
Task Reconstruction and Extrapolation for π_0 using Text Latent

Quanyi Li

Independent

quanyili0057@gmail.com

Abstract

Vision-language-action models (VLAs) often achieve high performance on demonstrated tasks but struggle significantly when required to extrapolate, combining skills learned from different tasks in novel ways. For instance, VLAs might successfully put the cream cheese in the bowl and put the bowl on top of the cabinet, yet still fail to put the cream cheese on top of the cabinet. In this work, we demonstrate that behaviors from distinct tasks can be effectively recombined by manipulating the VLA’s internal representations during inference. Concretely, we identify the *text latent* by averaging the text tokens’ hidden states across all demonstrated trajectories for a specific base task. For executing an extrapolated task, we can temporally interpolate the *text latent* of the two base tasks and add it back to the text hidden states, so sub-behaviors from the two tasks will be activated sequentially. We evaluate this approach using the newly created *libero-ood* benchmark, featuring 20 tasks extrapolated from the standard LIBERO suites. The results on *libero-ood* show that all SOTA VLAs achieve $< 15\%$ success rate, while π_0 with text latent interpolation reaches an 83% success rate. Further qualitative analysis reveals a tendency for VLAs to exhibit spatial overfitting, mapping object names to their locations in the demonstration rather than achieving genuine object and goal understanding. In addition, we find that decoding the *text latent* yields human-unreadable prompts that can nevertheless instruct the VLA to achieve a 70% success rate on the standard LIBERO suites, enabling private instruction and backdoor attacks. Code is available at: <https://github.com/QuanyiLi/pi0-text-latent>

1 Introduction

Building toward generalist capabilities, vision-language-action models (VLAs) trained with large-scale multi-modal datasets [26] have shown remarkable visual and language generalizability, leading to strong performance on diverse manipulation tasks [3, 26, 18, 14, 6, 11, 8, 39, 2, 32, 10, 28, 40]. Typically, for satisfactory deployment on new tasks, VLAs are fine-tuned using demonstrations of target tasks [13]. Though this paradigm ensures good in-distribution generalizability, we empirically find that VLAs struggle with out-of-distribution (OOD) tasks, particularly those extrapolated from demonstrated tasks they can perform well individually after fine-tuning. For instance, a VLA might successfully learn to "put the cream cheese in the bowl" and "put the bowl on top of the cabinet" from demonstrations, yet fail when asked to perform the extrapolated task of "put the cream cheese on top of the cabinet". As shown in figure 1, current state-of-the-art VLAs on the LIBERO [19] benchmark often fail to complete extrapolated tasks that require novel combinations of behaviors, even when the fundamental movements (like grasping a certain object and reaching a certain location) have been shown individually in demonstrations. This raises a critical question: Are VLAs merely overfitting to the specific demonstrated trajectories to move a specific object to a fixed location, or are they learning composable behaviors that could support broader generalization? In this work, we demonstrate that

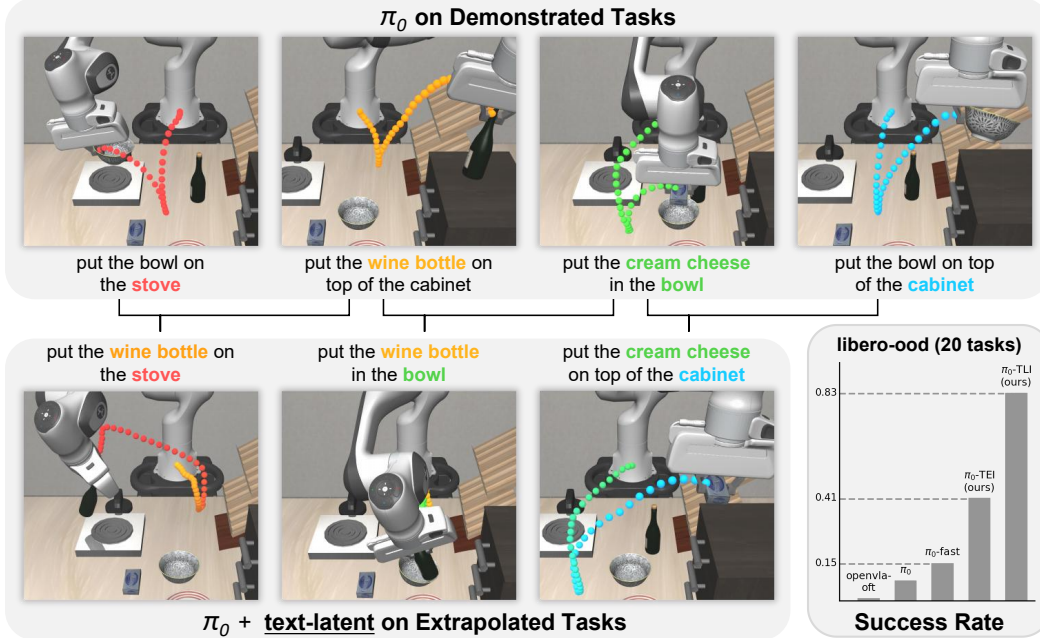


Figure 1: After fine-tuning on the LIBERO [19] demonstrations, π_0 , π_0 -fast, and openvla-oft can complete the *libero-goal* tasks shown in the first row with more than 95% success rate. However, when executing on the proposed *libero-ood* benchmark containing tasks extrapolated from standard LIBERO task suites, they achieve <15% success rate. By applying *Text Embedding Interpolation* (TEI) and *Text Latent Interpolation* (TLI), we can improve the performance of π_0 up to 42% and 85%, respectively. Behaviors of π_0 -TLI on three exemplary extrapolated tasks are shown in the second row.

behaviors from distinct base tasks can be effectively recombined to complete extrapolated tasks by directly manipulating the VLA’s internal representations during the inference time. We focus on the SOTA VLA, π_0 , and introduce a method to identify *text latent* that carries task-specific semantics and use it to control the VLA’s behaviors for task reconstruction and extrapolation.

To be specific, we identify the *text latent* for a given task by recording the hidden states of the text tokens for each transformer layer and averaging layer-wise features across all demonstrated transitions for the task of interest. The resulting *text latent* thus captures the core instruction needed to execute the task. If we want to activate a specific task behavior, we can add the associated *text latent* to text tokens’ residual streams [7] to remind it of the target behavior. This allows us to reconstruct the task trajectory without providing the prompt, but by intervening in the model’s internal states. We also find that unembedding early layers of *text latent* can produce alternative task prompts which are hard to understand even for those who are familiar with the original prompts. However, feeding the VLA with these alternative prompts can still reconstruct tasks from the standard LIBERO benchmark with about 70% success rate, enabling private instruction and backdoor attack. To execute an extrapolated task, which requires transitioning from one base behavior (Task 1) to another (Task 2), we propose *Text Latent Interpolation* (TLI). This technique involves applying a temporally interpolated *text latent* to the model’s hidden states during each inference step. At the start of the episode, the intervention guides the model towards Task 1’s behavior, and as the episode progresses, the influence smoothly shifts towards Task 2’s behavior by adjusting the mixing ratio of the two *text latent*. This manipulation effectively stitches together sub-trajectories learned from different base tasks. We also explore a simpler approach, *Text Embedding Interpolation* (TEI), which linearly interpolates the text embeddings of the two base task prompts over time and shows a 41% success rate. As shown in figure 1, π_0 with *text latent* can finish multiple extrapolated behaviors, even if the newly composed trajectories are not shown in the training data.

To evaluate our approach, we introduce the *libero-ood* benchmark, comprising 20 challenging extrapolated tasks derived from standard LIBERO task suites (*libero-goal*, *libero-spatial*, and *libero-*

object). These tasks are designed such that the required grasping and placement locations have been seen individually in the training demonstrations, but the specific combination required for the extrapolated task has not. Our experimental results on *libero-ood* show that baseline state-of-the-art VLAs achieve less than a 15% success rate, highlighting their limitations in extrapolation. In contrast, π_0 augmented with our *Text Latent Interpolation* (TLI) method achieves a remarkable 83% success rate, demonstrating the effectiveness of manipulating internal representations for task behavior recombination. Further qualitative analysis reveals that VLAs often exhibit spatial overfitting, associating or mapping object names with the locations where they appear in the training data rather than learning true object or goal understanding.

2 Related Work

2.1 Vision-Language-Action Models

VLAs are usually initialized from vision-language models (VLMs) pretrained with large-scale cross-modality data. To adapt it for decision-making, we first pre-train it with large-scale cross-embodiment dataset like Open X-Embodiment [26], following a fine-tuning on specific tasks with fewer demonstrations [3, 26, 18, 14, 6, 11, 8, 39, 2, 32, 10, 28, 35, 23, 31, 27, 40]. Though VLAs are all built upon the transformer-based pre-trained VLMs [34], it remains unclear what the best way to decode actions is. OpenVLA [14], RT-2 [3], and π_0 -fast [27] follow the next-token prediction manner used by VLMs and LLMs and decode discrete action tokens, which will be converted to continuous actions according to different tokenization schemes. Another widely adopted way to predict action is to use a regression head or diffusion model with extra parameters [23, 2, 14, 37, 15, 5, 13, 16]. Despite the various designs of action decoders, they both use transformer-based VLMs as encoders. Thus, we choose to mine the meaningful internal representation of the VLM backbone of π_0 , which adopts an additional action expert to generate continuous actions with flow matching.

2.2 Mechanistic Interpretability (MI)

This is a sub-field of interpretability, where researchers reverse engineer the model’s internal computations to understand how it works [30, 7, 34] and reconstruct or activate certain behaviors. Some early works analyze image classification models [24, 1, 41] and find that neurons play the role of feature detector for patterns from simple curves to complex objects like cats [4]. Recent studies of MI on LLM and VLM identifies important circuits that can perform specific tasks like induction head [25] to output repeating words, function vectors [33, 21] to produce antonyms, attention head to detect number or shape [9], neurons contributing to recognize specific objects [9], and internal features making VLMs hallucinate [12]. However, there are limited works that study the neural policies or VLAs from the MI perspective. The most relevant works study adversarial attack with feature attribution [36], symbolic representation uncovering [20], and motion-relevant neuron identifying and characterization [17]. Our work instead finds causal effects between the internal representations and VLAs’ behaviors. As a result, we can produce obscure prompts for task reconstruction with *logit lens* [22], enabling private instruction or adversarial backdoor attack. In addition, the identified functional component enables the π_0 for extrapolated tasks, which all SOTA VLAs struggle with.

3 Method

We start by formulating how VLAs work generally. Then, we illustrate how to identify *text latent* and how to use them to change the model’s internal representation, thus steering its behavior.

3.1 Preliminary

Existing VLAs adopt VLMs as encoders to fuse both vision and language information. Concretely, a pre-trained vision encoder, e.g., CLIP [29] and SigLIP [38], is used to generate d -dimensional image embeddings from image patches, followed by d -dimensional text embeddings that are tokenized and projected from the task description. For some VLAs [14], the task description is encapsulated with certain context, like "what actions the robot should take to *{task description}*", which introduces extra tokens. In addition, the proprioceptive state will be projected [14] or tokenized [27] into the d -dimensional shared space to work with image-text embeddings. After tokenization, we assume

there are totally m d -dimensional embeddings obtained from image, text, and robot proprioception, $e = \{e^i : e^i \in \mathbb{R}^d, i = 1 \dots m\}$, which will go through L transformer layers. Except for the last layer, each layer l outputs hidden states $h_l = \{h_l^i : h_l^i \in \mathbb{R}^d, i = 1 \dots m\}$. The action is then generated by $a = f(e, h)$, where $h = \{h_l : h_l \in \mathbb{R}^{m \times d}, l = 1 \dots L - 1\}$ is hidden states of all tokens across $L - 1$ layers. More precisely, the action generation uses the KV cache derived from e and h , where the image-text embeddings e are passed through the first transformer layer (layer 0) to compute key and value projections, and the hidden states h_l are passed through their subsequent layer $l + 1$ to compute the corresponding projections. To simplify the notation, we condition the action generation on e and h . This formulation applies to most existing VLAs¹ as long as the action generation module reuses the VLM backbone to do decoding in an autoregressive way [14, 28, 3, 27] or the action prediction module is also transformer-based [23, 2], which thus takes the KV-cache to do casual self-attention.

3.2 Text Latent

As VLAs' behavior depends on the task description, we hypothesize that behavior-primitives are managed by the internal representations of text (task description) tokens. To study this mechanism, we denote the text embeddings as $e^T = \{e^i : e^i \in \mathbb{R}^d, i \in T\}$ and their hidden state at each layer as $h_l^T = \{h_l^i : h_l^i \in \mathbb{R}^d, i \in T\}$, where T is the set of text tokens indices. Therefore, the hidden states of all text tokens across all $L - 1$ layers can be represented by a tensor as $h^T \in \mathbb{R}^{L-1 \times |T| \times d}$ after preserving orders and stacking them. By denoting the rest of the embeddings and their hidden states as $e^- = e \setminus e^T$ and $h^- = h \setminus h^T$, we can update the action generation function at timestep i as

$$a = f(e^T, h^T(i), e^-(i), h^-(i)) \quad (1)$$

This is because the text (task description) embeddings are fixed throughout the whole episode, while the image and proprioceptive tokens are changed at different timesteps, and thus all hidden states are also distinct after doing attention across tokens. Our goal is to manipulate the e^T or $h^T(i)$ at each decision-making step to activate behaviors with specific semantics.

Text latent is the ingredient for doing this. We identify it by averaging a set of text hidden states $\{h^T(i) : h^T(i) \in \mathbb{R}^{L-1 \times n \times d}, i \in B\}$, where B is all timestep indices of a demonstrated episode for the target task, and $h^T(i)$ is thus obtained by forwarding the model with the observation at timestep i . As multiple demonstrated episodes exist for a single task in the training sets, the average is thus taken over K demonstrations. Therefore, the *text latent* can be calculated by element-wise average:

$$\mathcal{T} = \frac{1}{\sum_{k=1}^{k=K} |B_k|} \sum_{k=1}^{k=K} \sum_{i \in B_k} h^T(i) \quad (2)$$

In our reconstruction experiment, we demonstrate that \mathcal{T} captures the most essential knowledge for finishing the corresponding task. Furthermore, we can combine two *text latent* to recombine skills from the two associated tasks for completing extrapolated tasks.

3.3 Text Latent Interpolation

An extrapolated task constructed from two base tasks can be interpreted as starting with Task 1 and gradually transforming into Task 2. Though text token ids are discrete and therefore cannot be changed smoothly throughout the episode, we can approximate a continuous transition by linearly interpolating the text embeddings of the two base task prompts, e_1^T and e_2^T , given time step index i . This technique—called *Text Embedding Interpolation* (TEI)—is defined as

$$e^T = e^T(i) = \left(1 - \frac{i}{\lambda}\right)e_1^T + \frac{i}{\lambda}e_2^T, \quad 0 \leq i \leq \lambda, \quad (3)$$

where the hyperparameter λ controls the transition speed or mixing ratio at different timesteps. Unless otherwise noted, we set λ to the average number of policy-execution steps required to finish a task in LIBERO (24 steps). For tasks where the object is already near its destination—e.g., *put the wine bottle in the bowl*—we shorten λ to 14 so the second task behavior is activated sooner. Intuitively, TEI rewrites the task prompt at every step with a weighted blend of the two source instructions, achieving a smooth instruction switching. On the other hand, we can leave the target task prompt

¹Except for models using bidirectional attention where text and image token attend to the action tokens [13].

(and thus its embedding) unchanged and operate directly on the model’s hidden states with the *text latent* of the two base tasks ($\mathcal{T}^1, \mathcal{T}^2$). The core idea is to suppress all information about the second task at the beginning of the episode and all information about the first task at the end. As the episode progresses, Task 2’s context is gradually injected into the residual stream while Task 1’s context fades out. We term this approach *Text Latent Interpolation* (TLI), which modifies the text hidden states $h^T(i)$ as follows:

$$h^T(i) = h^T(i) + \left(1 - \frac{i}{\lambda}\right)(\mathcal{T}^1 - \mathcal{T}^2) + \frac{i}{\lambda}(\mathcal{T}^2 - \mathcal{T}^1), \quad 0 \leq i \leq \lambda. \quad (4)$$

TLI can also be applied to a specific layer l by replacing $h_l^T(i)$ with the interpolated pair $(\mathcal{T}_l^1, \mathcal{T}_l^2)$ in the same fashion. The full interpolation procedure is listed in Algorithm 1. Both TEI and TLI can be deployed independently or jointly. In either case, the ratio i/λ is clipped between 0 and 1. If the length of the text dimension of \mathcal{T}^1 or \mathcal{T}^2 differs from the number of tokens of the target prompt, we truncate it or pad it with zeros to match the length of the target prompt.

Algorithm 1 TEI and TLI during Task Execution

Require: *text latent* $\mathcal{T}^1, \mathcal{T}^2$, Interpolation steps λ , Initial observations o , Max timestep J

- 1: **for** $i = 1$ to J **do**
- 2: $e^-, h^-, e^T, h^T \leftarrow \text{encode}(o)$ ▷ Get internal representation
- 3: **if** Text Embedding Interpolation (TEI)
- 4: $e^T = (1 - i/\lambda)e_1^T + (i/\lambda)e_2^T$ ▷ Overwrite text embedding
- 5: **if** *text latent* Interpolation (TLI)
- 6: $h^T(i) = h^T(i) + (1 - i/\lambda)(\mathcal{T}^1 - \mathcal{T}^2) + i/\lambda(\mathcal{T}^2 - \mathcal{T}^1)$ ▷ Add back to hidden states
- 7: $a = f(e^T, h^T(i), e^-(i), h^-(i))$ ▷ Decode action
- 8: $o \leftarrow \text{simulation}(a)$ ▷ Forward simulation
- 9: **end for**

4 Experiments

Benchmark. We conducted experiments using the LIBERO benchmark [19], a simulation environment widely employed for evaluating Vision-Language-Action (VLA) models [2, 27, 13, 28, 40]. LIBERO features four distinct task suites designed to assess various abilities of manipulation policies. Our experiments use three standard suites from LIBERO: *libero-goal*, *libero-object*, and *libero-spatial*. Each suite contains 10 tasks, and most of them require pick-and-place. The details of these standard suites can be found in the Appendix A. Additionally, we introduce a novel task suite, *libero-ood*, comprising two sub-suites *libero-goal-ood*, and *libero-spatial-ood*. Each contains 10 extrapolated tasks. The key idea behind these 20 tasks is to ensure that both the grasping and placement locations have individually appeared in the training data, so the policy has already learned how to reach these locations. However, the specific trajectory connecting these two locations has not been presented in the demonstration. As a result, solving these tasks requires the policy to stitch together sub-trajectories it has learned from tasks in *libero-goal* and *libero-spatial*. Details can be found in figure 2. This new suite allows for further evaluation of VLA generalizability, especially extrapolation ability.

Experiments. Our method is verified with the state-of-the-art VLA π_0 , which features 18 transformer layers, resulting in 17 intermediate hidden states. For each task in the three standard suites, we identify the corresponding *text latent* using 20 demonstrations, calculated according to equation 2. Although more than 40 demonstrations are available per task, we empirically find that using 20 episodes is sufficient for robust identification of the *text latent*. We first demonstrate that *text latent* encapsulates the essential knowledge required to complete tasks in the three standard task suites. This allows for the reconstruction of necessary task behaviors even when provided without prompts or with ambiguous or even human-unreadable prompts. Second, we show that by interpolating between *text latent* or *text embeddings*, π_0 can successfully complete challenging extrapolated tasks from our proposed *libero-ood* suite, where tasks pose difficulties for current state-of-the-art VLAs. Finally, our analysis across specific tasks reveals that the VLA’s behavior relies on the mechanistic memorization of demonstrated trajectories, rather than a genuine conceptual understanding of objects or goals.

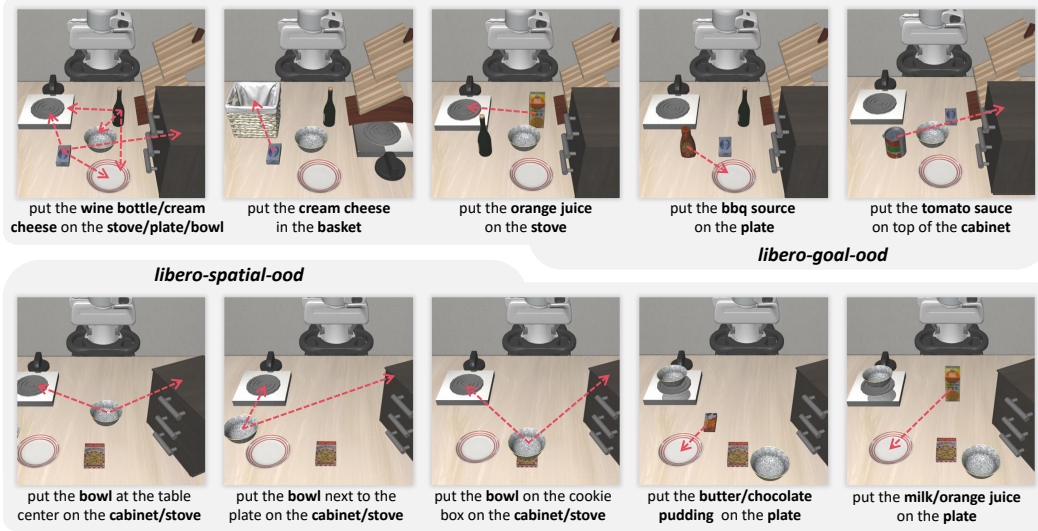


Figure 2: Visualization of scene layouts, task goals (denoted by the red arrows), and prompts for tasks in *libero-ood*, which can be further split into *libero-goal-ood* and *libero-spatial-ood*. *libero-goal-ood* includes six extrapolated tasks that require combining sub-skills learned from *libero-goal* and operating in the same scene layout as *libero-goal* tasks (the top-left figure), while the remaining four tasks slightly change the layout and additionally demand transferring knowledge about objects. On the other hand, *libero-spatial-ood* provides six tasks that require transferring the skills learned in *libero-goal* and *libero-spatial* to place the object on the cabinet or stove. The remaining four tasks require placing unseen objects from *libero-object* at new locations on the plate.

Evaluation. For each task, we execute 10 independent runs using different random seeds. Therefore, the final success rate for each task suite is calculated as the proportion of successful episodes across the total 100 runs (10 tasks \times 10 runs/task). All experiments are conducted with Nvidia RTX 4090.

4.1 Task Reconstruction

In this experiment, we try to reconstruct the behavior or trajectory that is required to complete the tasks from *libero-goal*, *libero-object*, and *libero-spatial* with *text latent*. We use two ways to examine the effectiveness of *text latent*. The first way is to mask all the text tokens or set each text token to the space character (" ") so the text prompt doesn't provide any task information. We can then add back the *text latent* to each hidden layer representation when executing the policy. The second way is to unembed the early layer vectors of *text latent* into a set of token ids [22] different from the original prompt. As a result, we can feed the alternative prompt to the model directly.

	libero-object	libero-goal	libero-spatial
<i>Original</i>	0.98	0.95	0.97
Mask Prompt	0.11	0.14	0.24
Blank Prompt	0.28	0.16	0.23
Blank Prompt+ \mathcal{T}	0.94	0.82	0.81
\mathcal{T}_1 -Prompt	<u>0.92</u>	<u>0.66</u>	0.98
\mathcal{T}_2 -Prompt	0.73	0.19	<u>0.85</u>
\mathcal{T}_3 -Prompt	0.56	-	0.68

Table 1: The success rate of different ways to reconstruct tasks.

lines show the performance of different ways to reconstruct the task. For each task suite, the settings yielding the best performance are bolded, and the second-best performance is underlined. The result shows that by adding back the *text latent* to the model's hidden states by $h^T(i)+ = \mathcal{T}$ at each policy

The results are shown in table 1. The first line is the official performance of π_0 with the original task prompt and serves as the upper bound of performance. The second line uses only image input for all task execution with all text tokens masked, while the third line adds back prompts but fills with space characters (" "). In both settings, the policy has no task instructions, suggesting the lower bound of performance. The rest of the

Original Prompt	Prompt unembedded from <i>text-latent</i> (tokens)
pick up the black bowl next to the plate and place it on the plate	'aŕŕ' 'QSize', 'U000e0062', 'black', 'aŕŕ' 'aŕŕ' 'QSize', 'U000e0062', 'plate', 'QSize', 'QSize', 'QSize', 'U000e0062', 'QSize'
pick up the black bowl on the cookie box and place it on the plate	'aŕŕ' 'aŕŕ' 'U000e0062', 'black', 'aŕŕ' 'U000e0062', 'U000e0062', 'cookie', 'QSize', 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl on the wooden cabinet and place it on the plate	'aŕŕ' 'aŕŕ' 'U000e0062', 'black', 'aŕŕ' 'U000e0062', 'U000e0062', 'wooden', 'cabina', 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl on the ramekin and place it on the plate	'aŕŕ' 'aŕŕ' 'U000e0062', 'black', 'aŕŕ' 'U000e0062', 'U000e0062', 'aŕŕ' 'kin', 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl on the stove and place it on the plate	'aŕŕ' 'aŕŕ' 'U000e0062', 'black', 'aŕŕ' 'U000e0062', 'U000e0062', 'stove', 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl next to the ramekin and place it on the plate	'aŕŕ' 'QSize', 'U000e0062', 'black', 'aŕŕ' 'aŕŕ' 'QSize', 'U000e0062', 'aŕŕ' 'kin', 'QSize', 'QSize', 'QSize', 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl from table center and place it on the plate	'aŕŕ' 'QSize', 'U000e0062', 'black', 'QSize', 'FROM', 'table', 'aŕŕ' 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'aŕŕ'
pick up the black bowl between the plate and the ramekin and place it on the plate	'aŕŕ' 'aŕŕ' 'U000e0062', 'black', 'aŕŕ' 'between', 'U000e0062', 'aŕŕ' 'QSize', 'U000e0062', 'aŕŕ' 'kin', 'QSize', 'QSize', 'aŕŕ' 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl in the top drawer of the wooden cabinet and place it on the plate	'QSize', 'QSize', 'U000e0062', 'black', 'QSize', 'U000e0062', 'U000e0062', 'Top', 'aŕŕ' 'QSize', 'U000e0062', 'wooden', 'cabina', 'QSize', 'QSize', 'QSize', 'U000e0062', 'U000e0062', 'QSize'
pick up the black bowl next to the cookie box and place it on the plate	'aŕŕ' 'QSize', 'U000e0062', 'black', 'QSize', 'QSize', 'QSize', 'U000e0062', 'cookie', 'QSize', 'QSize', 'QSize', 'QSize', 'QSize', 'U000e0062', 'QSize'

Table 2: Prompts decoded from \mathcal{T}_3 for *libero-spatial* tasks and can instruct π_0 to finish tasks with around 70% success rate. This enables adversarial attacks and private instructions.

execution step, the task can be finished even if the prompt is masked or provides none of the task information. *Text latent* improves the performance to $> 80\%$ for the three task suites. Therefore, it confirms that *text latent* capture the essential task instruction for action generation, and it is feasible to remind π_0 of a task by adding its corresponding *text latent* to each layer’s residual stream.

As the $\mathcal{T} \in \mathbb{R}^{L-1 \times n \times d}$, we can thus applying the language embedding matrix E to unembed \mathcal{T}_l into tokens by calculating the cosine similarity between each column of E and \mathcal{T}_l , then selecting the indices with maximum similarity as alternative token. In our experiments, we choose \mathcal{T}_1 , \mathcal{T}_2 , and \mathcal{T}_3 , which are the averaged hidden states of the first three layers output respectively. As shown in table 1, we found that for *libero-goal*, the prompt produced by unembedding \mathcal{T}_1 has a 66% success rate, while the next layer’s vector can not reconstruct the task well. However, for *libero-object* and *libero-spatial*, even the prompt unembedded by \mathcal{T}_2 and \mathcal{T}_3 can reconstruct most of the tasks in this suite. In table 2, we show the prompt unembedded by \mathcal{T}_3 for *libero-spatial*. We also show embedded prompts for the other two tasks in Appendix B. Though most of the unembedded prompts are human-unreadable, they can still instruct the VLA to finish a specific task with about 70% success rate for the three standard task suites. These prompts are hard to understand, even for people who are familiar with the original prompts and the tasks that the VLA is fine-tuned with. This enables an application of the *text latent*, obscure prompting, which can be used for private instruction or backdoor attack.

4.2 Task Extrapolation

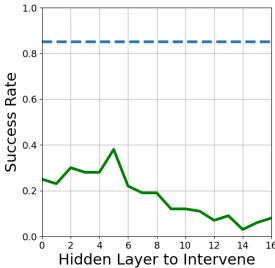
The previous experiment demonstrates that *text latent* encodes how to complete individual tasks and thus can trigger associated behaviors when added to the residual streams. We then ask whether the behaviors learned from separate tasks can be recombined with the relevant *text latent*. We refer to any task that demands such a behavior recombination as an *extrapolated task*. For example, the VLA learns to pick up cream cheese in the task *put the cream cheese in the bowl*, and learns to place an object on top of the cabinet in the task *put the wine bottle on top of the cabinet*. Consequently, it is supposed to be able to carry out *put the cream cheese on top of the cabinet* and *put the wine bottle in the bowl*. These new tasks don’t change objects’ locations or scene layouts but only need to recombine learned skills. In this paper, we focus on the extrapolated pick-and-place tasks and propose the *libero-ood* task suite for evaluation of such an ability, as shown in figure 2.

The benchmark results on the two sub-suites of *libero-ood* are shown in table 4.2 with the overall performance. It shows that all SOTA policies on LIBERO [19] benchmark failed to complete the extrapolated tasks, even if only a simple recombining of learned skills is needed, and objects are placed at locations that the VLA is supposed to be able to reach after fine-tuning with demonstration. Among all baselines, the best one is π_0 -fast. It generalized well to two tasks, *put the cream cheese on the plate* and *put the chocolate pudding on the plate*, with a 100% success rate. But it also struggles with all other extrapolated tasks. These baselines usually fail to pick the correct object or fail to put it in the correct place. After applying the proposed techniques, TEI and TLI, for π_0 , we can largely improve its success rate to 41% and 83%, respectively, on 20 extrapolated tasks. This suggests our method allows rearranging different skills the VLA has learned for extrapolation tasks. We also applied both TEI and TLI to the model inference together. The results of π_0 -TEI-TLI show that

	openvla-oft	π_0 -fast	π_0	π_0 -TEI	π_0 -TLI	π_0 -TEI-TLI	π_0 -TLI*
libero-goal-ood	0.01	0.13	0.02	0.42	0.85	0.85	0.33
libero-spatial-ood	0	0.17	0.16	0.41	0.81	0.69	0.18
All	0.01	0.15	0.09	0.41	0.83	0.77	0.25

Table 3: The success rate of SOTA VLAs and our methods on the *libero-goal-ood* task suite. Our methods outperform all baselines by simply interpolating the model’s latent representation. Detailed performance of each task and the behavior visualization of π_0 -TLI is available in Appendix D and C.

TEI doesn’t further improve the performance of π_0 -TLI for *libero-goal-ood*, while it even harms the performance of *libero-spatial-ood*. In addition, we run another experiment (π_0 -TLI*) using blank prompts similar to the previous experiment with TLI. Its performance drastically drops to 33% and 18%, compared to π_0 -TLI, indicating the importance of the prompt for extrapolated tasks. As the prompt of extrapolated tasks is mixed from two base tasks by stitching, the text embedding of the extrapolated task can be viewed as $e^T(i) = \text{concat}(e_1^T[:a], e_2^T[b:])$, where a and b indicates the prompt truncation position. It is another way for interpolating text embedding and benefits π_0 -TLI, while using this stitching prompt solely produces a 9% success rate (naive π_0), worse than TEI.



In addition, we try to find a more compact structure for *text latent*. Specifically, we intervene in a specific layer l of the model with \mathcal{T}_l to find whether it contributes significantly to the extrapolated tasks. As shown in the left figure 4.2, the early several layers can work alone to finish tasks in *libero-ood* with more than 20% success rate. After layer 6, the success rate begins to drop, while at layer 16 the success rate recovers to 10%. As each layer of \mathcal{T} can work independently and contributes more or less to the success, we decide to keep all of them. As a result, the π_0 with all hidden layer interventions can achieve an 83% success rate indicated by the blue dashed line.

4.3 Qualitative Results

As shown in Fig. 2, the last 4 tasks in *libero-goal-ood* are special, as they require the model to pick up objects appearing in the scenes of *libero-object*. Although π_0 was also fine-tuned with *libero-object* demonstrations, those objects had never appeared in any *libero-goal* scene before. To test generalization, we swap the original objects from *libero-goal* with these new ones while keeping the displaced objects in the scene as well. Consider the task *put the orange juice on the stove*. The orange juice is placed at the original position of the wine bottle, and the wine bottle is moved to the former position of the cream cheese. Consequently, the *text latent* used for *put the orange juice on the stove* can be borrowed from two base tasks: *put the wine bottle on top of the cabinet* and *put the bowl on the stove*. The trajectories produced by π_0 -TEI-TLI for all four tasks are shown in figure 3.

Using both TEI and TLI, π_0 achieves an average success rate of 85% on these four tasks. In this setting, the text embeddings and hidden states both convey the instruction to pick the cream cheese or wine bottle and place it on the plate or stove. However, π_0 ignores the current positions of the cream cheese, wine bottle, and stove; instead, it grasps whichever object occupies the positions where those items are located in the training data. In the rightmost visualization, it likewise disregards the stove’s true location and places the object in the basket, because the basket occupies the stove’s original position in the demonstration. This behavior indicates that the VLA does not understand object identity but instead maps object names to fixed locations. Thus, “cream cheese” actually means “the object at the location where the cream cheese appeared during training.” Thus, the proposed *text latent* interpolation is a way to take advantage of this feature. On the other hand, the performance on these four tasks reveals that the *text latent* found for π_0 is robust to the types of objects to grasp and the destination to drop them. This robustness is also reflected by the performance on another four tasks from *libero-spatial-ood* that also require picking unseen objects like milk, butter, and so on. Furthermore, the *libero-spatial-ood* requires the policy to reuse the skills learned from *libero-goal-ood*. Therefore, it proves that the *text latent* is general and robust to the scene layouts and thus the change of visual observations.

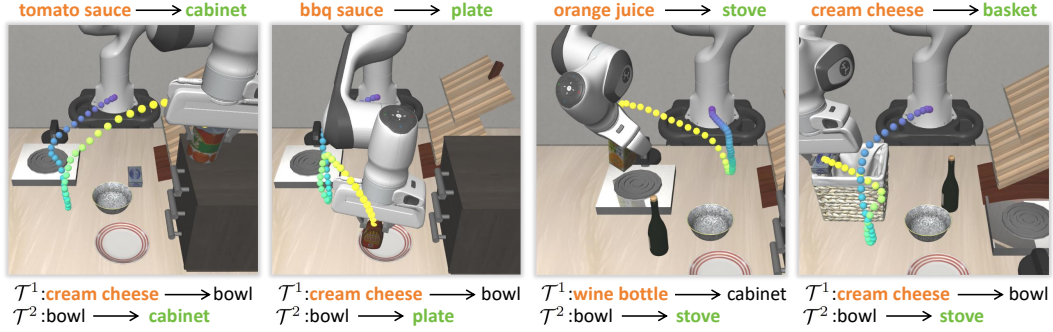


Figure 3: Trajectories of π_0 -TEI-TLI for the four tasks that require object recognition. Object to grasp and place to drop are shown on top of the figure; The two *text latent* used to complete the task are listed underneath. These behaviors suggest spatial overfitting. For example, we can use the *text latent* and text embedding of "grasping the cream cheese" to pick the tomato sauce that occupied the original location of the cream cheese in the demonstration, ignoring the cream cheese in the scene.

4.4 VLAs Overfit to Locations

The spatial overfitting pattern also emerges in the standard *libero-object* suite, where five tasks place the target object at the center of the scene, and five place it in the top-right corner. After fine-tuning with demonstrations, the policy can always pick the central object with any of the five "center" prompts and the top-right object with any of the five "top-right" prompts. In the two-prompt experiments shown in table 4, we always use the prompt, *pick up the cream cheese and place it in the basket* to pick the object at the scene center, and the prompt, *pick up the alphabet soup and place it in the basket* for the objects at the top-right corner. We then run the task suite with the two prompts. All SOTA VLAs can maintain the performance on the *libero-object* with incorrect prompts.

	π_0	π_0 -fast	openvla-oft
Two-prompt	0.94	0.86	0.99
OOD-position	0	0	0

Table 4: The success rate on modified *libero-object* suite

elled to the original location of the target object, picked up whatever object happened to occupy that spot, and dropped it in the basket. This consistent failure mode reinforces that current VLAs overfit to the trajectories they observed during training. However, if these objects are placed at locations that the VLA can reach, we can still instruct it to put the objects into the basket with the proposed TLI and TEI, which is actually a way to exploit spatial overfitting.

In addition, when the target object is located anywhere other than the center or top-right, the policy fails. In the OOD-position experiment 4, we relocated the target objects to a place other than the centre or the top-right corner. As expected, all VLAs failed. The end effector still trav-

5 Conclusion

In this work, we identify the functional component, *text latent*, for triggering specific task behaviors. We thus use it to reconstruct tasks that the VLA is originally capable of by intervening in the hidden states of text tokens or using the surrogate prompt unembedded from the *text latent*. We also propose two methods, *Text Latent Interpolation* (TLI) and *Text Embedding Interpolation* (TEI), to recombine skills learned from different tasks, suggesting that the learned skills are composable and can be used for extrapolated tasks. We benchmark our method on the proposed *libero-ood* task suite, where π_0 augmented with TLI achieves an 83% success rate, while all existing SOTA VLAs achieve less than a 15% success rate. This result proves that the π_0 is actually capable of finishing those extrapolated tasks, as we simply recombine its learned latent features to make it work. This justifies the design of *libero-ood* and we hope it can help the community design more generalizable policies. Our analysis also reveals spatial overfitting existing in VLAs, where the policy learn to map the object name to a specific location that it has been placed in the demonstration.

Limitations. Unlike LLM, there is no unified training recipe and model architecture in the VLA community. We don’t study at the university of the proposed method in this work on different VLAs.

Societal Impact. Though our method can be used to augment existing VLA’s capability, obscure prompts used to reconstruct tasks potentially induce a negative societal impact. Especially, for an open-sourced VLA, the alternative human-unreadable prompt can be identified with our method and used for adversarial attack in the safety-critical robot manipulation environment.

References

- [1] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations, 2017.
- [2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- [3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Chormanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [4] Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 2020. <https://distill.pub/2020/circuits>.
- [5] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.
- [6] Zane Durante, Bidipta Sarkar, Ran Gong, Rohan Taori, Yusuke Noda, Paul Tang, Ehsan Adeli, Shrinidhi Kowshika Lakshmikanth, Kevin Schulman, Arnold Milstein, et al. An interactive agent foundation model. *arXiv preprint arXiv:2402.05929*, 2024.
- [7] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. <https://transformer-circuits.pub/2021/framework/index.html>.
- [8] Andrew Sohn et al. Introducing rfm-1: Giving robots human-like reasoning capabilities, 2024.
- [9] Yossi Gandelsman, Alexei A. Efros, and Jacob Steinhardt. Interpreting clip’s image representation via text-based decomposition, 2024.
- [10] Zhi Hou, Tianyi Zhang, Yuwen Xiong, Hengjun Pu, Chengyang Zhao, Ronglei Tong, Yu Qiao, Jifeng Dai, and Yuntao Chen. Diffusion transformer policy: Scaling diffusion transformer for generalist vision-language-action learning, 2025.
- [11] Jiangyong Huang, Silong Yong, Xiaojian Ma, Xiongkun Linghu, Puhao Li, Yan Wang, Qing Li, Song-Chun Zhu, Baoxiong Jia, and Siyuan Huang. An embodied generalist agent in 3d world. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.
- [12] Nick Jiang, Anish Kachinthaya, Suzie Petryk, and Yossi Gandelsman. Interpreting and editing vision-language representations to mitigate hallucinations, 2025.
- [13] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025.
- [14] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.
- [15] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

- [16] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation, 2024.
- [17] Quanyi Li, Zhenghao Peng, Haibin Wu, Lan Feng, and Bolei Zhou. Human-ai shared control via policy dissection, 2023.
- [18] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023.
- [19] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- [20] Hong Lu, Hengxu Li, Prithviraj Singh Shahani, Stephanie Herbers, and Matthias Scheutz. Probing a vision-language-action model for symbolic states and integration into a cognitive architecture, 2025.
- [21] Grace Luo, Trevor Darrell, and Amir Bar. Task vectors are cross-modal, 2024.
- [22] nostalgebraist. Interpreting GPT: The logit lens. LessWrong, Aug 2020.
- [23] NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" 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.
- [24] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. <https://distill.pub/2020/circuits/zoom-in>.
- [25] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. <https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html>.
- [26] Abby O’Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.
- [27] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models, 2025.
- [28] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, Jia Yuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. Spatialvla: Exploring spatial representations for visual-language-action model, 2025.
- [29] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- [30] Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. A practical review of mechanistic interpretability for transformer-based language models, 2025.
- [31] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann,

- Arunkumar Byravan, Serkan Cabi, Ken Caluwaerts, Federico Casarini, Oscar Chang, Jose Enrique Chen, Xi Chen, Hao-Tien Lewis Chiang, Krzysztof Choromanski, David D’Ambrosio, Sudeep Dasari, Todor Davchev, Coline Devin, Norman Di Palo, Tianli Ding, Adil Dostmohamed, Danny Driess, Yilun Du, Debidatta Dwibedi, Michael Elabd, Claudio Fantacci, Cody Fong, Erik Frey, Chuyuan Fu, Marissa Giustina, Keerthana Gopalakrishnan, Laura Graesser, Leonard Hasenclever, Nicolas Heess, Brandon Hernaez, Alexander Herzog, R. Alex Hofer, Jan Humplik, Atil Iscen, Mithun George Jacob, Deepali Jain, Ryan Julian, Dmitry Kalashnikov, M. Emre Karagozler, Stefani Karp, Chase Kew, Jerad Kirkland, Sean Kirmani, Yuheng Kuang, Thomas Lampe, Antoine Laurens, Isabel Leal, Alex X. Lee, Tsang-Wei Edward Lee, Jacky Liang, Yixin Lin, Sharath Maddineni, Anirudha Majumdar, Assaf Hurwitz Michaely, Robert Moreno, Michael Neunert, Francesco Nori, Carolina Parada, Emilio Parisotto, Peter Pastor, Acorn Pooley, Kanishka Rao, Krista Reymann, Dorsa Sadigh, Stefano Saliceti, Pannag Sanketi, Pierre Sermanet, Dhruv Shah, Mohit Sharma, Kathryn Shea, Charles Shu, Vikas Sindhwani, Sumeet Singh, Radu Soricut, Jost Tobias Springenberg, Rachel Sterneck, Razvan Surdulescu, Jie Tan, Jonathan Tompson, Vincent Vanhoucke, Jake Varley, Grace Vesom, Giulia Vezzani, Oriol Vinyals, Azyaan Wahid, Stefan Welker, Paul Wohlhart, Fei Xia, Ted Xiao, Annie Xie, Jinyu Xie, Peng Xu, Sichun Xu, Ying Xu, Zhuo Xu, Yuxiang Yang, Rui Yao, Sergey Yaroshenko, Wenhao Yu, Wentao Yuan, Jingwei Zhang, Tingnan Zhang, Allan Zhou, and Yuxiang Zhou. Gemini robotics: Bringing ai into the physical world, 2025.
- [32] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.
 - [33] Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models, 2024.
 - [34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
 - [35] Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers, 2024.
 - [36] Taowen Wang, Cheng Han, James Chenhao Liang, Wenhao Yang, Dongfang Liu, Luna Xinyu Zhang, Qifan Wang, Jiebo Luo, and Ruixiang Tang. Exploring the adversarial vulnerabilities of vision-language-action models in robotics, 2025.
 - [37] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2409.12514*, 2024.
 - [38] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training, 2023.
 - [39] Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.
 - [40] Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang, and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies, 2024.
 - [41] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns, 2015.

Appendix

A LIBERO-Benchmark

The *libero-goal* suite evaluates goal-completion capabilities, reflecting procedural knowledge (knowing how to complete a task). Conversely, the *libero-object* and *libero-spatial* suite assess object recognition, localization, and interaction capabilities, highlighting declarative knowledge (understanding entities and concepts). Tasks in *libero-object* share the same goal, picking up a specific object and placing it into a basket, but differ in object layouts. In contrast, *libero-goal* tasks require policies to achieve various goals within the same scene layout. The *libero-spatial* instead operates in different layouts to test whether the policy can find the correct bowl to pick and place it on the plate.

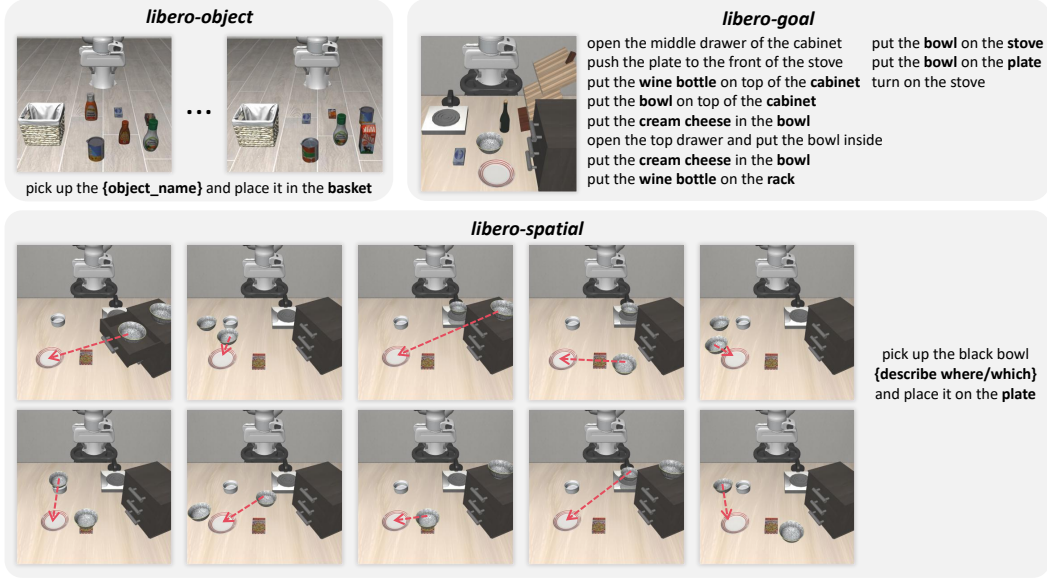


Figure 4: The three basic libero task suites used to build extrapolated tasks. The *libero-object* tasks ask the robot to pick a specific object and place it in the basket; The *libero-goal* tasks operate in the same scene and require the robot to finish several different tasks where 7 of them are pick and place tasks and will be used to finish extrapolated tasks; The *libero-spatial* aims to test whether the robots can understand the space, and thus asks the robot to pick the specified bowl and place it on the plate.

B Unembedded Prompts for *libero-goal* and *libero-object*

For *libero-goal*, we use \mathcal{T}_1 to get the alternative prompt, while for *libero-object*, we use \mathcal{T}_2 .

libero-object	libero-goal
['a0', 'a0', 'U000e0062', 'a0', 'sauc', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['U000e0062', 'U000e0062', 'wine', 'bottle', 'U000e0062', 'Top', 'QSize', 'U000e0062', 'cabinet']
['a0', 'a0', 'U000e0062', 'QSize', 'soup', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['U000e0062', 'U000e0062', 'a0', 'U000e0062', 'U000e0062', 'plate']
['a0', 'a0', 'U000e0062', 'milk', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['U000e0062', 'U000e0062', 'a0', 'U000e0062', 'U000e0062', 'stove']
['a0', 'a0', 'U000e0062', 'salad', 'dressing', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['U000e0062', 'U000e0062', 'cream', 'cheese', 'U000e0062', 'U000e0062', 'a0']
['a0', 'a0', 'U000e0062', 'a0', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['Open', 'U000e0062', 'middle', 'ValueError', 'QSize', 'U000e0062', 'cabinet']
['a0', 'a0', 'U000e0062', 'cream', 'a0', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['push', 'U000e0062', 'plate', 'QSize', 'U000e0062', 'front', 'QSize', 'U000e0062', 'stove']
['a0', 'a0', 'U000e0062', 'orange', 'a0', 'QSize', 'QSize', 'U000e0062', 'U000e0062', 'QSize']	['Turn', 'QSize', 'U000e0062', 'stove']
['a0', 'a0', 'U000e0062', 'a0', 'a0', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['Open', 'U000e0062', 'Top', 'Einf', 'QSize', 'Einf', 'U000e0062', 'a0', 'inside']
['a0', 'a0', 'U000e0062', 'a0', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'basket']	['U000e0062', 'U000e0062', 'wine', 'bottle', 'U000e0062', 'U000e0062', 'rack']
['a0', 'a0', 'U000e0062', 'QSize', 'sauc', 'QSize', 'QSize', 'a0', 'U000e0062', 'U000e0062', 'QSize']	['U000e0062', 'U000e0062', 'a0', 'U000e0062', 'Top', 'QSize', 'U000e0062', 'cabinet']

Table 5: The alternative prompts for *libero-goal* and *libero-object*. It is hard to understand them, even if one knows the original prompts shown in Appendix A,

C Behavior Visualization

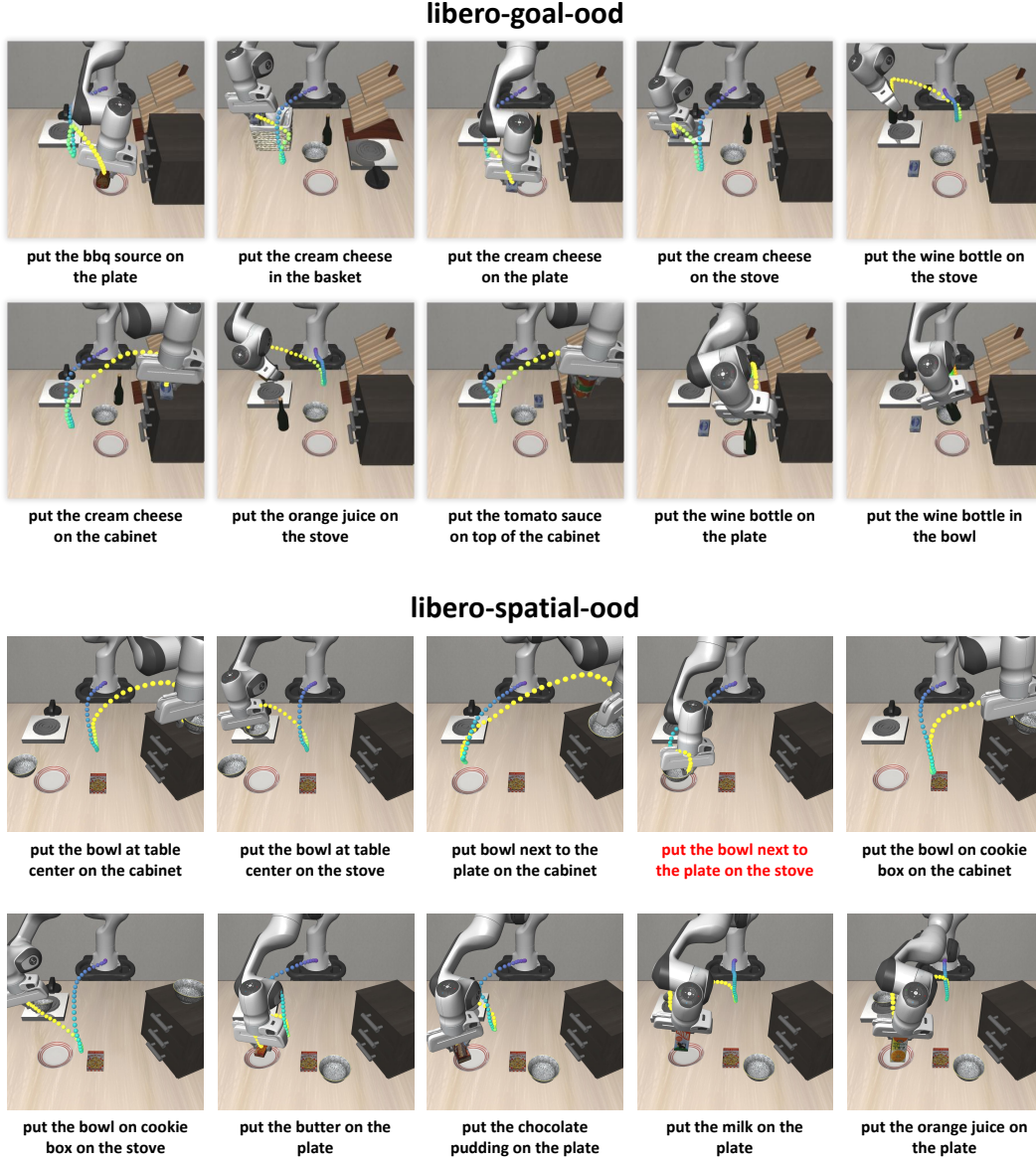


Figure 5: Trajectories of finishing tasks in *libero-ood*. The only failed task is highlighted in red. A fun fact is that these colored dots indicating the trajectory can be observed by the π_0 as well, while it is still robust to this visual perturbation. The only failure case is *put the bowl next to the plate on the stove*. It keeps picking up the bowl next to the plate and placing it on the plate. We suspect that the second task context is not strong enough to trigger the behavior "put on the stove", or the context of the first task is too strong to be fully removed from the residual stream, so the policy keeps implementing the behaviors of the first task and put the bowl on the plate.

D Detail Task Performance

libero-goal-ood

	Put The Bbq Source On The Plate	Put The Cream Cheese In The Basket	Put The Cream Cheese On The Plate	Put The Cream Cheese On The Stove	Put The Cream Cheese On Top Of The Cabinet	Put The Orange Juice On The Stove	Put The Tomato Sauce On Top Of The Cabinet	Put The Wine Bottle In The Bowl	Put The Wine Bottle On The Plate	Put The Wine Bottle On The Stove	Total Success Rate (%)
open/ia-ofc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
π_{0^-} fast	0.1	0.0	1.0	0.0	0.2	0.0	0.0	0.0	0.1	0.0	0.14
π_0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.02
π_{0^-} TEI	0.9	0.8	0.2	0.5	0.3	0.7	0.6	0.0	0.1	0.1	0.42
π_{0^-} TLI	1.0	0.9	1.0	1.0	1.0	0.7	1.0	0.5	0.8	0.6	0.85
π_{0^-} TEI-TLI	0.9	0.8	1.0	1.0	0.9	0.8	0.9	0.6	0.7	0.9	0.85
π_{0^-} TLI*	0.0	0.0	0.2	0.4	1.0	0.3	0.6	0.2	0.5	0.1	0.33

libero-spatial-ood

	Put The Bowl At Table Center On The Cabinet	Put The Bowl At Table Center On The Stove	Put The Bowl Next To The Plate On The Cabinet	Put The Bowl Next To The Plate On The Stove	Put The Bowl On Cookie Box On The Cabinet	Put The Bowl On Cookie Box On The Stove	Put The Butter On The Plate	Put The Chocolate Pudding On The Plate	Put The Milk On The Plate	Put The Orange Juice On The Plate	Total Success Rate (%)
open/ia-ofc	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
π_{0^-} fast	0.0	0.1	0.0	0.0	0.0	0.0	0.0	1.0	0.4	0.2	0.17
π_0	0.0	0.1	0.0	0.0	0.0	0.0	0.6	0.7	0.2	0.0	0.16
π_{0^-} TEI	0.1	0.6	0.0	0.1	0.0	0.1	0.5	1.0	0.9	0.8	0.41
π_{0^-} TLI	0.7	1.0	0.8	0.0	1.0	0.8	1.0	0.9	1.0	0.9	0.81
π_{0^-} TEI-TLI	0.4	0.9	0.4	0.0	0.4	0.9	0.9	1.0	1.0	1.0	0.69
π_{0^-} TLI*	0.0	0.4	0.2	0.0	0.2	0.2	0.2	0.5	0.1	0.0	0.18