT task vector masked at the same sparsity K but without learning $\mathcal{G}_\phi$. This upper-bounds any method that uses the same K and mask. All evaluations are run with the official RLinf [67] pipeline and averaged over 3 seeds (typical std. $\pm 2.7\%$).

Skill compression and hypernetwork. Each τ_m is compressed via drop-and-rescale at per-module top-K=10% with global rescale α=1.1. The hypernetwork $\mathcal{G}_\phi$ covers 18 transformer blocks and 9 weight types (162 tensors), tiled into 256×256 patches (T=6,480 patches per task). The shared FC has hidden width h=2048 with rank-1024 per-task LoRA, and the Multi-head Output allocates one $h\to$65,536 projection per task ($\sim$430 M trainable parameters).

Training. We train 6,000 steps with batch size 3, base LR 3×10^{-4}, LoRA multiplier 3.0, 600-step warmup and 1,500-step cosine tail, and task weights $(1.0, 0.4, 1.0)$ for (MW, MA, CA) to compensate for the disproportionately large $\|\tau_{\text{MA}}\|$. All runs fit on a single A6000 GPU ($\sim$10 h).

Baselines. **Static merging**: Averaging, TIES [65], WUDI [5], TSV-Merging [18]. **Dynamic merging**: TALL-Mask [50], EMR-Merging [22], Twin-Merging [35], and Free-Merging [70], all of which retain per-task masks (and in some cases scaling factors or low-rank residuals) at inference. **Per-task FlowSDE**: independent fine-tuning, the performance ceiling.

Table 2: Architecture ablations of $\mathcal{G}_\phi$ at 2,000 steps. $\mathcal{G}_\phi$ targets the action expert only; #params refers to the hypernetwork itself, not to the underlying $\pi_{0.5}$ backbone. We ablate each of the three trainable paths – the per-task output projection (*Multi-head path*), the per-task patch embeddings (*patch path*), and the shared-FC LoRA (*LoRA path*) – and report the final in-mask standardized loss per task (lower is better; averaged over 3 seeds, std. ± 0.05).

Variant	#params	MW ↓	MA ↓	CA ↓
w/o Multi-head path	161 M	1.33	1.71	1.36
w/o patch path	430 M	1.02	1.14	1.05
w/o LoRA path	417 M	**0.91**	**0.92**	**0.89**
narrow (h=1024)	221 M	1.43	1.56	1.55
Default	430 M	1.08	1.17	1.14

4.2 Main Results

Table 1 reports our headline result. NestedVLA attains an average success rate of 66.7%, outperforming every static and dynamic merging baseline by 41–54 pp on average. On **ManiSkill**, the multi-task hypernetwork reaches 87.2%, within 0.9 pp of the per-task FlowSDE ceiling (88.1%). On **CALVIN**, NestedVLA reaches 75.0% `success_at_end`, demonstrating that a single shared ϕ can encode the skill at the precision required for chain-completion-grade evaluation. The remaining gap to the per-task FlowSDE reference (-13.8 pp on average) concentrates almost entirely on **MetaWorld**, which we analyze in §4.4.

Static merging. None of the static merging baselines is competitive on cross-benchmark VLA fusion. Plain parameter averaging reaches 14.8% average success; TIES [65], WUDI [5], and TSV-Merging [18] each *underperform* averaging despite incorporating sign-conflict resolution, weighted updates, or singular-vector compression. All four methods drive CALVIN to 0% chain completion, consistent with our hypothesis (§1) that VLA representations are more sensitive to parameter perturbations than the SFT deltas these methods were primarily evaluated on.

Dynamic merging. Retaining per-task information at inference time helps, but only modestly. TALL-Mask [50] achieves 22.6% average success with per-task binary masks at $2.09\times$ storage. EMR-Merging [22] adds per-task sign modulators and scalars and reaches 23.0% at $2.89\times$ storage, the highest budget of any baseline. Twin-Merging [35] attaches per-task low-rank residuals at $2.25\times$ and reaches 25.3%. Free-Merging [70], the strongest dynamic baseline, applies frequency-domain filtering on per-task deltas and reaches 25.6% at $2.16\times$. Despite their parameter budgets ranging from $2.09\times$ to $2.89\times$, the four methods cluster within a 3 pp band, suggesting the bottleneck is not storage but the underlying paradigm of *selecting* from a fixed set of base values.

Storage analysis. NestedVLA's $1.5\times$ overall budget decomposes asymmetrically across the two halves of the backbone. On top of the $1.0\times$ shared base, the VLM half ($\sim$3.21 B parameters, $\sim 89\%$ of the backbone) is stored per-task as a top-K=10% compressed delta (a 1-bit mask plus the kept values), contributing $\sim 0.35\times$ across the three tasks. The action expert ($\sim$0.40 B parameters, $\sim 11\%$) is *generated* by the hypernetwork at inference and stores only a precomputed mask per task ($\sim 0.01\times$); the hypernetwork itself ($\sim$0.43 B parameters) adds $\sim 0.12\times$. The detailed per-component derivation is given in App. C. The trainable capacity is therefore concentrated on the action side, the harder consolidation target where per-task deltas are denser and more conflicting, precisely where it is needed. This contrasts sharply with the dynamic baselines: EMR-Merging stores per-task masks *and* per-task scaling factors uniformly across the entire backbone at $2.89\times$ to reach only 23.0%, while Twin-Merging and Free-Merging both pay $\sim 2.2\times$ for per-task residuals or frequency-domain corrections without distinguishing where additional capacity is useful. NestedVLA's storage advantage at the evaluated three-skill setting comes from concentrating generation capacity on the action expert. Total storage still grows with the number of skills because compressed VLM residuals, masks, and task-specific generator components must be retained, but each skill adds substantially less than a full VLA checkpoint.

Table 3: Per-task ceiling analysis. *Ideal*: the upper bound at sparsity K, computed by adding $\tau_m \odot M_m$ directly to Θ_0 without learning $\mathcal{G}_\phi$. The MetaWorld gap is bounded by the ideal itself, indicating a sparsity-budget effect rather than a hypernetwork-capacity problem.

Task	NestedVLA	Per-task FlowSDE	Ideal (K=10%)	Ideal (K=20%)
MetaWorld	37.9	72.3	48.4	60.2
ManiSkill	87.2	88.1	87.5	87.5
CALVIN	75.0	81.3	78.1	85.4

4.3 Ablations

Architectural ingredients. Table 2 ablates the three trainable paths of $\mathcal{G}_\phi$ at $2{,}000$ steps, reporting in-mask standardized loss. *Multi-head Output* is the largest single contributor: removing this path (*w/o Multi-head path*) and falling back to a shared `fc_main` plus per-task LoRA caps every benchmark and degrades the loss by 0.2–0.5, indicating that the per-task targets $\{z_m\}$ contain genuinely incompatible value distributions that a low-rank shared head cannot reconcile. *Per-task patch embeddings*, in contrast, give small marginal gains (*w/o patch path* differs from default by < 0.1 on every task); their primary value is enabling sharper task discrimination at convergence rather than reducing average loss. Surprisingly, the *shared FC LoRA* is dispensable: removing it (*w/o LoRA path*) yields the lowest loss on all three benchmarks at 97% of default parameter cost. Once the output is fully task-specific via Multi-head Output, additional task-specific perturbation in the FC harms early optimization without improving the final fit.

Compression operator. We compared three operators at matched parameter budgets in a per-task fidelity study ($\Theta_0 + \hat{\tau}_m$ evaluated on benchmark m). One-bit quantization with a per-tensor scalar (adapted from D-QRELO [29]) yields reconstruction errors of 0.60–0.96 and degrades task success by -71 to -81 pp, because VLA RL deltas have an approximately zero-mean, near-isotropic element distribution that a single scalar cannot capture. Augmenting drop-and-rescale with a rank-r SVD residual yields at most $+0.005$ absolute fidelity improvement while inflating the token budget by an order of magnitude. Drop-and-rescale at K=10% achieves reconstruction errors of 0.30 on the VLM half and 0.19 on the action half (substantially below the one-bit baseline at lower storage than residual SVD), and is the only operator of the three that requires no calibration data.

Loss-side choices. We further ablated three knobs of the training objective on top of our default. Disabling the out-of-mask penalty ($w_{\text{out}} = 0$) is the single most effective lever, adding $+2.4$ pp average success; positions outside M_m are masked at inference, so penalizing them at train time forces $\mathcal{G}_\phi$ to allocate capacity to a function it does not need. Magnitude weighting ($|\tau_m|^p$) and absolute loss space each *harm* performance individually but combine constructively with $w_{\text{out}} = 0$, suggesting they act as implicit regularizers that compensate for the otherwise unbounded growth of the out-of-mask region. The standardized + $w_{\text{out}} = 0$ recipe is our default; full ablation results are reported in Appendix G.

4.4 Where Does the Remaining Gap Come From?

Table 3 localizes the remaining gap to MetaWorld. On ManiSkill and CALVIN, NestedVLA reaches 99% and 96% of the K=10% ideal respectively, indicating that the hypernetwork extracts near-ceiling value from the sparsity budget it was given. MetaWorld, in contrast, attains only 37.9% against an 72.3% FlowSDE reference. However, the K=10% ideal itself is only 48.4% on MetaWorld, against which NestedVLA's retention is also 78%, on par with the other two benchmarks.

The MetaWorld gap is therefore primarily a sparsity-budget problem at skill acquisition, not a hypernetwork-capacity problem at consolidation. Raising K to 0.20 lifts the MetaWorld ideal to 60.2% at little cost elsewhere (Tab. 3), making per-task adaptive K a natural remedy.

5 Conclusion

We studied skill consolidation in VLA models when training data is fragmented across benchmark-specific tasks. Standard model-merging methods degrade catastrophically on cross-benchmark VLA fusion: the strongest baseline reaches only 25.6% average success across MetaWorld, ManiSkill, and CALVIN. We introduced NestedVLA, which replaces action-expert parameter *selection* with parameter *generation* via a nested hypernetwork, reaching 66.7% average success at $1.5\times$ storage ($+41.1$ pp over the strongest dynamic baseline), and matching per-task fine-tuning within 1 pp on ManiSkill while approaching the $K{=}10\%$ sparsity-budget ceiling on CALVIN. The MetaWorld gap is bounded by the top-K ideal itself, with adaptive sparsity the natural next direction.

Acknowledgments and Disclosure of Funding

Funding statement. This work was supported by the Hong Kong Polytechnic University Internal Research Fund (the Projects of RIAIoT, Project ID P0065264), UDomain Web Hosting Company Limited (Collaboration Scheme, Project ID P0064507), and the Innovation and Technology Fund–Guangdong-Hong Kong Technology Cooperation Funding Scheme (ITF-TCFS) (Project ID P0049179).

Competing interests. The authors declare that they have no competing interests.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. doi: 10.48550/ARXIV.2410.24164.

[2] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems (RSS)*, 2023. doi: 10.15607/RSS.2023.XIX.025.

[3] S. Cai, Y. Gu, Z. Wang, Y. Wang, Y. Yan, W. Wang, Y. Liu, G. Zhu, S. Huang, M. Li, and H. Yang. From parameters to behaviors: A survey of model fusion for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. URL `https://arxiv.org/abs/2609.19553`.

[4] K. Chen, Z. Liu, T. Zhang, Z. Guo, S. Xu, H. Lin, H. Zang, X. Li, et al. π_{RL}: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 2025.

[5] R. Cheng, F. Xiong, Y. Wei, W. Zhu, and C. Yuan. Whoever started the interference should end it: Guiding data-free model merging via task vectors. *arXiv preprint arXiv:2503.08099*, 2025.

[6] S. Dey, J.-N. Zaech, N. Nikolov, L. Van Gool, and D. P. Paudel. ReVLA: Reverting visual domain limitation of robotic foundation models. In *ICRA*, pages 8679–8686. IEEE, 2025. doi: 10.1109/ICRA55743.2025.11128823.

[7] J. Dong, J. Li, T. Zheng, and W. Lin. HybridFlow: Resource-adaptive subtask routing for efficient edge-cloud LLM inference. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 25922–25943. PMLR, 2026. URL `https://proceedings.mlr.press/v306/dong26g.html`.

[8] G. Du and W. Lin. Dynamic model merging made slim. In *Advances in Neural Information Processing Systems*, 2026. URL `https://arxiv.org/abs/2605.18904`.

[9] G. Du, H. Deng, J. Su, and Y. Huang. End-to-end rain streak removal with RAW images. *arXiv preprint arXiv:2312.13304*, 2023.

[10] G. Du, R. Jiang, S. Yang, H. Li, W. Chen, K. Li, S. K. Goh, and H.-K. Tang. Impacts of darwinian evolution on pre-trained deep neural networks. In *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2024.

[11] G. Du, J. Lee, J. Li, R. Jiang, Y. Guo, S. Yu, H. Liu, S. K. Goh, et al. Parameter competition balancing for model merging. *NeurIPS*, 37, 2024.

[12] G. Du, J. Li, H. Liu, R. Jiang, S. Yu, Y. Guo, S. K. Goh, and H.-K. Tang. Knowledge fusion by evolving weights of language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11727–11742. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.698. URL `https://aclanthology.org/2024.findings-acl.698/`.

[13] G. Du, Z. Fang, J. Li, J. Li, R. Jiang, S. Yu, Y. Guo, Y. Chen, S. K. Goh, H.-K. Tang, D. He, H. Liu, and M. Zhang. Neural parameter search for slimmer fine-tuned models and better transfer. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32668–32687. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1570. URL `https://aclanthology.org/2025.acl-long.1570/`.

[14] G. Du, Z. Li, X. Zhou, J. Li, Z. Shi, W. Lin, H.-K. Tang, X. Li, F. Liu, W. Wang, M. Zhang, and J. Li. Knowledge fusion of large language models via modular SkillPacks. In *International Conference on Learning Representations*, 2026. URL `https://arxiv.org/abs/2505.18502`.

[15] Z. Fang, G. Du, S. Yu, Y. Guo, Y. Zhang, Y. Cao, J. Li, H.-K. Tang, and S. K. Goh. To see a world in a spark of neuron: Disentangling multi-task interference for training-free model

merging. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15720–15740. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.emnlp-main.793/.

[16] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3259–3269. PMLR, 2020.

[17] Y. Fu, Z. Zhang, Y. Zhang, Z. Wang, Z. Huang, and Y. Luo. MergeVLA: Cross-skill model merging toward a generalist vision-language-action agent. *arXiv preprint arXiv:2511.18810*, 2025.

[18] A. A. Gargiulo, D. Crisostomi, M. S. Bucarelli, S. Scardapane, F. Silvestri, and E. Rodolà. Task singular vectors: Reducing task interference in model merging. In *CVPR*, pages 18695–18705, 2025.

[19] Y. Gu, S. Cai, Z. Wang, W. Wang, Y. Wang, P. Wang, S. Huang, S. Lu, J. Wu, and H. Yang. FeatCal: Feature calibration for post-merging models. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2605.13030.

[20] Y. Gu, Y. Wang, Z. Li, S. Cai, Y. Liu, J. Li, Z. Wang, and H. Yang. SMAT: Simple and efficient merge-aware training, 2026. URL https://arxiv.org/abs/2609.33437.

[21] D. Ha, A. M. Dai, and Q. V. Le. HyperNetworks. In *International Conference on Learning Representations (ICLR)*, 2017.

[22] C. Huang, P. Ye, T. Chen, T. He, X. Yue, and W. Ouyang. EMR-Merging: Tuning-free high-performance model merging. *NeurIPS*, 37:122741–122769, 2024.

[23] G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *ICLR*. OpenReview.net, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

[24] J. Jin, H. Liu, Y. Bai, Y. Lou, Z. Wang, T. Yuan, J. Chen, Y. Zhu, F. Zeng, X. Zhu, T. Feng, and Y. Xu. Unveiling fine-grained visual traces: Evaluating multimodal interleaved reasoning chains in multimodal STEM tasks, 2026. URL https://arxiv.org/abs/2604.19697.

[25] S. Karamcheti, S. Nair, A. Balakrishna, P. Liang, T. Kollar, and D. Sadigh. Prismatic VLMs: Investigating the design space of visually-conditioned language models. In *ICML*. OpenReview.net, 2024. URL https://openreview.net/forum?id=6FXtu8clyp.

[26] R. Karimi Mahabadi, S. Ruder, M. Dehghani, and J. Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Association for Computational Linguistics (ACL)*, 2021.

[27] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

[28] J. Li, G. Du, J. Li, S. K. Goh, W. Wang, Y. Wang, F. Liu, H.-K. Tang, S. Alharbi, D. He, and M. Zhang. Multi-modality expansion and retention for LLMs through parameter merging and decoupling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30866–30887. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1491. URL https://aclanthology.org/2025.acl-long.1491/.

[29] J. Li, S. Song, G. Du, N. Wong, X. Liu, Y. Li, M. Zhang, J. Li, and X. Li. D-QRELO: Training- and data-free delta compression for large language models via quantization and residual low-rank approximation. *arXiv preprint arXiv:2604.16940*, 2026.

[30] S. Li, G. Du, A. Zhao, W. Lin, T. Yuan, and X. Shen. Post-training shifts confidence: A three-stage analysis of how SFT, RL, and OPD shape CoT calibration. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, pages 34126–34141, Budapest, Hun-

gary, 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.findings-emnlp.1602/.

[31] Z. Li, G. Du, W. Guo, Y. Zhou, X. Li, W. Wang, F. Liu, Y. Wang, D. Ye, M. Zhang, and J. Li. Multi-objective large language model alignment with hierarchical experts. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2505.20925.

[32] Z. Li, G. Du, Z. Shi, W. Guo, W. Yao, Y. Zhou, J. Zhang, and J. Li. Skill weaving: Efficient LLM improvement via modular Skillpacks. In *Annual Meeting of the Association for Computational Linguistics*, 2026. URL https://arxiv.org/abs/2605.22205.

[33] Z. Liang, D. Tang, Y. Zhou, X. Zhao, M. Shi, W. Zhao, Z. Li, P. Wang, K. Schürholt, D. Borth, M. M. Bronstein, Y. You, Z. Wang, and K. Wang. Drag-and-drop LLMs: Zero-shot prompt-to-weights. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5126.

[34] W. Lin, H. Lan, H. He, and B. Li. Graph-relational federated learning: Enhanced personalization and robustness. *IEEE Transactions on Dependable and Secure Computing*, 22(5): 5773–5785, 2025. doi: 10.1109/TDSC.2025.3574329. URL https://doi.org/10.1109/TDSC.2025.3574329.

[35] Z. Lu, C. Fan, W. Wei, X. Qu, D. Chen, and Y. Cheng. Twin-Merging: Dynamic integration of modular expertise in model merging. *NeurIPS*, 37:78905–78935, 2024.

[36] D. Marczak, S. Magistri, S. Cygert, B. Twardowski, A. D. Bagdanov, and J. van de Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. *arXiv preprint arXiv:2502.04959*, 2025.

[37] M. Matena and C. Raffel. Merging models with fisher-weighted averaging. In *NeurIPS*, 2022.

[38] R. McLean, E. Chatzaroulas, L. McCutcheon, F. Röder, T. Yu, Z. He, K. Zentner, R. Julian, et al. Meta-World+: An improved, standardized, RL benchmark. In *NeurIPS Datasets and Benchmarks Track*, 2025.

[39] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. doi: 10.1109/LRA.2022.3180108.

[40] C. Meng, P. Feng, R. Fu, H. L. Lee, X. Du, Z. Kang, Z. Zhang, W. Zhou, C. Ouyang, and Z. Gan. Group cognition learning: Making everything better through governed two-stage agents collaboration. *arXiv preprint arXiv:2605.00370*, 2026.

[41] C. Meng, P. Feng, J. J. Pang, C.-T. Liao, R. Fu, Z. Kang, Z. Gan, and C. Ouyang. Adaptive hierarchical representation alliance for multimodal learning. *arXiv preprint arXiv:2608.22863*, 2026.

[42] C. Meng, J. Luo, P. Feng, Z. Yan, C. Hu, Z. Gan, and C. Ouyang. Mitigating shared-private branch imbalance via dual-branch rebalancing for multimodal sentiment analysis. *arXiv preprint arXiv:2604.25179*, 2026.

[43] Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. doi: 10.48550/ARXIV.2504.16054.

[44] M. Qi, R. Fu, P. Feng, X. Li, Y. Cai, Y. Guo, S. Zhang, S. J. Fong, L. Ma, and B. Li. CACR: Reinforcing temporal answer grounding in instructional video via candidate-aware causal reasoning. *arXiv preprint arXiv:2606.08436*, 2026.

[45] H. Qiu, X. Wu, L. Feng, and K. C. Tan. Fine-grained model merging via modular expert recombination. *arXiv preprint arXiv:2602.06552*, 2026. URL https://arxiv.org/abs/2602.06552.

[46] H. Qiu, X. Wu, and K. C. Tan. Towards adaptive continual model merging via manifold-aware expert evolution. *arXiv preprint arXiv:2604.22464*, 2026. URL https://arxiv.org/abs/2604.22464.

[47] G. Stoica, P. Ramesh, B. Ecsedi, L. Choshen, and J. Hoffman. Model merging with SVD to tie the knots. In *ICLR*. OpenReview.net, 2025. URL https://openreview.net/forum?id=67X93aZHII.

[48] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, et al. ManiSkill3: Gpu parallelized robotics simulation and rendering for generalizable embodied AI. *arXiv preprint arXiv:2410.00425*, 2024. doi: 10.48550/ARXIV.2410.00425.

[49] J. von Oswald, C. Henning, B. F. Grewe, and J. Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2020.

[50] K. Wang, N. Dimitriadis, G. Ortiz-Jiménez, F. Fleuret, and P. Frossard. Localizing task information for improved model merging and compression. In *ICML*. OpenReview.net, 2024. URL https://openreview.net/forum?id=DWT9uiGjxT.

[51] K. Wang, D. Tang, B. Zeng, Y. Yin, Z. Xu, Y. Zhou, Z. Zang, T. Darrell, Z. Liu, and Y. You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

[52] K. Wang, D. Tang, W. Zhao, K. Schürholt, Z. Wang, and Y. You. Scaling up parameter generation: A recurrent diffusion approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3111.

[53] W. Wang, Y. Gu, S. Cai, Y. Wang, P. Wang, J. Wu, and H. Yang. E-PMQ: Expert-guided post-merge quantization with merged-weight anchoring, 2026. URL https://arxiv.org/abs/2605.16882.

[54] Y. Wang, P. Ding, L. Li, C. Cui, Z. Ge, X. Tong, W. Song, H. Zhao, et al. VLA-Adapter: An effective paradigm for tiny-scale vision-language-action model. *arXiv preprint arXiv:2509.09372*, 2025. doi: 10.48550/ARXIV.2509.09372.

[55] Y. Wang, Y. Gu, S. Lu, Y. Yang, Z. Yan, C. Xie, J. Wu, and H. Yang. Access sets matter: Budgeting expert reads for scalable weight-space model merging, 2026. URL https://arxiv.org/abs/2605.29489. ICML 2026 Workshop on Weight-Space Symmetries.

[56] Y. Wang, Y. Gu, Y. Zhang, Q. Zhou, Z. Yan, C. Xie, X. Wang, J. Yuan, and H. Yang. Model merging scaling laws in large language models. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 126402–126437. PMLR, 2026. URL https://proceedings.mlr.press/v306/wang26bl.html.

[57] Y. Wang, Y. Li, J. Liu, F.-L. Zhang, X. Zheng, and N. A. Dodgson. RL-ScanIQA: Reinforcement-learned scanpaths for blind 360° image quality assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 37401–37412, 2026.

[58] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, et al. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, pages 23965–23998. PMLR, 2022.

[59] M. Wu, X. Zhao, Z. Wang, R. Feng, Z. Wang, and K. Wang. HY-WU (part i): An extensible functional neural memory framework and an instantiation in text-guided image editing. *arXiv preprint arXiv:2603.07236*, 2026.

[60] X. Xiao, Y. Zhang, X. Li, T. Wang, X. Wang, Y. Wei, J. Hamm, and M. Xu. Visual instance-aware prompt tuning. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 2880–2889, 2025. doi: 10.1145/3746027.3754858. URL https://doi.org/10.1145/3746027.3754858.

[61] X. Xiao, X. Li, C. Han, T. Wang, L. Zhao, Y. Zhang, G. Hu, R. Jiang, X. Li, X. Wang, and M. Xu. Adapting vision foundation models with cascaded semantics. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=SSsobNZJPO.

[62] X. Xiao, X. Li, Y. Zhang, C. Han, T. Liu, T. Wang, R. Jiang, J. Hamm, X. Wang, and M. Xu. Layer-specific prompt fusion discovery via differentiable search in vision foundation models. In *Computer Vision – ECCV 2026*, volume 17077 of *Lecture Notes in Computer Science*, pages 391–409. Springer, 2026. doi: 10.1007/978-3-032-37041-9_21. URL https://doi.org/10.1007/978-3-032-37041-9_21.

[63] X. Xiao, C. Liu, C.-T. Liao, Y. Zhang, Q. Lan, Y. Wei, L. Zhao, J. Wang, J. Gu, M. Ye, T. Wang, and H. Xu. Staying VIGILant: Mitigating visual laziness via counterfactual visual alignment in MLLMs. In *Computer Vision – ECCV 2026*, volume 17014 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2026. doi: 10.1007/978-3-032-37232-1_16. URL https://doi.org/10.1007/978-3-032-37232-1_16.

[64] X. Xiao, C. Ma, Y. Zhang, C. Liu, Z. Wang, Y. Li, L. Zhao, G. Hu, T. Wang, and H. Xu. Not all directions matter: Towards structured and task-aware low-rank model adaptation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2132–2154. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.97. URL https://aclanthology.org/2026.acl-long.97/.

[65] P. Yadav, D. Tam, L. Choshen, C. A. Raffel, and M. Bansal. TIES-Merging: Resolving interference when merging models. In *NeurIPS*, 2023.

[66] S. Yang, C. Wang, Y. Chen, Z. Wang, L. Tang, H. Gui, J. Ye, C. Lu, X. Wu, M. Zhu, P. Chen, S. Liu, Z. Tian, H. Zhao, B. Yu, and J. Jia. Beyond data scaling: Representation-centric continued pre-training for vision-language-action models. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2608.27550.

[67] C. Yu, Y. Wang, Z. Guo, H. Lin, S. Xu, H. Zang, Q. Zhang, Y. Wu, et al. RLinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. *arXiv preprint arXiv:2509.15965*, 2025.

[68] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*. OpenReview.net, 2024. URL https://openreview.net/forum?id=fq0NaiU8Ex.

[69] J. Zhang, S. Chen, J. Liu, and J. He. Composing parameter-efficient modules with arithmetic operations. *arXiv preprint arXiv:2306.14870*, 2023.

[70] S. Zheng and H. Wang. Free-merging: Fourier transform for model merging with lightweight experts. *arXiv preprint arXiv:2411.16815*, 2024.

[71] S. Zheng, H. Wang, C. Huang, X. Wang, T. Chen, J. Fan, S. Hu, and P. Ye. Decouple and orthogonalize: A data-free framework for LoRA merging. *arXiv preprint arXiv:2505.15875*, 2025.

[72] Q. Zhou, Y. Zhang, Y. Gu, Y. Wang, Z. Sang, Z. Yan, Z. Li, S. Zhang, F. Wu, and H. Yang. Democratizing AI through model fusion: A comprehensive review and future directions. *Nexus*, 2025.

[73] Q. Zhou, Y. Zhang, Y. Gu, Y. Wang, Z. Yan, Z. Li, C. Y. Chung, and H. Yang. Model fusion for scalable and sustainable artificial intelligence: A review and outlook. *Journal of Modern Power Systems and Clean Energy*, 14(1):37–49, 2026.

[74] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

Appendix Overview

This appendix provides supplementary material that supports the main paper. We organize it into four sections. **Appendix A** reports the full implementation details of NestedVLA, including hypernetwork architecture, training hyperparameters, and evaluation protocol, for reproducibility. **Appendix B** describes every baseline used in Table 1, listing their merge mechanism and the configuration we used. **Appendix C** makes the parameter budgets reported in Table 1 explicit, deriving the Params column entry-by-entry. **Appendix D** expands the discussion of skill compression in §4.3, comparing one-bit quantization, low-rank decomposition, and drop-and-rescale on VLA deltas, and reporting a full keep-ratio sweep that motivates our default $K{=}10\%$. **Appendix F** presents the static VLM merging sanity test that justifies our design choice of confining $\mathcal{G}_\phi$ to the action expert and keeping per-task VLM storage. **Appendix G** reports the full loss-side ablation summarized in §4.3, including the non-monotone interaction between out-of-mask penalty, loss space, and magnitude weighting. **Appendix H** lists the limitations of the current study. **Appendix I** states the broader-impact considerations referenced in the NeurIPS checklist.

For broader multimodal context, related studies examine governed agent collaboration, reinforcement-learned panoramic scanpaths for quality assessment, shared–private branch rebalancing, temporal answer grounding, and fine-grained reasoning evaluation [40, 57, 42, 44, 24]. Earlier visual studies by the authors investigate evolutionary refinement of pretrained networks and RAW-image restoration [10, 9].

A Implementation Details

This section reports the exact hyperparameters, compute budget, and evaluation protocol used in our experiments. All quantities not stated explicitly in the main paper are listed here.

A.1 Backbone and Skill Collection

We use $\pi_{0.5}$ [43] as the shared base Θ_0, which factorizes into a $\sim$3.21 B-parameter VLM half (a PaliGemma backbone with vision tower, language model, and projectors) and a $\sim$0.40 B-parameter action expert (transformer decoder for continuous action chunks via flow matching). The total backbone is $\sim$3.61 B parameters.

Each per-task skill is obtained by independently fine-tuning Θ_0 with FlowSDE [4] on the corresponding benchmark dataset. The three skills span heterogeneous task suites (MetaWorld: 32 language-conditioned manipulation tasks; ManiSkill: 25 object manipulation tasks across diverse rigid bodies; CALVIN: long-horizon language-conditioned tabletop tasks). We do not modify the FlowSDE training pipeline; the resulting per-task checkpoints constitute our $\{\Theta_m\}_{m=1}^{3}$.

A.2 Hypernetwork Architecture

Table 4 lists the full hypernetwork specification. The hypernetwork covers all 162 trainable weight tensors of the action expert (18 transformer blocks $\times$ 9 weight types: self-attention $Q/K/V/O$, FFN gate/up/down, and two layer norms). Tensors are tiled into $256{\times}256$ patches, with zero-padding for tensors smaller than $256{\times}256$ and grid splits for tensors larger than this size. The total patch count per task is $T = 6{,}480$.

A.3 Training Details

The hypernetwork is trained with AdamW (base learning rate $3{\times}10^{-4}$, LoRA group multiplier 3.0, weight decay 0.1 on non-LoRA parameters and 0 on LoRA), batch size 3 with one example per task per step, 6,000 total steps, 600-step linear warmup, and a 1,500-step cosine decay tail. Task-balancing weights are $(1.0, 0.4, 1.0)$ for (MetaWorld, ManiSkill, CALVIN); the lower weight on ManiSkill compensates for the disproportionately large $\|\tau_{\text{ManiSkill}}\|$, which would otherwise dominate multi-task gradients (see §G).

All experiments fit on a single NVIDIA A6000 GPU (48 GB memory). Wall-clock training time is $\sim$10 hours per run; peak GPU memory is 42.6 GB during training. We enable `PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True` to reduce fragmentation under bf16.

Table 4: Hypernetwork architecture and training hyperparameters.

Component	Value
Backbone	
Base model	$\pi_{0.5}$ ($\sim$3.61 B params)
VLM half	$\sim$3.21 B params (frozen, per-task stored)
Action expert	$\sim$0.40 B params (target of $\mathcal{G}_\phi$)
Trainable tensors	162 (18 blocks $\times$ 9 weight types)
Patch tokenization	
Patch tile size	256×256
Total patches per task	$T = 6{,}480$
Embeddings	
`ptype` embedding dim	256
`layer` embedding dim	128
`patch_h` / `patch_w` embed dim	64 each
Task embedding dim	512 (reshaped to 16×16 grid)
Task encoder	
Layers	$3\times$ Conv2D (kernel 5, channels $64\to256\to512$)
Pooling	global average
Output dim (D_t)	512
Trunk and output	
Shared FC hidden width h	2048
Per-task LoRA rank r	1024
Multi-head output	1 FC head per task ($h \to 65{,}536$)
Total trainable params	$\sim$430 M
Skill compression	
Keep ratio K	0.10 (per-module top-K)
Global rescale α	1.1
Optimization	
Optimizer	AdamW
Base learning rate	3×10^{-4}
LoRA group LR multiplier	3.0
Weight decay	0.1 (non-LoRA), 0 (LoRA)
Batch size	3 (one example per task per step)
Total steps	6,000
Warmup steps	600
Cosine tail steps	1,500
Task weights $\{w_m\}$	$(1.0, 0.4, 1.0)$ for (MW, MA, CA)
Precision	bfloat16
Compute	
Hardware	$1\times$ NVIDIA A6000 (48 GB)
Peak GPU memory	42.6 GB
Wall-clock training	$\sim$10 hours per run

Inference cost is dominated by the forward pass through $\pi_{0.5}$; the additional hypernetwork query adds negligible overhead and is performed once per skill switch.

A.4 Evaluation Protocol

MetaWorld: 256 trajectories across 32 tasks; we report `success_once`, the fraction of trajectories that reach the goal at any timestep. **ManiSkill:** 320 trajectories across 25 tasks; same metric. **CALVIN ABC-D:** 96 rollouts on the long-horizon split, each a sequence of five language-conditioned sub-tasks; we report `success_at_end`, the fraction of rollouts that complete the entire chain. All evaluations are run with the official RLinf [67] pipeline. All numbers in the main paper are averaged over 3 evaluation seeds with fixed initial-state sampling; the typical seed-to-seed standard deviation is $\pm 2.7\%$.

B Baselines Details

This section describes each baseline used in Table 1: four *static* merging methods (producing a single fixed delta), four *dynamic* merging methods (retaining per-task auxiliary information at inference), and a per-task fine-tuning reference. All baselines start from the same set of task vectors $\{\tau_m = \Theta_m - \Theta_0\}_{m=1}^{3}$ obtained by independently fine-tuning $\Theta_0 = \pi_{0.5}$ with FlowSDE on MetaWorld, ManiSkill, and CALVIN.

Per-task FlowSDE (reference upper bound). Each task is fine-tuned independently from Θ_0 with FlowSDE [4], the SDE-form flow-matching objective of $\pi_{0.5}$, yielding three full checkpoints $\{\Theta_m\}_{m=1}^{3}$ at $3.0\times$ storage. This is the performance ceiling for any approach that does not change the underlying RL pipeline.

Static merging.

- **Averaging** [58, 16] merges the three task vectors by element-wise averaging, $\bar{\tau} = \frac{1}{M}\sum_m \tau_m$, and adds the result to the base, $\Theta_0 + \bar{\tau}$. The simplest possible baseline, with no conflict resolution.
- **TIES-Merging** [65] resolves cross-task conflicts in three steps: *trim* (zero-out low-magnitude entries of each τ_m), *elect-sign* (assign each parameter the sign of the task with the largest contribution), and *disjoint-merge* (average surviving entries across tasks). We use trim ratio k=0.8 and scaling 0.5, which perform best on our $\pi_{0.5}$ task vectors.
- **WUDI-Merging** [5] formulates merging as an optimization problem that minimizes residual interference between the merged delta and the per-task deltas, iteratively refining the merged vector via a whitening-style update. We use the v1 variant with 300 iterations following the authors' defaults.
- **TSV-Merging** [18] (Task-Singular-Vector Merging) defines a measure of task interference based on the interaction of singular vectors of different tasks and merges in their joint top subspace. The output is a single fused delta at $1.00\times$ storage; we retain the rank that captures $\sim$95% of the per-task spectral energy.

Dynamic merging. Dynamic methods retain per-task auxiliary information (binary masks, scalars, low-rank residuals, or frequency-domain corrections) and combine it with a shared static merge at inference.

- **TALL-Mask** [50] stores a per-task binary mask $M_m \in \{0,1\}^P$ that selects which positions of the averaged delta are used for task m, and inferences with $\Theta_0 + \bar{\tau} \odot M_m$. Masks are obtained by thresholding $|\tau_m|$ at the per-task top-K=50% default of the original paper.
- **EMR-Merging** [22] extends TALL-Mask by adding a per-task sign modulator and scaling factor that align both the direction and magnitude of the masked delta with each source model: $\Theta_0 + s_m \cdot \bar{\tau} \odot M_m$, where M_m now combines the binary mask and per-element sign and s_m is the per-task rescaler. The modulators and scalars are stored alongside the masks; we follow the original paper without additional top-K thresholding.
- **Twin-Merging** [35] stores a shared mean delta plus a per-task low-rank residual $A_m B_m^\top$ (rank r=16 in our setup) capturing what the mean cannot represent. Inference uses $\Theta_0 + \bar{\tau} + A_m B_m^\top$.
- **Free-Merging** [70] applies a frequency-domain low-pass filter (DCT) on each per-task delta before merging, removing high-frequency components that are mostly task-specific noise. The filtered deltas are stored with per-task binary masks. We use the default DCT cut-off from the original paper.

C Storage Accounting

This section makes the Params column of Table 1 explicit. All ratios are stated relative to one fine-tuned $\pi_{0.5}$ checkpoint (P_{base}=3.61 B parameters in 32-bit floats), with P_{VLM}=3.21 B and P_{act}=0.40 B for the two halves. Two storage atoms are used throughout:

Table 5: Hyperparameters used for every baseline in Table 1. "Storage atom" summarises the per-task auxiliary state introduced beyond the merged base.

Method	Hyperparameter(s) we use	Storage atom (per task)
Static merging		
Averaging	none	—
TIES	trim ratio k=0.8, scaling 0.5	—
WUDI (v1)	300 optimization iterations	—
TSV-Merging	spectral retention $\sim$95%	shared SVD factors
Dynamic merging		
TALL-Mask	per-task top-K=50% mask	binary mask
EMR-Merging	per-task top-K=100% mask + scalar s_m	binary mask + scalar
Twin-Merging	per-task residual rank r=16	low-rank residual
Free-Merging	default DCT cut-off; per-task top-K= $\sim$39% mask	binary mask
Reference		
Per-task FlowSDE	full FlowSDE checkpoint per task (no merging)	full delta

- a *binary mask* costs 1 bit per parameter, i.e. $1/32$ of one 32-bit float at the same denomination;
- a *top-K kept-value* block costs $K \cdot P$ floats.

Per-task scaling factors and per-task low-rank residuals are included where relevant. Table 6 reports the component-wise decomposition for every method.

Static merging. Averaging, TIES, WUDI, and TSV-Merging all replace the base with a single merged delta and contribute $1.00\times$ total storage.

Dynamic merging. Dynamic methods retain a static merged delta on top of the base ($1.00+1.00$ across the first two columns of Tab. 6) plus per-task auxiliary state at inference. *TALL-Mask* adds per-task binary masks of the average delta ($M \cdot 1/32 \approx 0.09$), totalling $2.09\times$. *EMR-Merging* additionally stores per-task sign modulators and scalars, totalling $2.89\times$. *Twin-Merging* attaches per-task low-rank residuals (rank r=16, $M \cdot r(d_{\text{in}} + d_{\text{out}})/P_{\text{base}} \approx 0.25$), totalling $2.25\times$. *Free-Merging* stores per-task frequency-domain modulators (≈ 0.16), totalling $2.16\times$.

NestedVLA. Storage decomposes asymmetrically across the two halves of the backbone. The VLM half is kept per-task as a top-K compressed delta with K_{VLM}=10% (mask plus kept values); the action half is generated by $\mathcal{G}_\phi$ at inference and only its precomputed mask M_m is stored:

$$\text{storage}_{\text{NestedVLA}} \;=\; 1 \;+\; M \cdot \big(K_{\text{VLM}} + \tfrac{1}{32}\big) \cdot \tfrac{P_{\text{VLM}}}{P_{\text{base}}} \;+\; M \cdot \tfrac{1}{32} \cdot \tfrac{P_{\text{act}}}{P_{\text{base}}} \;+\; \tfrac{P_{\mathcal{G}_\phi}}{P_{\text{base}}}. \tag{7}$$

With K_{VLM}=0.10, $P_{\text{VLM}}/P_{\text{base}}$=0.89, $P_{\text{act}}/P_{\text{base}}$=0.11, and $P_{\mathcal{G}_\phi}$=0.43 B, the per-task masks across both halves contribute $M \cdot 1/32 \approx 0.09$ (Mask column of Tab. 6); the per-task VLM kept values together with the shared hypernetwork contribute $M \cdot K_{\text{VLM}} \cdot P_{\text{VLM}}/P_{\text{base}} + P_{\mathcal{G}_\phi}/P_{\text{base}} \approx 0.27 + 0.12 = 0.39$ ($\mathcal{G}_\phi$ column), summing to

$$1 \;+\; 0.09 \;+\; 0.39 \;\approx\; 1.48\times,$$

which we report as $1.5\times$ in Table 1. NestedVLA remains the lowest-storage method among all approaches that condition on task identity, $\sim 0.7\times$ smaller than the strongest dynamic baseline (Free-Merging) and $\sim 1.5\times$ smaller than per-task FlowSDE.

D Skill Compression Study

This section expands the brief discussion of compression operators in §4.3. We compared three operator families on VLA fine-tuning deltas: low-bit quantization, low-rank decomposition via SVD, and drop-and-rescale (top-K magnitude masking with global rescale). All experiments use per-task fidelity evaluation: the compressed delta $\hat{\tau}_m$ is added to Θ_0 and evaluated on the corresponding benchmark m, isolating compression loss from consolidation loss.

Table 6: Bit-level storage decomposition for every method in Table 1, in units of one fine-tuned $\pi_{0.5}$ checkpoint (P_{base}=3.61 B parameters); M=3 tasks. Columns: (i) shared backbone or shared merged delta; (ii) shared low-rank or mean-delta residual; (iii) per-task masks (summed over M); (iv) per-task experts, scalars, or low-rank residuals (summed over M); (v) hypernetwork. "–" indicates the component is not used. Numbers may not sum exactly to the headline due to rounding.

Method	Static	Shared	Mask	Experts / Skills	$\mathcal{G}_\phi$	Total
Per-task FlowSDE	–	–	–	3.00	–	3.00×
Static merging						
Averaging	1.00	–	–	–	–	1.00×
TIES	1.00	–	–	–	–	1.00×
WUDI	1.00	–	–	–	–	1.00×
TSV-Merging	1.00	–	–	–	–	1.00×
Dynamic merging						
TALL-Mask	1.00	1.00	0.09	–	–	2.09×
EMR-Merging	1.00	1.00	0.09	0.8	–	2.89×
Twin-Merging	1.00	1.00	–	0.25	–	2.25×
Free-Merging	1.00	1.00	–	0.16	–	2.16×
NestedVLA (ours)	1.00	–	0.09	∼0	∼0.39	∼1.48×

Table 7: Compression operator comparison on VLA fine-tuning deltas, per-task fidelity evaluation. *Storage*: effective storage relative to the dense τ_m. *Rec. err.*: relative Frobenius error $\|\hat{\tau}_m - \tau_m\|_F / \|\tau_m\|_F$ on VLM and action halves. *Penalty*: drop in `success_once` (MW/MA) or `success_at_end` (CA) vs. FlowSDE. Drop-and-rescale gives the best trade-off with no tuning.

Operator	Storage	Rec. err. (VLM / action)	Penalty (avg)
1-bit quantization	∼6%	0.60–0.96	−71 to −81 pp
Drop-and-rescale + SVD residual	∼15%	0.295 / 0.187	−3 to −5 pp
Drop-and-rescale (ours)	10%	0.300 / 0.190	−3 to −5 pp

D.1 Operator Comparison at Matched Storage

Table 7 compares the three operators at matched effective-storage budgets. **One-bit quantization** (adapted from D-QRELO [29] by storing each parameter as a 1-bit sign with a per-tensor scalar) collapses on VLA deltas: relative reconstruction error reaches 0.60–0.96 across the three tasks, and downstream task success drops by −71 to −81 pp relative to per-task FlowSDE. The failure mode is structural: VLA RL deltas have an approximately zero-mean, near-isotropic per-element distribution (driven by exploration noise and multi-step credit assignment), so a single per-tensor scalar cannot capture within-tensor variance. We verified this pattern holds for both VLM and action halves and for all three benchmarks.

Drop-and-rescale (K=10%, α=1.1, no SVD residual) achieves substantially lower reconstruction errors (0.30 on the VLM half and 0.19 on the action half) and retains $\geq 96\%$ of per-task FlowSDE on ManiSkill and CALVIN. It is the only operator of the three that requires no calibration data.

Drop-and-rescale plus SVD residual extends drop-and-rescale with a per-parameter rank-r SVD residual, in the spirit of "main + residual" designs adapted from D-QRELO. Even at a 10% residual storage budget, the additional fidelity improvement is at most $+0.005$ in absolute reconstruction error (Table 7, last column), while the token budget inflates by an order of magnitude (rank-r residuals require $r \cdot (m + n)$ extra tokens per parameter). We conclude that VLA deltas have no usable low-rank structure beyond the top-K tail and adopt drop-and-rescale alone throughout NestedVLA.

D.2 Keep-Ratio Sweep

Table 8 reports the full keep-ratio sweep $K \in \{0.05, 0.10, 0.15, 0.20\}$ at fixed rescale $\alpha = 1.0$ on all three benchmarks. Three observations stand out.

First, the optimal keep-ratio is task-dependent: ManiSkill peaks at K=0.15 (91.6% `success_once`, +3.4 pp over FlowSDE), while CALVIN peaks at K=0.20 (85.4% `success_at_end`, +4.2 pp over

Table 8: Keep-ratio sweep at fixed rescale $\alpha = 1.0$, per-task fidelity evaluation. Numbers are `success_once` (%) for MetaWorld and ManiSkill; `success_at_end` (%) for CALVIN. The optimal K is task-dependent: ManiSkill peaks at K=0.15, CALVIN at K=0.20; MetaWorld is monotonically improving but never reaches its FlowSDE reference. Per-task FlowSDE references: MW 72.3, MA 88.1, CA 81.3.

Task	K=0.05	K=0.10	K=0.15	K=0.20	FlowSDE
MetaWorld	21.1	48.4	57.4	60.2	72.3
ManiSkill	86.3	87.5	**91.6**	87.5	88.1
CALVIN	63.5	78.1	78.1	**85.4**	81.3

Table 9: Weight-space diagnostics relative to the pretrained reference. Pair columns are concatenated-delta cosine similarities; sign agreement spans three pairs. IF denotes interference.

Group	MA–MW	CA–MW	CA–MA	Sign agreement	IF
Action expert	0.062	0.069	0.106	0.518–0.533	0.379
VLM body	0.011	0.081	0.015	0.504–0.511	0.399
Action interfaces	0.114	0.082	0.073	0.533–0.546	0.373

FlowSDE). MetaWorld is monotonically improving from K=0.05 to K=0.20 but never reaches FlowSDE's 72.3% even at K=0.20 (60.2%).

Second, ManiSkill is non-monotonic in K: K=0.15 outperforms both K=0.10 and K=0.20. We interpret this as a signal-to-noise sweet spot: below 0.15, important parameters are dropped; above 0.15, low-magnitude noise is reintroduced. The same sweet-spot behavior is consistent with the rescale sweep (not shown).

Third, on a per-module basis, the embedding layer is *always* fully dropped at every keep-ratio we tested, while small action-interface layers (action input/output projections) are retained at 44–64%. This structural pattern is consistent across all three benchmarks and motivates module-level top-K selection rather than per-tensor selection. The non-uniform sparsity budget across the network is what drop-and-rescale captures implicitly and what explicit per-tensor compression cannot.

E Supplementary Weight-space Diagnostics

We measure source-model deltas relative to the pretrained checkpoint. For each parameter group, Table 9 reports the cosine similarity of concatenated deltas and sign agreement on valid coordinates. The three-way interference quantity is $\text{IF} = 1 - \|\sum_m \delta_m\|_2 / \sum_m \|\delta_m\|_2$. Action-expert and VLM-body deltas have low pairwise alignment. These statistics describe the source checkpoints; they do not alone establish a causal explanation of rollout failure or the superiority of one objective.

F VLM Static Merging Sanity Test

The main paper confines the hypernetwork to the action expert and keeps per-task VLM storage. This section provides the empirical evidence supporting that design choice. We ran a sanity test in which only the VLM half is consolidated across tasks (via TIES or WUDI), while the action expert remains the GT per-task FlowSDE checkpoint. This isolates the cost of VLM consolidation and answers whether the VLM is consolidatable in principle.

F.1 Setup

For each of the three benchmarks, we constructed a sanity checkpoint by replacing only the VLM portion of the per-task FlowSDE checkpoint with a 3-task-merged VLM. The action expert is the unmodified GT per-task FlowSDE expert. Two merge strategies were tested:

- **TIES VLM:** TIES-Merging [65] applied with trim-rate k=0.8 and scaling 0.5, applied to the VLM half only.

Table 10: VLM static merging sanity test. Only the VLM half is consolidated across the three tasks; the action expert is the GT per-task FlowSDE expert. Penalty is the drop in `success_once` (MetaWorld and ManiSkill) or `success_at_end` (CALVIN) relative to the same-batch FlowSDE reference. Both methods incur penalties exceeding 25 pp on at least two of three benchmarks, with CALVIN collapsing to 0% chain completion.

VLM merge method	MetaWorld	ManiSkill	CALVIN
FT reference (no merge)	66.8	90.0	86.5
TIES VLM (k=0.8, s=0.5)	1.9 (-64.9)	64.7 (-25.3)	0.0 (-86.5)
WUDI VLM (300 iter)	4.7 (-62.1)	80.3 (-9.7)	0.0 (-86.5)

- **WUDI VLM:** WUDI [5] run for 300 optimization iterations on the VLM half.

We did *not* apply mask or rescale corrections, in order to isolate the cost of static VLM consolidation alone. One technical exception: $\pi_{0.5}$ uses tied embedding for `language_model.embed_tokens`, which is not stored in the merged VLM checkpoint; we fall back to per-task FlowSDE for this single tensor (representing 0.17% of VLM parameters). Each sanity checkpoint is evaluated only on its corresponding benchmark; no cross-task transfer is measured.

F.2 Results

Table 10 reports the penalty in success rate relative to the same-batch FlowSDE reference. The penalties are severe across both methods and all three benchmarks. CALVIN is particularly diagnostic: `success_at_end` drops to 0% for both methods, meaning no rollout completes the entire five-step chain. The MetaWorld penalty exceeds -60 pp for both methods, indicating that even strong consolidation strategies destroy the language-instruction routing required to identify the active sub-task within the 50-task suite. Only WUDI on ManiSkill stays within 10 pp of the FlowSDE reference, a single positive cell out of six, far below any usable threshold.

We conclude that static VLM consolidation, even in this favorable setting (action expert is GT), is not viable as a standalone design. The diagnostic also clarifies what kind of information the VLM encodes: cross-task language-instruction grounding, which is highly sensitive to parameter perturbation in a way that action-side representations are not. NestedVLA therefore confines its hypernetwork to the action expert and keeps per-task VLM storage; an extension to also generate VLM deltas via a hypernetwork is left to future work.

Full-model extension (v2). We use v2 to denote a possible extension in which the addressed parameter blocks cover both the VLM and the action expert, so the hypernetwork generates both updates for the selected task. This removes the stored VLM values but substantially enlarges the output space and may require additional generator capacity and metadata. We do not evaluate this extension here; all reported results and storage measurements correspond to v1.

G Loss-Side Ablation

This appendix expands the brief discussion in §4.3 of the three knobs of the training objective in Eq. 4: the loss space (per-tensor standardized vs. absolute), an in-mask magnitude weighting $|\tau_m|^p$ that emphasizes large-magnitude positions, and the out-of-mask penalty weight w_{out} that pulls $\mathcal{G}_\phi$ toward zero on $M_m{=}0$ positions. Table 11 reports five configurations evaluated under the $6{,}000$-step schedule of our default architecture.

Removing the out-of-mask penalty is the most effective lever. Setting $w_{\text{out}} = 0$ improves average success by $+2.4$ pp, the largest single-knob effect we measure. The mechanism is mechanical: positions outside M_m are masked away at inference and contribute nothing to the deployed delta, so penalizing them at train time forces $\mathcal{G}_\phi$ to allocate capacity to a function it does not need. Empirically, the out-of-mask region's loss drifts to $\sim$20 in standardized units when $w_{\text{out}} = 0$, but this drift is absorbed by the inference-time mask and does not affect deployed performance.

Table 11: Loss-side ablations on top of the architecture in Table 2. We vary the loss space (*std*: per-tensor standardized; *abs*: raw absolute), the in-mask magnitude exponent p (each loss term is multiplied by $|\tau_m|^p$), and the out-of-mask penalty weight w_{out}. All runs share the same $6{,}000$-step schedule. Single-seed evaluation; the headline NestedVLA number in Table 1 is the 3-seed average of the same configuration.

Variant	space	p	w_{out}	MW	MA	CA	Avg.
default	std	0	0.05	29.7	90.9	66.7	62.4
w/o out-penalty (w_{out}=0)	std	0	0	**37.9**	88.8	67.7	**64.8**
abs space	abs	0	0.05	23.0	87.8	67.7	59.5
mag-weighting (p=1)	std	1	0.05	27.7	86.9	56.2	57.0
abs + mag p=1	abs	1	0.05	31.2	88.1	70.8	63.4

Loss space and magnitude weighting are non-monotone. The other two knobs each *harm* performance when applied alone: absolute loss space costs -2.9 pp and magnitude weighting at p=1 costs -5.4 pp relative to the default. Yet `abs + mag p=1` recovers most of the loss (-0.6 pp). We interpret this as evidence that absolute and magnitude weighting function as implicit regularizers compensating for the unbounded growth of the out-of-mask region; when the standardized loss already provides scale normalization, neither operator is needed in isolation. The standardized + $w_{\text{out}} = 0$ recipe cleanly beats every other configuration we tested and is our final default.

H Limitations

The current evaluation has four limitations that we mark as future work. **(i) Skill-set scale.** All experiments consolidate M=3 skills; we have not measured how training time, hypernetwork capacity, or output quality scale as M grows by an order of magnitude. **(ii) Adding new skills.** A new skill currently requires extending the per-task patch-embedding table, the per-task multi-head output, and re-training the hypernetwork; we have not demonstrated incremental or low-cost addition of an unseen skill. **(iii) VLM consolidation.** The VLM half is kept per-task as a top-K compressed delta, contributing $\sim$0.35$\times$ of the $1.5\times$ budget; extending the hypernetwork to also generate VLM deltas would lower this floor but is left to future work (App. F shows that naïve static VLM merging is not viable). **(iv) Real-robot evaluation.** All numbers are from simulated benchmarks (MetaWorld, ManiSkill, CALVIN); transfer to physical hardware is not evaluated.

I Broader Impact

NestedVLA reduces the per-skill storage and serving cost of multi-task robotic policies by replacing per-task fine-tuned checkpoints with a single shared parameter generator. Like other efficient deployment methods, lowering this cost can amplify both beneficial uses (low-resource personalisation, on-device multi-skill manipulation, faster transfer between robot platforms) and risks already present in the underlying VLA stack, such as dual-use of robotic policies in unintended physical environments. Our hypernetwork operates entirely on parameter deltas of an existing publicly released backbone ($\pi_{0.5}$); we introduce no new training data or labels, and the deployed behaviour inherits the biases and failure modes of the source FlowSDE checkpoints. All our experiments are simulated; no physical robot is operated in this study. We see no direct path from the parameter-generation framework to a novel misuse scenario beyond those already discussed in the broader VLA literature.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and §1 state four headline claims—parameter-generation framing, $+41$ pp average success over the strongest dynamic merging baseline, matching per-task fine-tuning within 1 pp on ManiSkill, and $1.5\times$ storage—each backed by Tab. 1 and 3 and discussed in §4.2. Limitations are made explicit in §H.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: §H is a dedicated section discussing four limitations: (i) the consolidation experiments are at $M{=}3$ skills only and we do not measure scaling, (ii) adding a new skill requires retraining the hypernetwork, (iii) the VLM half is still stored per-task, contributing $\sim 0.35\times$ of the $1.5\times$ budget, and (iv) all evaluations are simulated.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents an empirical methods contribution and does not include formal theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: §4.1 reports the main training and evaluation settings. App. A (Tab. 4) provides the full hypernetwork architecture, optimizer, schedule, task weights, evaluation protocol, and seed counts. Baselines and their hyperparameters are documented in App. B (Tab. 5). The storage budget reported in Tab. 1 is derived component-wise in App. C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The core NestedVLA algorithm is publicly available at `https://github.com/WanyuGroup/NeurIPS2026-NestedVLA`, with installation instructions and a minimal example. The current release does not include the benchmark-specific training and evaluation pipelines, simulator configurations, or checkpoints needed to reproduce every reported result. The data assets we rely on are public: the $\pi_{0.5}$ backbone, the FlowSDE training pipeline, the RLinf evaluation pipeline, and the three benchmarks (MetaWorld, ManiSkill, CALVIN). Their citations and access points appear at first mention in §4.1 and App. A.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: §4.1 states the backbone, benchmarks, evaluation protocol, and main hyperparameters of the hypernetwork. App. A (Tab. 4) lists every architectural, optimisation, and evaluation choice. Baseline configurations are summarised in Tab. 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All numbers in Tab. 1 and Tab. 3 are averaged over 3 evaluation seeds with fixed initial-state sampling; the typical seed-to-seed standard deviation is $\pm 2.7\%$ (§4.1). Architecture ablations (Tab. 2) report 3-seed averages with std. ± 0.05 on the in-mask loss.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: All hypernetwork runs fit on a single NVIDIA A6000 GPU (48 GB) at $\sim$10 h wall-clock per run, with peak memory 42.6 GB; full details are in App. A (Tab. 4). Per-task FlowSDE checkpoints used as the merging baselines are reused from publicly available recipes and are not re-trained for this paper.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: We have reviewed the NeurIPS Code of Ethics. The work uses publicly available simulated benchmarks and a publicly released VLA backbone; it involves no human subjects, scraped data, or sensitive information. Anonymity is preserved at submission time.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: A short Broader Impact statement is included in App. I: lower per-skill storage and serving cost can amplify both beneficial uses (low-resource personalisation, multi-skill robotic deployment) and downstream risks already present in the underlying VLA backbone (e.g., dual-use of robotic policies). NestedVLA inherits the biases of the source checkpoints and introduces no new training data.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk model, dataset, or generative system. The base $\pi_{0.5}$ backbone is publicly available with its own release terms, and we add only a parameter generator that targets that backbone in a simulated benchmark setting.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All used assets are cited at first mention: the $\pi_{0.5}$ backbone [43], the FlowSDE training pipeline [4], the RLinf evaluation pipeline [67], and the three benchmarks (MetaWorld [38], ManiSkill [48], CALVIN [39]). Each is used in accordance with its public release terms.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released algorithm code is documented at `https://github.com/WanyuGroup/NeurIPS2026-NestedVLA`, which provides installation instructions, a minimal example, the scope of the release, and its license. The architecture, training schedule, and evaluation protocol are further documented in App. A.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were used only for incidental writing and editing assistance. They are not part of the core method, data pipeline, or scientific contribution: the proposed hypernetwork is a small task-conditioned generator over compressed task-vector patches, with no language-model component beyond the frozen $\pi_{0.5}$ VLM backbone.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.