MoS-VLA: A Vision-Language-Action Model with One-Shot Skill Adaptation

Anonymous authors
Paper under double-blind review

Abstract

Vision-Language-Action (VLA) models trained on large-scale robotics datasets promise robust control across diverse domains and embodiments, yet they frequently fail zero-shot in novel scenarios. We introduce Mixture-of-Skills VLA (MoS-VLA), a framework that formulates robot manipulation policies as continuous linear combinations of learned basis functions. Pretrained jointly on the Open X-Embodiment data mixture, MoS-VLA constructs a skill space for rapid adaptation. At test time, adaptation in our evaluated robot settings uses a single expert demonstration. We infer the optimal mixture coefficients via a lightweight, gradient-free convex optimization that minimizes L^1 action error. MoS-VLA achieves 70–100% success in simulated and real-robot tasks, where baseline success is 0% in all but one comparison. It also achieves lower action-prediction error than two pretrained VLAs and two in-context baselines on their evaluated datasets, while adapting at a fraction of the compute of gradient-based finetuning.

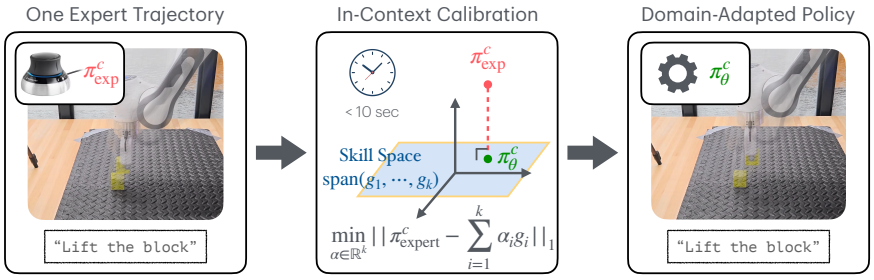

Figure 1: **In-context Adaptation with Function Encoders.** (*Left*) A human expert collects one trajectory of data in the new context. (*Middle*) The model is calibrated by solving a small, least absolute error optimization problem. This process takes only a few seconds on an RTX 3090. (*Right*) The adapted model autonomously completes the task in the new environment.

1 Introduction

Inspired by the success of large language models, modern robotics aims to achieve generalization and human-like performance through the use of internet-scale data and large, attention-based architectures. To this end, researchers have collected enormous datasets of robotic arm trajectories (Open X-Embodiment Collaboration et al., 2024) and trained so-called vision-language-action foundation models to map natural language task descriptions and state observations to robot actions (Kim et al., 2024; Octo Model Team et al., 2024; Brohan et al., 2023b;a; Ma et al., 2024). While these models achieve acceptable performance on in-distribution tasks and lab settings, they are so far brittle to out-of-distribution data. The key challenge is that there is no preexisting dataset of robotic demonstrations analogous to the enormous amount of natural language data present on the internet, and collecting data in robotics settings is expensive and difficult to scale. Thus, there is currently not enough data to achieve generalization through scale alone. Therefore, creating a generalist robotic policy inherently requires a more principled approach to transfer.

The current state-of-the-art approach to transfer is to finetune a pretrained vision-language-action model on a (relatively) small calibration dataset collected on the new task and/or lab setting. Often, this calibration dataset consists of tens or hundreds of expert demonstrations, and finetuning the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

models requires significant resources. Even parameter-efficient finetuning techniques (e.g., LoRA, QLora) (Han et al., 2024; Hu et al., 2022; Dettmers et al., 2023) inherently require significant resources, as numerous forward and backward passes are needed during gradient descent.

In contrast, we present a lightweight adaptation method based on function encoders. The function encoder algorithm learns a set of neural basis functions to span a space of interest. In this case, they span the space of policies, with each policy specialized to a certain lab setting. After training, a new policy for a new lab setting is computed as a linear combination of the basis functions, where the coefficients are calculated as the solution to a least absolute error regression problem. Because the optimization space is small and well-structured, only a tiny calibration dataset consisting of one expert trajectory is needed. Furthermore, the least absolute error problem is just a linear program written in terms of the basis function outputs. Therefore, model adaptation has low overhead: only forward passes and a small linear-program solve are required. Consequently, we demonstrate the ability to adapt to a new lab setting given one expert demonstration in only a few seconds.

We empirically verify our approach by building on top of OpenVLA (Kim et al., 2024) and training on the OXE dataset (Open X-Embodiment Collaboration et al., 2024). We demonstrate that our approach achieves improved accuracy on the training and evaluation datasets. Additionally, we evaluate the policies on a Franka Panda robot arm in unseen lab settings in both simulation and the real world. OpenVLA, $\pi_{0.5}$ (Physical Intelligence et al., 2025), and ICRT (Fu et al., 2025) achieve 0% success across all tasks, while X-VLA (Zheng et al., 2026) achieves 15% on simulated door opening and 0% elsewhere. In contrast, MoS-VLA achieves 70–100% success across all tasks after calibration with a single expert trajectory.

Main Contributions:

- We introduce MoS-VLA, a VLA framework pretrained to enable rapid in-context adaptation. In our robot evaluations, it adapts from a single expert demonstration in seconds by fitting mixture coefficients without gradient updates.
- We demonstrate that the function encoder algorithm is applicable to billion-parameter architectures and mixed vision-language datasets.
- We empirically show that MoS-VLA outperforms zero-shot and in-context baselines on their evaluated robotic datasets and in real-world experiments. It also uses a fraction of the adaptation compute and memory required by gradient-based finetuning.

2 Related Work

Foundation Models for Robotics A recent paradigm for robotics seeks to replicate the successful formula of language models: internet-scale data combined with billion-parameter attention-based architectures. The Open X-Embodiment dataset is a massive collection of robotic data designed to mimic an internet-scale training regime (Open X-Embodiment Collaboration et al., 2024). Numerous works train large models using this data, which may be diffusion models (Octo Model Team et al., 2024) or auto-regressive (Kim et al., 2024), and often incorporate pretrained vision and language models such as Llama 2 (Touvron et al., 2023), Google-T5 (Raffel et al., 2020), SigLIP (Zhai et al., 2023), and DinoV2 (Oquab et al., 2024). Despite the magnitude of resources required, they empirically fail to generalize to out-of-distribution tasks and lab settings. Furthermore, given the expense of robotic data collection and the diminishing returns implied by the so-called scaling laws (Kaplan et al., 2020), robotic foundation models require more sophisticated and principled methods of transferring to new tasks and settings. We present one such method based on transfer within the span of a set of neural network basis functions (Ingebrand et al., 2025).

Adaptation and In-Context Learning The principal method for adaptation is to finetune the foundation model on a relatively small dataset collected from the new setting and/or task (Kim et al., 2025). While this dataset is typically orders of magnitude smaller than the pretraining dataset, it may still consist of many trajectories over many tasks, presenting a significant overhead. Furthermore, given the size of foundation models, full finetuning requires a powerful machine comprising numerous GPUs. Parameter-efficient finetuning methods such as LoRA are used to reduce the finetuning overhead at the cost of expressivity and performance (Han et al., 2024; Hu et al., 2022; Wang

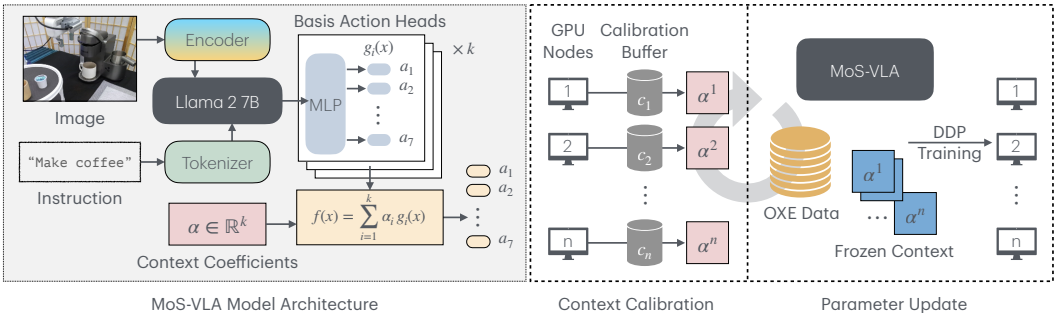

Figure 2: **The MoS-VLA Architecture and Training Pipeline**. *(Left)* MoS-VLA employs the OpenVLA backbone where images and text are processed by the Llama 2 model. The language modeling head is replaced with k basis action heads. Their action predictions are linearly combined as described in Section 4.2. *(Right)* During training, we iterate between context calibration and parameter updates, utilizing hardware parallelism in both stages. For calibration, each node computes and broadcasts the coefficient only for the datasets it hosts. During model updates, the context coefficients are held frozen and trainable weights are adjusted with DDP.

et al., 2025). Nonetheless, any method that leverages gradient descent requires significant overhead, both in computation and memory.

Rather than computing gradient updates, ICRT (Fu et al., 2025) appends expert demonstrations as additional input, in the style of in-context learning for language models (Dong et al., 2024; Min et al., 2022). As is typical for transformers, this method is quadratic in both memory and compute with respect to the expert trajectory length. In contrast, our method treats the expert trajectory as a *set* of input-output pairs rather than a *sequence*, removing the need to compute attention between all data pairs. Hence, feature extraction scales linearly with demonstration length, while inference cost after calibration is independent of that length. X-VLA (Zheng et al., 2026) conditions a shared backbone on learned per-embodiment soft prompts, inferred via gradient descent at adaptation time. Other recent few-shot adaptation methods include retrieval-augmented in-context agents (Sridhar et al., 2025a), keypoint-based policies (Haldar & Pinto, 2025), and RICL (Sridhar et al., 2025b).

Skill Learning and Task Representations Prior methods, especially from the reinforcement learning (RL) community, learn diverse skills and task representations. These methods often leverage information-theoretic proxy objectives such as cross entropy (Eysenbach et al., 2019; Hausman et al., 2018). Other approaches investigate how to best leverage pretrained skills for downstream tasks (Rhim et al., 2022). Meta-RL aims to learn policies that can quickly adapt to new tasks (Lan et al., 2019), while zero-shot RL uses offline datasets to learn spaces of task solutions that do not require extra training at inference time (Touati & Ollivier, 2021; Ingebrand et al., 2024). Our method is similar in that we learn a space of policies, although we only leverage supervised learning. Additionally, our space of policies encodes different behaviors *and* different contexts due to the visual variations between lab settings. Lastly, mixture of experts is a common technique for foundation models that creates sub-networks specialized to certain tasks. Each task is routed to a different sub-network via a learned gating mechanism (Cai et al., 2025; Puigcerver et al., 2024). The intention is to lower the inference-time computation since only a subset of the total parameters is active. Hence, only one or a few experts are active for any given input. In contrast to learning a finite *set* of experts, our method trains basis functions to work in tandem, creating a *space* of policies. See Appendix D for a thought experiment motivating the use of continuous policy spaces over discrete sets.

3 Background

The function encoder algorithm (Ingebrand et al., 2025) learns basis functions for an arbitrary function space. Given a Hilbert space $\mathcal{H} = \{f : X \to Y\}$ with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and an induced norm $|| \cdot ||_{\mathcal{H}}$, function encoders learn a set of neural network basis functions $\{g_1, ..., g_k\}$, where k is a hyper-parameter, to approximate functions in $\mathcal{H}$. A function $f \in \mathcal{H}$ is represented as a linear combination of the basis functions, $f \approx \sum_{i=1}^{k} \alpha_i g_i$, where $\alpha \in \mathbb{R}^k$. Furthermore, given the

basis functions and a function f, α is calculated as the solution to the least-squares minimization, $\alpha := \arg\min_{\alpha \in \mathbb{R}^k} ||f - \sum_{i=1}^{k} \alpha_i g_i||_{\mathcal{H}}^2$.

The training procedure requires a set of datasets, $\mathcal{D} = \{D_1, ..., D_n\}$, where each dataset $D_i = \{x_j, f_i(x_j)\}_{j=1}^m$ is a set of input-output pairs for a function $f_i \in \mathcal{H}$. The training procedure uses gradient descent. At each gradient step, the coefficients α for function f_i are calculated by solving the least-squares minimization. Then, the function is approximated as a linear combination of the learned basis, and the loss is the error of this approximation. See Ingebrand et al. (2025).

In contrast to the typical function encoder definitions, we will leverage Banach spaces and L^1 optimization rather than L^2 optimization. We define the coefficients as

$$\alpha := \arg\min_{\alpha \in \mathbb{R}^k} ||f - \sum_{i=1}^{k} \alpha_i g_i||_1. \tag{1}$$

We use least absolute error for robustness to the heavy-tailed outliers in OXE. L^1 down-weights outliers while preserving precise control. We validate this in Appendix E. The only practical difference is the optimization procedure to compute the coefficients, where we solve a linear program instead of an unconstrained quadratic program. The rest of the algorithm remains unchanged.

4 METHOD

4.1 PROBLEM FORMULATION

We consider the following setup. The robot observes its environment through RGB images of a fixed size, denoted by the set I. Tasks are described in natural language and drawn from a set T. For simplicity, both I and T are assumed finite, representing the (large but bounded) number of images and tasks the robot may feasibly encounter. A robot's state is defined as a pair $s \in S = I \times T$, consisting of an image and a task description. The robot's action space is denoted $A \subset \mathbb{R}^m$, where each output dimension corresponds to a robot end effector's translation, rotation, etc. A trajectory $\tau = \{(s_t, a_t)\}_{t=1}^H$ is a sequence of states and actions of length H. Each trajectory is collected under a fixed but potentially partially observed context $c \in C$, which captures factors such as lighting, camera position, robot morphology, decision rate, etc. We write τ^c to highlight that a trajectory was collected under context c, and we emphasize that the context is relevant for decision making, but it may not be fully identifiable from a single observation. Lastly, we assume there exists an expert policy $\pi_{exp} : S \times C \to A$, which is optimal with respect to some objective function. The expert policy, typically a human, is assumed to take the context into consideration in its decision making.

In the zero-shot VLA setting, a policy $\pi_\theta : S \to A$ predicts expert actions from observed states alone. When different contexts require different actions for the same observed state, this input is insufficient to distinguish the appropriate behavior. Our formulation additionally provides an expert trajectory from the deployment context. MoS-VLA uses this trajectory to estimate mixture coefficients, adapting to the new context while keeping the pretrained model weights fixed.

Next, we will define the data assumptions. Let $D^c = \{\tau_1^c, \tau_2^c, ...\}$ be a set of expert trajectories from a common dataset source. During pretraining, dataset source serves as a practical proxy for context, although a source may include multiple tasks, sessions, or configurations. Let $\mathcal{D} = \{D^{c_1}, D^{c_2}, ...\}$ be a set of such datasets. For this paper, $\mathcal{D}$ is the Open X-Embodiment data mixture.

The in-context adaptation problem consists of two stages: First, there is an offline training period for π_θ using $\mathcal{D}$, where training time is presumed not a concern. Second, there is an online adaptation phase. The trained model π_θ is provided with one additional trajectory τ_{exp}^c collected by the expert policy under a context c. Typically, c is an unseen context that was not present in the offline training dataset. Then π_θ outputs an action a for a new state s, and should reproduce the behavior of π_{exp}. Thus, π_θ should use the additional information provided by τ_{exp}^c to improve its accuracy in this new context. In this second phase, adaptation time should be minimized, ideally to seconds or less.

4.2 MAIN IDEA

To design a policy that adapts to the context given the trajectory τ_{exp}^c, we will represent context-dependent expert policies within a shared function space, and use τ_{exp}^c to identify a function within



Figure 3: **Illustration of Simulated and Real Robot Tasks.** We evaluate MoS-VLA on five tasks in both simulated and on-robot experiments. From left to right: Robosuite block lifting, Robosuite open door, goal reaching, block lifting, pen insertion. See Appendix F for MoS-VLA trajectories.

this space. Formally, let $\pi^c_{exp}(\cdot) = \pi_{exp}(\cdot, c)$ be the expert policy conditioned on a context c, where this notation makes the dependence on the context implicit. Consider the set of such policies $\Pi_{exp} = \{\pi^c_{exp} | c \in \mathcal{C}\}$. We view these policies as elements of the ambient Banach space $L^1(S, \mu; \mathbb{R}^m)$, where μ is a reference distribution over states and the norm is $\|f\|_1 := \int_S \sum_{i=1}^m |f_i(s)| \, d\mu(s)$ (see Oden & Demkowicz (2018)). Using the function encoder algorithm, we learn neural basis functions $\{g_1, \ldots, g_k\}$ whose span defines a finite-dimensional approximation space for the expert policies, with $\pi^c_{exp} \approx \sum_{i=1}^k \alpha^c_i g_i$. During the online adaptation phase, we compute $\alpha^c \in \mathbb{R}^k$ by a least absolute error optimization problem. Once we have these coefficients α^c, we may approximate π^c_{exp} for any new state s using a linear combination of the basis functions.

A key advantage of this approach is its simplicity during the online phase. The function encoder adapts to new contexts via the least absolute error calculation, which has two simple steps. First, we must forward pass the basis functions on each state s in τ^c_{exp}. Note that we can forward pass all states in parallel. Second, we solve the least absolute error problem (1), which is effectively just a linear program solvable in a negligible amount of time on a CPU. The function encoder's overhead for in-context adaptation is minimal. Crucially, no backpropagation is required. Thus, our method adapts to new contexts in the same amount of time required to forward pass the model, which is only seconds even for relatively weak GPUs. Furthermore, once we have computed the coefficients α^c, we output actions as a linear combination of the basis functions. Therefore, the computational and memory costs after calibration are constant with respect to the expert trajectory length.

4.3 Architecture

To make a VLA-style function encoder, we require k separate output actions conditioned on the state, image, and language instruction, where k is a hyperparameter. Due to the large number of parameters in VLA models, creating k fully independent copies is infeasible. Instead, we leverage a single network body with multiple randomly-initialized output heads. In addition, we utilize the parallel decoding technique in OpenVLA-OFT to model all action dimensions simultaneously. The output of each head acts as a separate basis function, and each outputs an action. See Figure 2. For simplicity, we use OpenVLA Kim et al. (2024) as the base architecture, although the same ideas can be applied to any model. Note that the original OpenVLA implementation uses the Llama 2 head rather than training a new one from scratch. Thus, it outputs a distribution over Llama 2's vocabulary, which contains 32,000 tokens, and the least-used 256 tokens are replaced with action tokens. In contrast, each basis function only outputs scalar action values directly. Thus, despite the extra action heads, MoS-VLA uses only 0.2% more parameters than OpenVLA. In general, the overhead in terms of parameter count is small, and this same procedure can be applied to other foundation models as well.

4.4 Training Procedure

We randomly initialize the k basis-function heads and train them. The OpenVLA body is fine-tuned from its pretrained checkpoint via LoRA for the results in this paper, though full finetuning is also possible. Training progresses according to the modified function encoder algorithm using L_1 coefficient calibration. See Appendix B for the pseudocode.

Starting with a pretrained OpenVLA is a design choice to reduce the length of the offline training period. In principle, we could instead start with pretrained vision and language models, as other VLA models do (Bjorck et al., 2025), and train the basis functions from there. Doing so may

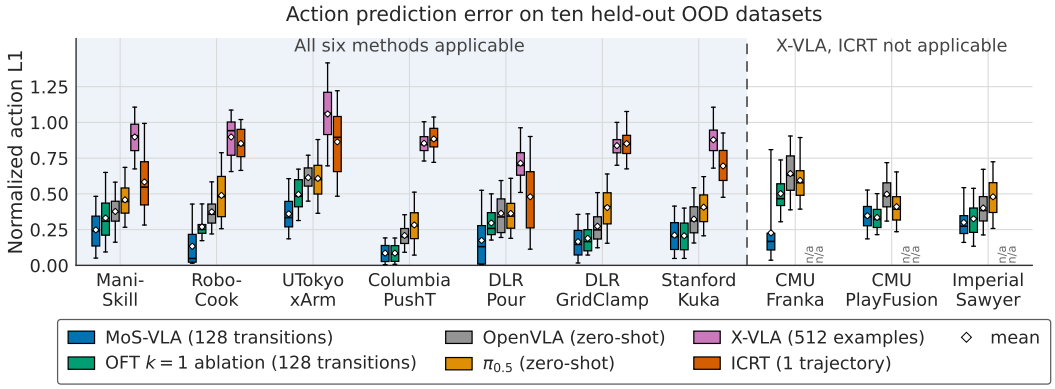

Figure 4: **Action prediction error across ten held-out datasets.** MoS-VLA, calibrated on 128 observation-action pairs, achieves lower mean action error than OpenVLA and $\pi_{0.5}$ on all ten OOD datasets and than X-VLA and ICRT on their evaluated subsets. We use 128 transitions for consistency across datasets. X-VLA and ICRT use separate evaluation episodes and are omitted on three datasets because their absolute EEF action-space assumptions are incompatible with those subsets. OFT ($k = 1$) is a calibrated single-head OpenVLA-OFT model trained on the same data, providing an ablation of the multiple basis heads; MoS-VLA outperforms it on eight of ten datasets.

improve the performance, as the pretrained OpenVLA model may trap the model's performance at a local optimum given that OpenVLA was trained on classification instead of regression. We leave this for future work.

Beyond directly parameterizing low-level actions, MoS-VLA can be extended to operate in a latent action space. Rather than learning linear combinations of instantaneous motor commands, the basis functions may generate latent skills corresponding to temporally extended behaviors, which are then decoded into action sequences (i.e., action chunking). This formulation may increase representational power by enabling nonlinear, high-level compositions of behaviors while also naturally supporting temporal abstraction. See Appendix H for this extension.

4.5 Theoretical Perspective

For an expert trajectory $\{(s_t, a_t)\}_{t=1}^{N}$, MoS-VLA estimates the context coefficients by solving

$$\min_{\alpha \in \mathbb{R}^k} \sum_{t=1}^{N} \left\| a_t - \sum_{i=1}^{k} \alpha_i g_i(s_t) \right\|_1 .$$

For fixed basis predictions, this is standard least-absolute-error regression. Introducing nonnegative slack variables for the action residuals converts the objective into a linear program with inequality constraints. Consequently, calibration is convex and admits a globally optimal solution using established linear-programming algorithms; it does not require backpropagation. The theory follows directly from prior work on function encoders and approximation in normed function spaces. In particular, before output clipping, unconstrained linear combinations of the learned basis functions define a finite-dimensional subspace of the ambient L^1 space, and deployment calibration minimizes empirical L^1 error within this subspace. Thus, MoS-VLA does not require a new optimization-theoretic result; it applies existing function-approximation and linear-programming theory. Under suitable boundedness and sampling assumptions, Rademacher-complexity bounds provide a framework for relating empirical calibration error to population error (Bartlett & Mendelson, 2002).

5 Experiments

Implementation Details. We train 16 basis functions by finetuning OpenVLA on the OXE dataset as described in Section 4. We utilize 32 distributed compute nodes, each of which has a GH200, using data parallelism (DDP) (Li et al., 2020). We train for a total of 100,000 gradient steps, which

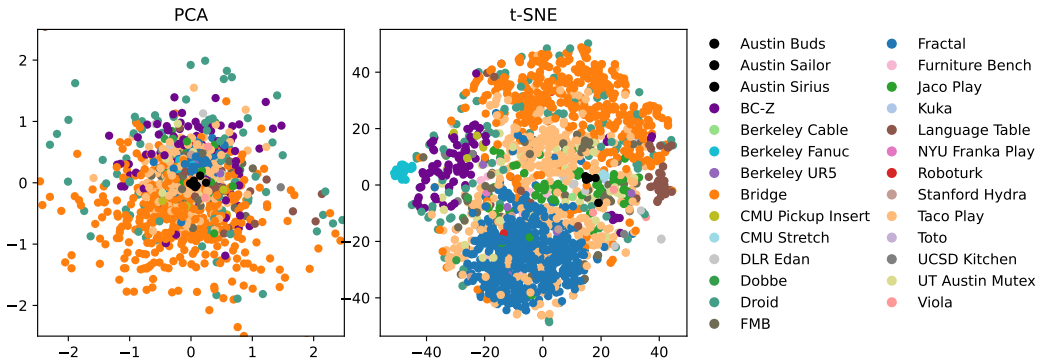

Figure 5: **Visualization of the Coefficients.** Coefficients fitted from expert trajectories across dataset sources are visualized using PCA and t-SNE. The projections reveal source-dependent structure with substantial overlap, illustrating how different contexts are organized within the learned coefficient space. The first two principal components show substantial overlap across sources. The t-SNE projection, by contrast, shows clear source-level clusters, indicating that dataset identity is captured in the local neighborhood structure of the coefficient space even though it does not dominate the leading principal components. Three Austin datasets with similar lab setups are highlighted in black; their proximity in the projections suggests shared structure in their fitted representations.

		Simulation ($m = 20$)		**On-robot** ($m = 10$)		
Method	N-Shot	Lift Block	Open Door	Reach	Lift Block	Insert Pen
OpenVLA	0	0%	0%	0%	0%	0%
$\pi_{0.5}$	0	0%	0%	0%	0%	0%
ICRT	5	0%	0%	0%	0%	0%
X-VLA	5	0%	15%	0%	0%	0%
MoS-VLA	1	70%	75%	100%	100%	100%

Table 1: Success rate comparison between MoS-VLA and baselines.

takes approximately 24 hours. In order to fairly compare against OpenVLA, we use the same *Open-X Magic Soup Plus* data mixture with a global batch size of 320. We use an Adam optimizer with a learning rate of $1e^{-4}$ and a warmup of 10 steps.

Scalable Skill Calibration. In the standard function encoder algorithm, the coefficients for each context are computed every training step using a batch of training data, and gradients are back-propagated through this computation. This procedure requires a reasonably large batch size to ensure stable estimation of the context coefficients, which is practically infeasible for VLA training. For scalability to distributed training of a large model, we instead maintain a calibration buffer with a capacity of 512 observation-action pairs for each dataset. During DDP training, the calibration buffers are evenly spread across computation nodes. In our case, with 32 nodes and 27 datasets, the first 27 nodes each host a unique dataset.

We calibrate every 16 gradient steps: nodes hosting the calibration buffer compute and broadcast the coefficients across all nodes. After each calibration, the coefficients are held frozen, and gradients are not back-propagated through this process. We find that this implementation is memory efficient since it does not have to solve a linear program at each gradient step, but it does not negatively impact performance. Additionally, it allows us to easily maintain the data sampling ratio from the data mixture, whereas the standard function encoder approach would not. We solve the linear program corresponding to (1) via CVXPY (Diamond & Boyd, 2016).

5.1 Results

L^1 Error on Datasets. After training, we evaluate our model against OpenVLA, $\pi_{0.5}$, X-VLA, ICRT and the calibrated OFT $k = 1$ ablation (Kim et al., 2024; Physical Intelligence et al., 2025; Zheng et al., 2026; Fu et al., 2025; Kim et al., 2025) on both in-distribution and out-of-distribution datasets, with the out-of-distribution results shown in Figure 4. Crucially, these datasets represent

Method	N = 32			N = 128			N = 512		
	L^1	Time (s)	VRAM (GB)	L^1	Time	VRAM	L^1	Time	VRAM
MoS-VLA (Ours)	0.274 ± 0.18	1.5	17.0	0.262 ± 0.18	2.8	17.1	0.262 ± 0.18	14.1	17.6
LoRA Discrete	0.271 ± 0.18	158.8	33.9	0.245 ± 0.16	82.6	35.4	0.214 ± 0.16	251.2	35.4
LM Head Only	0.399 ± 0.24	72.6	18.4	0.375 ± 0.23	76.6	19.7	0.345 ± 0.22	78.4	19.7
OpenVLA-OFT	0.262 ± 0.14	314.1	36.7	0.264 ± 0.15	276.8	50.1	0.235 ± 0.13	266.8	49.9

Table 2: **Adaptation Efficiency.** Comparison of adaptation time, peak GPU memory, and action-prediction error across calibration budgets N. Errors are mean $\pm$ standard deviation over validation transitions pooled across ten OOD datasets. LoRA Discrete finetunes OpenVLA's discrete-action model; LM Head Only updates only its output head; OpenVLA-OFT finetunes a continuous-action model. MoS-VLA achieves the lowest adaptation time and peak GPU memory across all budgets.

entirely unseen lab conditions and thus serve as a proxy for on-hardware performance: low error is likely necessary, but not sufficient, for successful on-hardware deployment. We compute the context coefficients for each subset from 128 observation-action pairs from its training split. Calibration and evaluation use separate episodes. Actions are normalized using training-split statistics, and predictions are clipped to $[-1, 1]$. We report mean absolute error over the seven action dimensions and validation transitions within each dataset.

While OpenVLA achieves similar performance on training datasets, it is significantly worse when evaluated out of distribution. One possible explanation is *mixed-context overfitting*, illustrated in a simplified setting in Appendix A. It occurs when an over-parameterized model, lacking mechanisms to distinguish between contexts, is trained on a mixed-context dataset. In such cases, the model learns a single function to approximate many contexts simultaneously, resulting in sharp, unstable outputs across contexts. Note that this type of overfitting occurs even if the data itself is noise-free. In contrast, using the function encoder algorithm mitigates this type of overfitting because there is effectively a different model for each context.

Computational Efficiency The key benefit of MoS-VLA is the low overhead in terms of compute and memory during adaptation. To demonstrate this, we compare our method against LoRA finetuning of OpenVLA and OpenVLA-OFT. For each finetuning baseline, we report the error and accumulated training time at the checkpoint with the lowest validation error among ten evaluations over 1,000 updates, excluding validation pauses. MoS-VLA's adaptation time includes forward passes and coefficient fitting. This experiment is conducted over the same 10 OOD datasets in Figure 4. The results in Table 2 indicate that MoS-VLA is significantly more efficient in computation and VRAM usage. The closest baseline is finetuning only the action head of an OpenVLA model, but this causes the performance to degrade.

Additionally, we leverage the validation splits of the training datasets to visualize the learned function space. For each task across all lab settings, we compute the corresponding skill coefficients and apply dimensionality reduction techniques to reveal the structure of the learned landscape. As shown in Figure 5, the coefficients form clusters that align with lab settings, supporting our hypothesis that each lab environment is effectively captured by a unique function. Notably, three Austin datasets, treated as separate during training, naturally cluster together, reflecting their underlying similarity in lab conditions. These results highlight the potential of the learned skill space to provide interpretability and expose similarities across datasets.

Policy Performance. To further evaluate MoS-VLA, we conduct simulated and real world experiments with the Franka Panda robot arm on a suite of tabletop manipulation tasks. For simulation, we use two tasks from Robosuite (Zhu et al., 2020): block lifting and door opening. For on-robot experiments, we solve three tasks: goal reaching, block lifting, and pen insertion. Figure 3 illustrates the tasks. Forward/backward motion is disabled in Robosuite block lifting, door opening, and pen insertion. In all experiments, the model is calibrated using only one expert trajectory. See Appendix F for detailed descriptions.

Our robot evaluation results are shown in Table 1. We find that the base OpenVLA model and $\pi_{0.5}$ are brittle to unseen environments and completely fail, both in simulation and on the real robot. In contrast, by calibrating with a single expert demonstration, MoS-VLA adapts to the new settings. In an additional simulated UR5E block-lifting experiment with a 45-degree camera view, MoS-VLA achieves 17/20 successes after calibration with one expert trajectory, suggesting applicability across

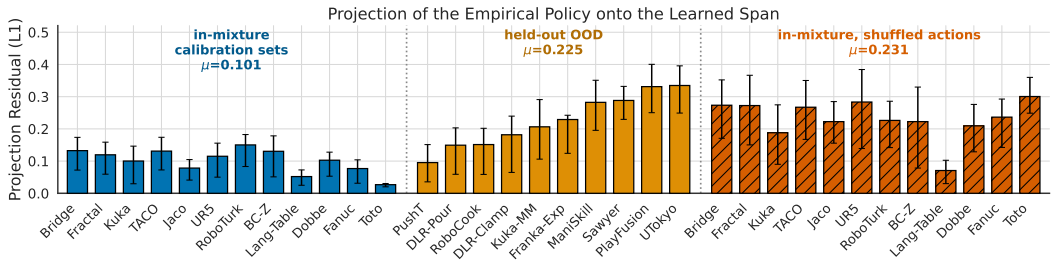

Figure 6: Comparison of projection residuals on a subset of in-distribution datasets (blue), OOD datasets (orange), and shuffled action datasets (red).

robot embodiments and camera viewpoints (Appendix F). Critically, ICRT and X-VLA receive five demonstrations but achieve 0% success in all settings except X-VLA's 15% on simulated door opening, highlighting the difficulty of adaptation in these environments.

L^1 Projection Residual. We investigate whether expert policies in unseen contexts are well approximated by the linear span of the learned basis. To assess this empirical representability, we measure the L^1 residual on subsets of in-distribution, out-of-distribution, and randomized datasets. The in-distribution datasets provide a reference for approximation quality on familiar contexts. The randomized datasets are constructed by shuffling actions, disrupting the observation-action correspondence, and providing a contrasting reference.

Lastly, we consider the OOD datasets. Some exhibit residuals comparable to the in-distribution reference, suggesting that the learned basis accommodates their demonstrated behavior, while others exhibit larger residuals approaching the shuffled-action reference. These residuals provide an empirical diagnostic of how well the learned span represents new demonstration data. See Figure 6.

6 Limitations

While MoS-VLA demonstrates clear advantages in our test environments, its evaluation is limited to relatively short-horizon tasks. MoS-VLA does not address the problem of long-horizon manipulation via mechanisms such as temporal abstraction or hierarchical skill composition. However, these mechanisms are orthogonal to MoS-VLA and as such any long-horizon advances can be incorporated into MoS-VLA. Moreover, environments with high stochasticity, such as the simulated block lifting task, may demand more than one demonstration to achieve consistent success because the set of observations has higher variability.

7 Conclusion

We introduced a novel method for training VLA models to form a structured skill space that enables one-shot domain adaptation. Adaptation is achieved via least absolute error optimization, requiring no gradient calculations. This design makes the approach especially suitable for hardware-constrained settings where finetuning is impractical. Empirically, we demonstrate that our method improves performance on unseen robot experiments, both in simulation and on real hardware, outperforming a pretrained OpenVLA model that lacks domain transfer capability.

This work represents another step toward in-context adaptation of robot foundation models. We argue that finetuning robot policies for every new domain remains a major barrier to the deployment of general-purpose robotics. Future directions include exploring alternative functional spaces, training schemes, and architectural choices. By building expandable skill spaces, our paradigm opens the door to VLA models that can rapidly acquire new skills without erasing prior knowledge.

AI USE STATEMENT

In this work, we used AI to generate code and to revise the text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Code is provided in the supplementary material.

REFERENCES

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, 2002.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith LLontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: an open foundation model for generalist humanoid robots. 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CORL*, 2023a.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong T. Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023b.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Trans. Knowl. Data Eng.*, 37(7):3896–3915, 2025.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *NeurIPS*, 2023.

Steven Diamond and Stephen Boyd. Cvxpy: a python-embedded modeling language for convex optimization. *J. Mach. Learn. Res.*, 2016.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, and Zhifang Sui. A survey on in-context learning. In *EMNLP*, pp. 1107–1128. Association for Computational Linguistics, 2024.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019.

Max Fu, Huang Huang, Gaurav Datta, Lawrence Yunliang Chen, Will Panitch, Fangchen Liu, Hui Li, and Ken Goldberg. ICRT: in-context imitation learning via next-token prediction. In *ICRA*, pp. 5937–5944. IEEE, 2025.

Siddhant Haldar and Lerrel Pinto. Point policy: Unifying observations and actions with key points for robot manipulation. *CORL*, 2025.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. 2024.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin A. Riedmiller. Learning an embedding space for transferable robot skills. In *ICLR*, 2018.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Tyler Ingebrand, Amy Zhang, and Ufuk Topcu. Zero-shot reinforcement learning via function encoders. In *ICML*, 2024.

Tyler Ingebrand, Adam J. Thorpe, and Ufuk Topcu. Function encoders: A principled approach to transfer learning in hilbert spaces. *ICML*, 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *CORL*, 2024.

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

Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. In *IJCAI*, pp. 2794–2800, 2019.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, 2020.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision-language-action models for embodied AI. 2024.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *EMNLP*, pp. 11048–11064. Association for Computational Linguistics, 2022.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, 2024.

J. Tinsley Oden and Leszek F. Demkowicz. *Applied Functional Analysis*. CRC Press, third edition edition, 2018.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim,

Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vikash Kumar, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. *IEEE ICRA*, 2024.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *Trans. Mach. Learn. Res.*, 2024, 2024.

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

Joan Puigcerver, Carlos Riquelme Ruiz, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *ICLR*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Jungsub Rhim, Eunseok Yang, and Taesup Kim. Efficient task adaptation by mixing discovered skills. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML*, 2022.

Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. REGENT: A retrieval-augmented generalist agent that can act in-context in new environments. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. RICL: Adding in-context adaptability to pre-trained vision-language-action models. In *9th Annual Conference on Robot Learning*, 2025b.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. In *NeurIPS*, pp. 13–23, 2021.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. 2023.

Luping Wang, Sheng Chen, Linnan Jiang, Shu Pan, Runze Cai, Sen Yang, and Fei Yang. Parameter-efficient fine-tuning in large models: A survey of methodologies. *Artificial Intelligence Review*, 2025.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *ICCV*, pp. 11941–11952. IEEE, 2023.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Tai Wang, Ya-Qin Zhang, Jingjing Liu, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

A Mixed-Context Overfitting

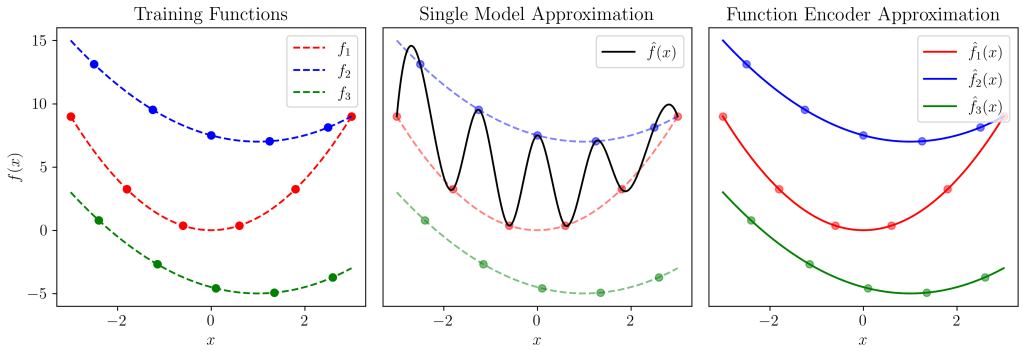

Figure 7: **Visualization of the Mixed-Context Overfitting Phenomenon.** *(Left)* The two training functions, f_1 and f_2, are shown in red and blue along with the training data points. The third function, f_3, is held out. *(Middle)* A single, over-parameterized function approximator is fit to the training data. *(Right)* A set of basis functions is fit to the training data. After calibrating, the basis functions may approximate all three functions accurately.

In Figure 7, we visualize how overfitting is especially problematic for over-parameterized approximators in a mixed-context (mixed-function) setting. Suppose we have a dataset consisting of two functions, f_1 and f_2, along with some data points from each. If we fit a single function approximator, e.g., a neural network, it may perfectly fit the training data as long as there is no input x common to both functions' datasets. However, this fit generalizes poorly and cannot be calibrated to fit f_3 without further training.

On the other hand, a function encoder approximation computes coefficients for any given function, and thus it effectively learns a space of functions rather than a single approximation. Assuming that the basis functions have been trained to span the training functions, each training function will be well-represented as a linear combination of the basis. After calibration, the function encoder even fits unseen functions as long as they are within the span of the basis. See Ingebrand et al. (2025) for more details on generalization to out-of-distribution functions. In summary, the function encoder avoids mixed-context overfitting by learning a separate function for each context.

B Algorithm Pseudocode

The algorithm is similar to the function encoder algorithm in Ingebrand et al. (2025), where the primary difference is the minimization problem solved to compute the coefficients. We additionally regularize the Gram matrix to be close to the identity, i.e., the basis functions are approximately orthonormal. For completeness, we include the pseudocode in Algorithm 1. Note that the inner product is defined as $\langle f, g \rangle = \int_{\mathcal{X}} f(x)^T g(x) dx$, which is approximated using the trajectory datasets.

C Function Encoder Action Head Architecture

The function encoder action head maps backbone features into k basis action predictions. It is implemented as a two-block MLPResNet, following the parallel decoding strategy from OpenVLA-OFT. The input is a feature vector of dimension 28672 (flattened from 4096×7). This vector is projected into a hidden dimension of 4096, processed through two residual MLP blocks, and then mapped to the output dimension $k \times d_a$, where the action dimension is $d_a = 7$. Thus, the output corresponds to k basis functions, each producing a 7-dimensional action vector. Formally, given input $h \in \mathbb{R}^{28672}$, the head produces $B(h) \in \mathbb{R}^{k \times 7}$, where each row $B_i(h)$ denotes the prediction of the i-th basis function.

Algorithm 1 Modified Function Encoder Algorithm (LAE)

given expert policy space Π_{exp}, learning rate α
Initialize basis $\{g_1, \ldots, g_k\}$ with parameters θ
while not converged **do**
  for all $\pi^c_{exp} \in \Pi_{exp}$ **do**
    $\alpha^c = \arg\min_{\alpha^c \in \mathbb{R}^k} ||\pi^c_{exp} - \sum_{i=1}^k \alpha^c_i g_i||_1$
    $\hat{\pi}^c_{exp} = \sum_{i=1}^k \alpha^c_i g_i$
  end for
  $L \leftarrow \sum_{\pi^c_{exp} \in \Pi_{exp}} \|\pi^c_{exp} - \hat{\pi}^c_{exp}\|_1$
  $L_{reg} \leftarrow \sum_{i=1}^k \sum_{j=1}^k (\langle g_i, g_j \rangle - \delta_{ij})^2$
  $\theta \leftarrow \theta - \alpha \nabla_\theta (L + L_{reg})$
end while
return $\{g_1, \ldots, g_k\}$

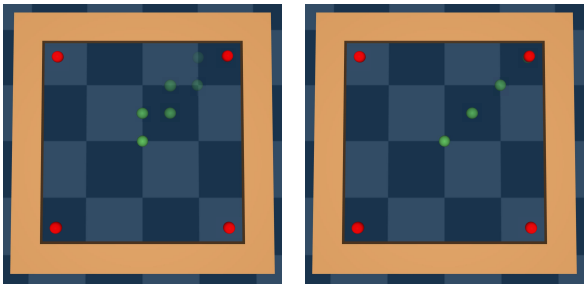

Figure 8: **Discrete vs. Continuous Skill Primitives.** *(Left)* A strictly hierarchical discrete policy restricted to "right" or "up" yields inefficient, Manhattan-style paths and cannot reach all goals. *(Right)* Continuous combinations of skills enable direct movement in any direction, allowing the robot to reach all goals.

D Discrete vs. Continuous Actions

OpenVLA learns a discrete distribution over a discretized version of the action space, and it generates the actions autoregressively Kim et al. (2024). This design decision was likely motivated by two factors. First, this allows OpenVLA to use the Llama 2 model without making any architectural changes. Second, it naturally allows for complex action distributions, such as stochastic bang-bang control. In contrast, a continuous action model is unlikely to learn bang-bang control and is more likely to learn the mean of this behavior. We choose to focus on continuous actions because this more naturally aligns with the function encoder algorithm. However, it is also possible to create a discrete distribution function encoder. See Ingebrand et al. (2025), Appendix C.

E Ablations

We perform ablations to evaluate how sensitive our model is to its hyper-parameters, namely the number of basis functions, the number of data points used to calibrate the model, and the distance function definition. Due to compute limitations, we only run one seed for each hyper parameter value. Unless otherwise specified, all ablations use 16 basis functions, 512 calibration data points, and L^1 distance.

Number of Basis Functions We vary the number of basis functions and retrain the model. We report the training results in Table 3. We observe that the algorithm is relatively insensitive to the number of basis functions beyond 8, suggesting that even 8 basis functions are sufficient to span the OXE dataset.

k	ManiSkill	CMU Exploration	KAIST Nonprehensile	Robocook	CMU Play	**Mean**
4	0.2846 ± 0.1135	0.3517 ± 0.1703	0.5785 ± 0.1210	0.2398 ± 0.1014	0.3297 ± 0.1009	0.3569 ± 0.1214
8	0.2535 ± 0.1275	0.2290 ± 0.2529	0.5798 ± 0.1275	0.1295 ± 0.1631	0.3021 ± 0.1831	0.2988 ± 0.1708
16	0.2570 ± 0.1383	0.2308 ± 0.2496	0.5844 ± 0.1316	0.1297 ± 0.1588	0.3057 ± 0.1782	0.3015 ± 0.1713
32	0.2646 ± 0.1402	0.2639 ± 0.2298	0.5901 ± 0.1345	0.1457 ± 0.1493	0.3135 ± 0.1539	0.3155 ± 0.1616

Table 3: Validation L^1 error for $k = 4, 8, 16, 32$ across OOD datasets (mean $\pm$ std).

Number of Calibration Data Points We vary the size of the calibration buffer and retrain the model. We plot the results in Figure 9. As expected, we find that a larger calibration buffer is more stable and converges faster, but it costs more compute.

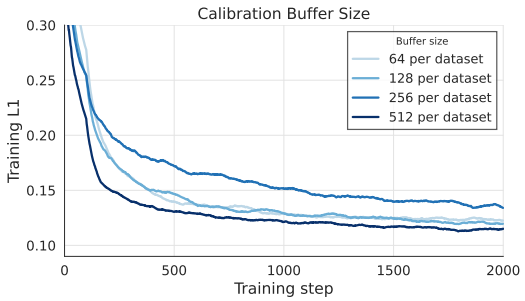

Figure 9: L^1 Loss during training for various calibration buffer sizes. The results indicate that a larger calibration buffer converges faster and more stably.

L^1 vs L^2 We compare L^1 distance definitions against L^2 distance definitions. L^1 requires solving a linear program, while L^2 has a closed form solution via least squares. We report the results in Table 4, which indicate that L^1 performs slightly better in this setting.

Method	ManiSkill	CMU Exploration	KAIST Nonprehensile	Robocook	CMU Play	**Mean**
L^1	0.257 ± 0.138	0.231 ± 0.250	0.584 ± 0.132	0.130 ± 0.159	0.306 ± 0.178	0.302 ± 0.171
L^2	0.265 ± 0.113	0.247 ± 0.205	0.580 ± 0.123	0.155 ± 0.134	0.332 ± 0.100	0.316 ± 0.135

Table 4: Validation L^1 error comparison between L^1 and L^2 models, reported as mean $\pm$ standard deviation for each dataset.

F Experiments Settings

The on-hardware and simulated experiments are as follows: (1) **Simulated Block Lifting:** A red block is placed on a table beneath the robot gripper, randomly to the left or right. The robot is allowed to move sideways or up and down to grasp and lift the block away from the table. (2) **Simulated Door Opening:** A door is placed to the right of the robot, approximately at the same height as its base. The robot gripper is allowed to move sideways and up and down to push and grab the handle. (3) **Goal Reaching:** A blue block is placed at a fixed location on the mat in front of the robot. The robot can move in x, y and z directions to reach and hover above the block. (4) **Block Lifting:** A larger yellow block is placed in front of the robot. The robot can move in x, y and z directions to grasp and lift the block. (5) **Pen Insertion:** The robot is initialized with a pen in its gripper. It is allowed to move sideways and up/down to insert the pen into the pen holder. Because the VLA models receive only a single front-facing RGB view and no proprioceptive state inputs, we simplify tasks 1,2, and 5 by disabling forward/backward motion to reduce the difficulty of depth estimation. See Figure 10.

Transfer to a Different Embodiment and Camera View. To further evaluate the transferability of MoS-VLA beyond the settings used in the main experiments, we consider a block-lifting task with a different robot embodiment and camera viewpoint. Specifically, we deploy MoS-VLA on a UR5E robot arm in Robosuite and place the camera at a 45° angle relative to the robot, rather than using the front-facing view adopted in the main experiments. We provide a single expert trajectory for calibration in this new setting, without any additional model training or gradient-based adaptation. MoS-VLA achieves an 85% success rate over 20 trials. These results suggest that the proposed adaptation mechanism can accommodate substantial changes in both embodiment and visual viewpoint from only a single demonstration.

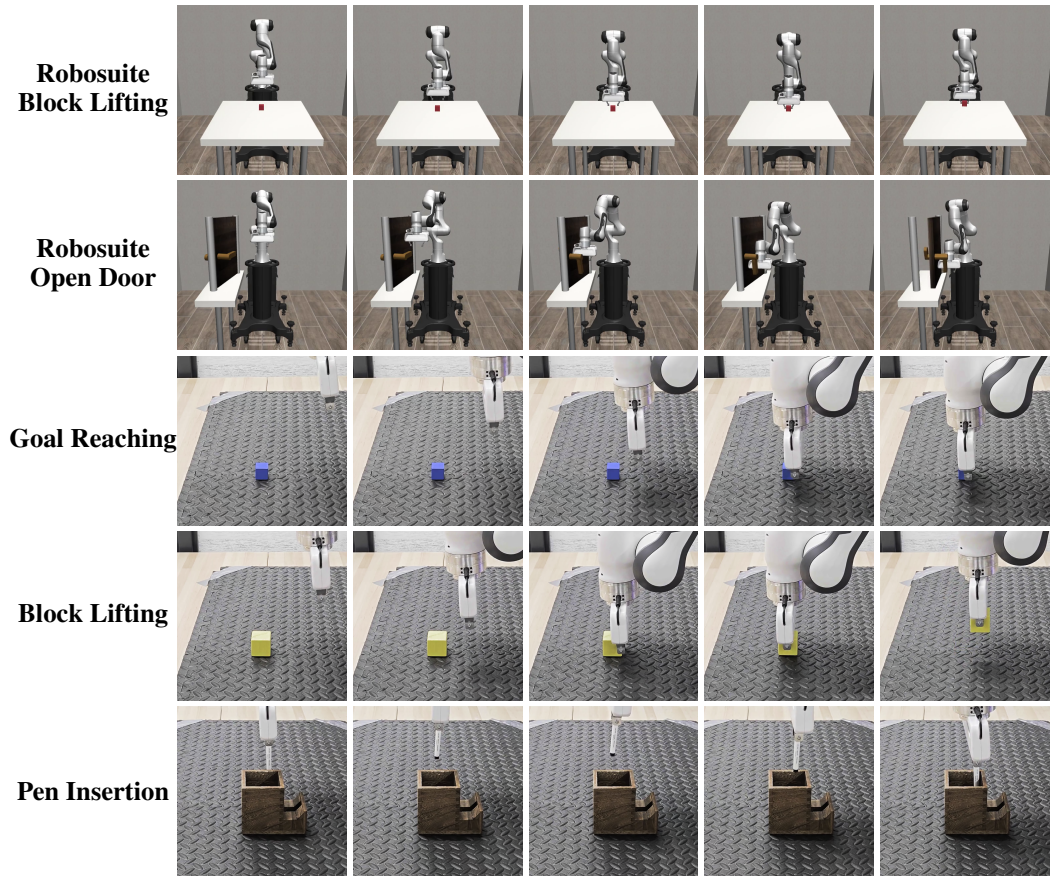

Figure 10: **Illustration of simulated and real robot tasks.** MoS-VLA rollouts across five tasks in both simulated and on-robot experiments. Having never seen these tasks during training, MoS-VLA needs only one demonstration for gradient-free adaptation.

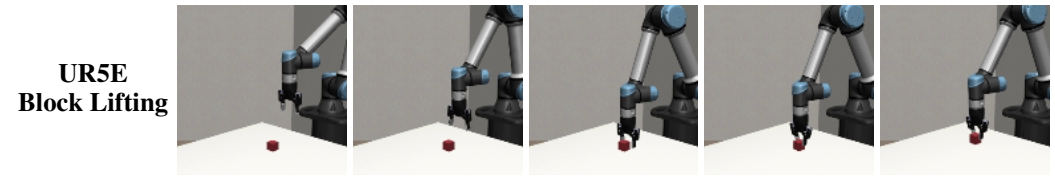

Figure 11: **Successful Transfer in a Different Embodiment and Camera Angle.** MoS-VLA rollouts on a block lifting task with the UR5E robot arm in simulation, with the camera placed at 45 degrees instead of in the front. MoS-VLA achieves an 85% success rate over 20 trials. The model is calibrated using one single expert trajectory in this new setting, but there is no extra training. This demonstrates the cross-embodiment and cross-camera-angle transferability of MoS-VLA.

G Extended Discussion and FAQ

Can the advantage of MoS-VLA be explained by differences in data or training duration? Since we train a new model bootstrapped from OpenVLA, one may wonder if this extra training explains the increase in performance. However, we carefully manage the training details to make sure this is not the case. 1) We use the exact same training datasets as OpenVLA, ensuring that there is no additional data to explain the performance. 2) We bootstrap from OpenVLA to minimize the compute required for the experiments. However, we are also changing the model significantly because we require it to output k different continuous actions rather than a distribution over discrete tokens. Naturally, changing the model in this way requires additional training. However, the total

number of training time we use for MoS-VLA is only about 6% of the training budget of OpenVLA, so this extra time is minimal. Since OpenVLA was trained till convergence (Kim et al., 2024), it's unlikely this extra 6% explains the change in performance.

Does MoS-VLA have additional parameters? Does this explain its performance? We use the single body, multiple head architecture (Ingebrand et al., 2025), meaning that almost the entire architecture for MoS-VLA is the same as the base model. The only difference is the addition of multiple output heads. Thus, for most architectures, there is a very small increase in parameters due to these extra heads, but they are negligible in size compared to the rest of the model. In the case of OpenVLA, MoS-VLA only requires 0.2% more parameters. So, this is unlikely to explain the model's performance.

H THE LATENT-SPACE FUNCTION-ENCODER EXTENSION

The action-space formulation of MoS-VLA represents a target policy as a linear combination of learned basis functions. Although each basis function itself could be highly nonlinear, their linear combination restricts the resulting family of policies to a linear function space. This may be restrictive when the variation across tasks is better described by a nonlinear manifold in action space.

We therefore consider a latent-space extension in which the basis functions predict elements of a learned latent space $Z \subset \mathbb{R}^{128}$ rather than actions directly. A nonlinear decoder then maps the resulting latent representation to the action space. In this formulation, policies remain linearly parameterized in latent space while inducing a nonlinear family of functions in action space.

To calibrate the basis coefficients from observation-action pairs, we additionally require a mapping from actions into this latent space. We therefore parameterize the latent representation using a VAE: an encoder maps actions to latent targets, while a decoder maps latent variables back to actions. The encoder, decoder, and latent basis are trained jointly so that the learned latent space both reconstructs actions accurately and admits a useful linear basis representation.

H.1 METHOD

We introduce a learned input token whose hidden representation after the VLA forward pass is $h(o) \in \mathbb{R}^{4096}$. A shared basis network maps $h(o)$ to $k = 16$ candidate latent vectors:

$$z_i(h(o)) \in \mathbb{R}^{128}, \qquad i \in \{1, \dots, k\}.$$

For each context c, a coefficient vector $\alpha^c \in \mathbb{R}^k$ combines these candidates:

$$z_{\text{mix}} = \sum_i \alpha_i^c z_i(h(o)).$$

A shared decoder $d(a \mid z)$ maps z_{mix} to the corresponding action or action chunk.

Calibrating α^c requires target latent representations for the observed actions. We obtain these targets with an encoder $e(z \mid a)$, which is trained jointly with the decoder and basis functions. Importantly, observations influence the predicted latent only through the input-conditioned basis $\{z_i(h(o))\}$ and the context-specific coefficients α^c. Thus, the basis is shared across contexts, while each context is represented by only k coefficients.

H.2 LEARNING A LINEAR LATENT FUNCTION SPACE

A standard VAE objective alone does not guarantee that the resulting latent function space is easier to represent linearly. Even if the decoder $d : Z \to A$ captures a nonlinear action manifold, the corresponding family of observation-to-latent mappings can itself remain nonlinear. We therefore train the latent representation together with the function-encoder basis, explicitly encouraging the target policies to be well approximated by a linear combination of latent basis functions.

The training objective combines four terms:

$$\mathcal{L} = \underbrace{\|d(z_{\text{mix}}) - a\|_1}_{\text{end-to-end prediction}} + w_l \underbrace{\|z_{\text{mix}} - z_{\text{post}}\|_1}_{\text{latent prediction}} + w_r \underbrace{\|d(z_{\text{post}}) - a\|_1}_{\text{reconstruction}} + w_k \underbrace{\text{KL}(q_\phi \| \mathcal{N}(0, I))}_{\text{prior}}, \tag{2}$$

where z_{post} denotes the encoded action latent. We use $w_r = 1$, $w_l = 1$, and $w_k = 10^{-5}$.

The end-to-end term trains the basis and decoder to predict actions directly, while the latent prediction term aligns the linear basis combination with the latent representation produced by the encoder. Together, these terms encourage the action-conditioned latent targets to lie near the span of the input-conditioned basis. The reconstruction and KL terms retain the usual VAE structure. Because the KL weight is small, the geometry of the learned latent space is driven primarily by reconstruction and prediction rather than by matching the Gaussian prior.

Calibration. Given a calibration set $\{(o_t, a_t)\}$, we first encode each action to obtain latent targets $e(a_t)$. We then solve for α^c by fitting the input-conditioned basis $\{z_i(o_t)\}$ to these targets. The resulting optimization is convex in α^c and requires no backpropagation through the VLA.

This construction separates shared and context-specific structure. The basis functions and VAE define a common latent representation, while a new context is specified by only $k = 16$ coefficients. When the target latent mapping is well represented by the learned span, adaptation therefore reduces to estimating a small number of coefficients from demonstrations. If the target falls substantially outside this span, the VAE encoder and decoder can instead be finetuned while keeping the much larger VLA backbone fixed.

H.3 Results

H.3.1 Out-of-Distribution Accuracy

Table 5 compares OOD action-prediction error. With 128 adaptation examples, both MoS-VLA variants substantially reduce prediction error relative to the zero-shot OpenVLA and $\pi_{0.5}$ models on the common seven-dataset evaluation. They also obtain lower error than X-VLA and ICRT, despite those methods using 512 adaptation examples.

Moving the function encoder into the learned latent space further improves performance: MoS-VLA (Latent) achieves a mean L^1 of 0.167 over the seven common datasets and 0.184 over all ten, compared with 0.197 and 0.225, respectively, for the action-space variant. In this experiment, the latent variant additionally finetunes the VAE encoder and decoder while leaving the VLA backbone fixed.

Method	**State-Action Pairs**	**Mean L^1 (7) ↓**	**Mean L^1 (10) ↓**
MoS-VLA (Latent)	128	**0.167**	**0.184**
MoS-VLA (Action)	128	0.197	0.225
OpenVLA	0	0.36	0.41
$\pi_{0.5}$	0	0.43	0.45
X-VLA	512	0.88	–
ICRT	512	0.74	–

Table 5: **OOD prediction accuracy.** The 7-column is the mean over the seven datasets supported by all baselines; X-VLA and ICRT do not support three of the datasets due to a absolute-action assumption, which cannot be accounted for in these datasets. The 10-column additionally includes evaluation on these three datasets.

H.3.2 Calibration Data Efficiency

We next isolate the gradient-free coefficient calibration step by freezing the basis functions, encoder, and decoder and varying the number of calibration examples. Table 6 shows that the resulting OOD error changes little beyond the smallest calibration set: mean L^1 decreases from 0.375 with $N = 16$ examples to 0.368 with $N = 512$.

Notably, this experiment measures the calibration-only regime and should not be compared directly with the 0.184 result above, which additionally finetunes the VAE encoder and decoder. Instead, the sweep shows that estimating the $k = 16$ context coefficients quickly reaches its performance plateau; additional examples provide little benefit once the low-dimensional coordinate vector is sufficiently constrained.

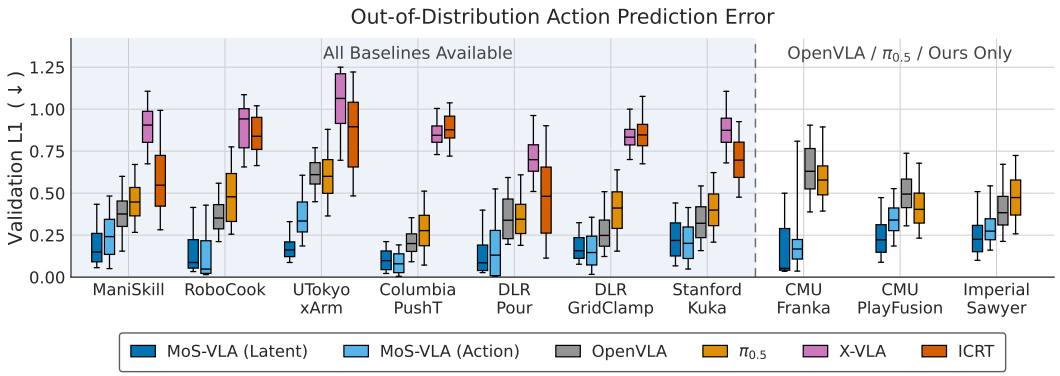

Figure 12: OOD action-prediction L^1 across the ten datasets.

N	16	32	64	128	256	512
mean OOD L^1	0.375	0.372	0.368	0.369	0.367	0.368

Table 6: Latent-variant calibration-only OOD L^1 vs. number of calibration examples. The loss barely decreases from N=16 to N=512.

H.3.3 Action Chunking Across Control Frequencies

The latent decoder predicts an entire action chunk from a single latent rather than generating future actions autoregressively. We therefore test whether prediction error increases for actions farther into the predicted chunk, particularly for datasets with low control frequencies where later chunk elements correspond to more distant future actions.

Across 38 datasets with verified control frequencies, we observe essentially no relationship between chunk position and prediction error. The mean per-chunk error slope is -0.0008, and this slope is uncorrelated with control frequency (Spearman $\rho = 0.004$, $p = 0.98$, $n = 38$). Thus, within the evaluated chunk horizon, later actions are not systematically harder to predict than earlier ones.

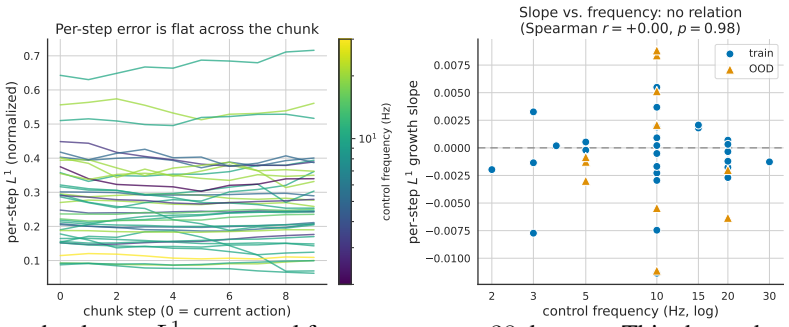

Figure 13: Per-chunk-step L^1 vs. control frequency across 38 datasets. This shows that the accuracy does not degrade for actions in the future.

H.4 What Does the Latent Basis Contribute?

The latent extension introduces two components at once: a learned VAE action representation and a function-encoder basis within that representation. To separate their effects, we compare MoS-VLA against a matched VAE model that removes the basis functions while keeping the same backbone, latent dimensionality, and VAE architecture.

Matched adaptation cost. For the primary comparison, both models finetune only the VAE encoder and decoder on the same 128 state–action pairs while keeping the VLA backbone frozen.

MoS-VLA additionally solves for the $k = 16$ coefficients α^c. Under this matched adaptation setting, MoS-VLA obtains a mean OOD L^1 of 0.184, compared with 0.189 for the VAE without basis functions (Table 7).

The difference is small on average but is larger on several of the more distinct held-out domains: 0.185 vs. 0.240 on CMU Franka, 0.179 vs. 0.200 on UTokyo xArm, and 0.135 vs. 0.185 on DLR Pour. A plain VAE reaches 0.176 only when it is additionally allowed to update the backbone through LoRA, increasing the adaptation cost.

These results indicate that imposing the shared latent basis does not trade away OOD accuracy under a matched adaptation budget. Instead, it provides a low-dimensional representation of context while retaining performance comparable to unconstrained VAE finetuning.

Method	**adaptation**	**mean OOD L^1 (10) $\downarrow$**
OpenVLA + VAE (no basis)	VAE finetune, frozen backbone	0.189
MoS-VLA (Latent, ours)	convex α + VAE finetune, frozen backbone	**0.184**
OpenVLA + VAE (no basis)	+ backbone LoRA finetune (more compute)	0.176
OpenVLA-OFT	full finetune	0.235

Table 7: **Matched-adaptation ablation.** Top block: at *identical* adaptation cost (VAE encoder / decoder finetune over a frozen backbone), the learned latent basis (ours) slightly outperforms a plain VAE (0.184 vs. 0.189). Bottom block (context): the plain VAE only surpasses ours (0.176) when additionally granted backbone gradient updates; a full OpenVLA-OFT finetune reaches 0.235.

Context representation. The distinction between the two models is most apparent in how they represent a new context. Without basis functions, adaptation is stored directly in the finetuned encoder and decoder weights. In latent MoS-VLA, the observation-dependent basis is shared across contexts, while context-specific variation is represented explicitly by the $k = 16$ coefficients α^c. A context therefore occupies a point in a common low-dimensional coordinate system rather than being represented solely through a separate set of finetuned weights. As with the action-space formulation studied in the main paper, this representation also makes the learned cross-context structure directly accessible through the coefficients.