EXPLOITING MODEL EMBEDDING MEMORY IN POLICY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Adapting a pretrained model to downstream tasks produces a collection of task-specific adaptations that can serve as reusable model memory. However, existing weight-space representations, such as full-parameter or low-rank, are primarily designed for optimization rather than memory organization: they lack a shared representation in which task-specific adaptations can be compactly stored, compared, and retrieved. In this work, we propose a *model embedding space*, a low-dimensional representation of task adaptations constructed in alignment with the intrinsic structure of pretrained weights. This space provides a shared coordinate system for organizing task-specific knowledge into a *model embedding memory*, enabling compact storage and similarity-based retrieval across adapted models. To make this memory both efficient to search and easy to access from other modalities, we further introduce a *model embedding index*: a lightweight representation that preserves retrieval-relevant information while forming a compact interface between model embeddings and language representations. We demonstrate the framework in robot policy learning, where task-specific policies are encoded as model embeddings and retrieved from memory for target tasks. We further enable language-guided retrieval by mapping LLM embeddings to the model embedding index, showing that the model embedding space provides a structured memory for storing and reusing task-specific policies.

1 INTRODUCTION

Pre-trained models are increasingly adapted to many downstream tasks, producing a collection of task-specific adaptations (Dosovitskiy et al., 2021; Rombach et al., 2022; Vaswani et al., 2017). With an increasing number of tasks, these adaptations function as a form of reusable model memory. However, conventional weight-space representations are poorly suited for this purpose. Full-parameter updates are expensive to store, while low-rank adaptations reduce the parameter count but still live in the ambient weight space (Hu et al., 2022; Liu et al., 2024). As a result, these representations are often inefficient for large-scale memory construction and costly to compare for retrieval.

In this work, we introduce a *model embedding space* designed to organize task-specific adaptations. By leveraging the intrinsic structure of pre-trained weights, we construct a low-dimensional space based on the premise that task adaptation typically occurs within specific parameter subspaces, as shown in Figure 1 (a-c). Given a pre-trained weight, we project downstream adaptations into compact embeddings using bases derived directly from the pre-trained weights. This framework treats task adaptation as a collection of operations within a shared space, establishing a *model embedding memory*. As these embeddings share a common coordinate system, the memory supports intuitive geometric operations, such as efficient retrieval (Figure 1 (d)).

To further improve retrieval efficiency, we introduce a *model embedding index*. While model embeddings are already compact, the model embedding index is a significantly lightweight representation designed to preserve the similarity structure of the original model embedding space. In practice, its similarity is highly correlated with full model embedding similarity, allowing task-relevant adaptations to be retrieved efficiently without directly comparing the full embeddings.

This framework supports memory-based policy learning. In Section 4, we demonstrate it in robot policy learning, where we adapt task-specific actors using model embeddings and store them in 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.

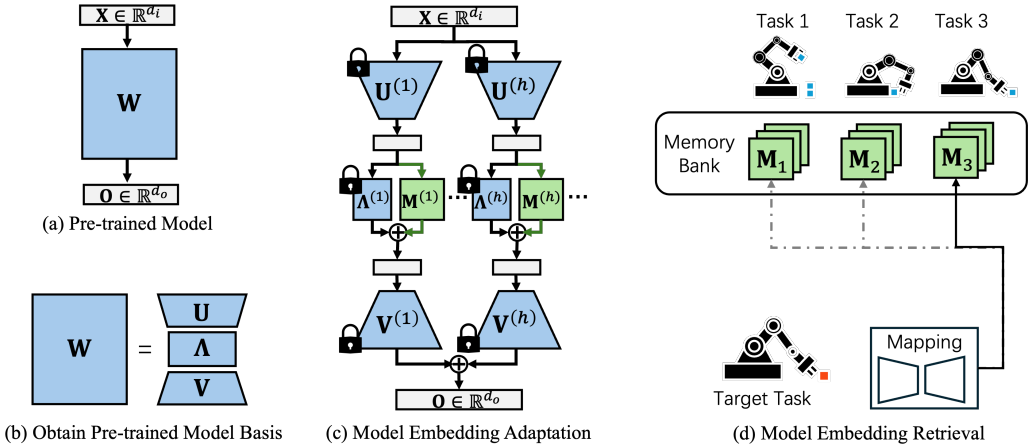


Figure 1: **Illustration of Model Embeddings and Constructed Memory.** Based on the (a) pre-trained model, (b) we construct it as a basis. (c) We enforce the adaptation to the downstream task within such a subspace. (d) Memory and retrieval in the model embedding space, where distinct tasks are mapped to a *model embedding memory*; the adaptation weights of target tasks can be efficiently retrieved via a *model embedding index*. The detailed discussion is provided in Section 3.

model embedding memory. Given a target action or task requirement, the system retrieves relevant actors through model embedding similarity or the more efficient model embedding index. We further show that the model embedding memory can be connected with language guidance. By mapping language model embeddings into the model embedding index space, language descriptions can be used to retrieve task-relevant model embeddings. This provides a path toward language-conditioned policy retrieval, where high-level instructions guide the selection of reusable task-specific actors.

Our contributions are summarized as follows:

- We propose a *model embedding space* that represents task-specific adaptations as compact embeddings constructed from the intrinsic structure of pre-trained weights.
- We introduce *model embedding memory*, a framework where task adaptations are mapped as embeddings within a shared coordinate system, enabling compact storage and similarity-based retrieval.
- We introduce a *model embedding index* that preserves retrieval-relevant information while forming a compact interface between model embeddings and language representations.

2 Structured Model Embedding Space

A pre-trained model is not arbitrary in parameter space: its parameters encode directions that are useful for the pre-trained task. We therefore use the pre-trained model to define a structured coordinate system for downstream adaptation.

For a linear layer with pre-trained parameter $\mathbf{W} \in \mathbb{R}^{d_o \times d_i}$. We characterize the structure of the pretrained model through the singular value decomposition (SVD) of its linear mapping,

$$\mathbf{W} = \mathbf{U}\Lambda\mathbf{V}^\top,$$

where $\mathbf{U} \in \mathbb{R}^{d_o \times r}$, $\mathbf{V} \in \mathbb{R}^{r \times d_i}$, and $\mathbf{\Lambda} \in \mathbb{R}^{r \times r}$. $r = \min(d_i, d_o)$ is the maximum rank of $\mathbf{W}$. We represent the task-specific update as

$$\Delta\mathbf{W} = \mathbf{U}\mathbf{M}\mathbf{V}^\top, \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{r \times r}$ contains the coordinates of the downstream task in the pretrained model's basis. We call $\mathbf{M}$ the *model embedding*. Figure 1 (a-c) illustrates the construction of model embeddings.

This view provides the main motivation of our representation: instead of comparing high-dimensional adapted models directly, we represent each downstream task by how it changes a shared

pretrained model. Because all tasks are expressed in the same pretrained coordinate system, their embeddings are compact, comparable, and can be organized for efficient storage and retrieval.

A linear-Gaussian example in Appendix A further shows that, when the downstream change lies in the pretrained subspace, such coordinates are sufficient to realize the adaptation, while avoiding unnecessary out-of-subspace changes.

2.1 Adaptation in Model Embedding Space

To impose a structured low-rank adaptation, we first define a block dimension r_h. We then partition the latent bases $\mathbf{U}$ and $\mathbf{V}$ into non-overlapping blocks of width r_h,

$$\mathbf{U} = \left[\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(H_{\max})}\right], \qquad \mathbf{V} = \left[\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(H_{\max})}\right], \tag{2}$$

where $\mathbf{U}^{(h)}$ and $\mathbf{V}^{(h)}$ contain r_h basis directions, and $H_{\max} = \lfloor r/r_h \rfloor$ denotes the maximum number of available blocks.

For each adapted model, we select $H \leq H_{\max}$ blocks and associate each selected block with a model embedding $\mathbf{M}^{(h)} \in \mathbb{R}^{r_h \times r_h}$. The complete model embedding is therefore represented as

$$\mathbf{M} = \left(\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(H)}\right). \tag{3}$$

The corresponding parameter update is

$$\Delta\mathbf{W} = \sum_{h=1}^{H} \mathbf{U}^{(h)} \mathbf{M}^{(h)} \mathbf{V}^{(h)\top}. \tag{4}$$

Each term is supported on an r_h-dimensional pair of basis subspaces. Consequently, $\mathrm{rank}(\Delta\mathbf{W}) \leq Hr_h$, while the number of trainable parameters in the model embedding is Hr_h^2. Thus, with r_h fixed, selecting fewer blocks directly reduces both the maximum adaptation rank and the number of trainable parameters, providing a simple mechanism for controlling the capacity and efficiency of adaptation.

The resulting structure of the model embeddings is illustrated in Figure 1 (c). Because different blocks are constructed from disjoint basis directions, their induced updates occupy mutually orthogonal subspaces when the underlying bases are orthonormal. For the case of multi-head attention (MHA), a detailed derivation of the model embedding is provided in Appendix B.1.

2.2 The Metric in Model Embedding Space

To analyze the structure of the model embedding space, we introduce a metric that enables comparison between different task embeddings. Let $\mathbf{M}_i, \mathbf{M}_j$ denote two model embeddings corresponding to different tasks. We define an inner product in the embedding space as

$$\langle \mathbf{M}_i, \mathbf{M}_j \rangle = \mathrm{Tr}(\mathbf{M}_i^\top \mathbf{M}_j), \tag{5}$$

which corresponds to the standard Frobenius inner product over matrices. This induces the norm $\|\mathbf{M}\|_F = \sqrt{\mathrm{Tr}(\mathbf{M}^\top \mathbf{M})}$ and the corresponding distance metric $d(\mathbf{M}_i, \mathbf{M}_j) = \|\mathbf{M}_i - \mathbf{M}_j\|_F$.

With this structure, the model embedding space becomes a Hilbert space, allowing us to analyze geometric relationships between tasks. In particular, we define the cosine similarity between two model embeddings as

$$\mathrm{sim}(\mathbf{M}_i, \mathbf{M}_j) = \frac{\langle \mathbf{M}_i, \mathbf{M}_j \rangle}{\|\mathbf{M}_i\|_F \|\mathbf{M}_j\|_F}. \tag{6}$$

2.3 The Relation With Task Space

We consider a family of robotic tasks indexed by $\tau \in \mathcal{T}$. Each task is characterized by task-dependent dynamics P_τ and objective R_τ,

$$\tau = (P_\tau, R_\tau), \tag{7}$$



(a) (b) (c)

Figure 2: **Functional relationship between model embeddings and task space.** (a) illustrates the task space. Learned policies successfully adapt to the target tasks. (b) The policies produce task-dependent commands that compensate for the corresponding actuator transformations. (c) illustrates the model embeddings. Different model embeddings induce distinct output signals in (b), resulting in different behaviors in the task space.

where P_τ specifies how the robot and environment evolve under actions, and R_τ specifies the desired behavior.

Let π_i and π_j denote the policy models adapted to tasks τ_i and τ_j, respectively. We characterize the distance between two tasks through their induced behaviors,

$$d_{\mathcal{T}}(\tau_i, \tau_j) = \left(\mathbb{E}_{s\sim\mu}\left[\|\pi_i(s) - \pi_j(s)\|_2^2\right]\right)^{1/2}, \tag{8}$$

where μ is a shared state distribution over which the two task-specific policies are compared.

Assumption 2.1. *For all parameters in a neighborhood of the pretrained model containing the considered task adaptations, the policy satisfies*

$$\left\|\pi_{\mathbf{W}_i}(s) - \pi_{\mathbf{W}_j}(s)\right\|_2 \leq L(s)\,\|\mathbf{W}_i - \mathbf{W}_j\|_F\,, \tag{9}$$

for some state-dependent constant $L(s)$.

Proposition 2.2. *Let $C = \left(\mathbb{E}_{s\sim\mu}[L(s)^2]\right)^{1/2}$. Under Assumption 2.1, the behavioral distance between two tasks is bounded by the distance between their model embeddings:*

$$d_{\mathcal{T}}(\tau_i, \tau_j) \leq C\,\|\mathbf{M}_i - \mathbf{M}_j\|_F\,. \tag{10}$$

Remark 2.3. *Two task adaptations that are close in model embedding space cannot induce arbitrarily different behaviors.*

The details are in Appendix A.3.

2.4 Localization of Model Embeddings

We next show that if two tasks are distinct in task space, then their model embeddings are also separated.

Assumption 2.4. *For two distinct tasks τ_i and τ_j, their task-induced functional shifts satisfy*

$$d_{\mathcal{T}}(\tau_i, \tau_j) \geq \gamma_{ij}, \tag{11}$$

where $\gamma_{ij} > 0$ measures the intrinsic task gap.

Theorem 2.5 (Distinct tasks remain separated in the model embedding space)**.** *Under Assumption 2.4 and Proposition 2.2, if two tasks τ_i and τ_j are distinct in task space with margin γ_{ij}, then their model embeddings satisfy*

$$\|\mathbf{M}_i - \mathbf{M}_j\|_2 \geq \frac{\gamma_{ij}}{C}. \tag{12}$$

Equivalently, distinct task-space regions correspond to separated regions in model embedding space.

The relationship between the task space and the model embedding space is illustrated in Figure 2 (b-c). Figure 2 (b) shows that local variations in the model embedding induce corresponding local changes in the distributional shift. Figure 2 (c) further demonstrates that combining multiple model embeddings allows the model to represent a more complex task space.

2.5 Toy Reacher Dynamics Experiment

We study a two-link planar reacher under different actuator dynamics (g_1, g_2). The policy command $\mathbf{a}_t$ is transformed into the applied action

$$\widetilde{\mathbf{a}}_t = \mathrm{diag}(g_1, g_2)\mathbf{a}_t.$$

We have 4 different actuator dynamics $(g_1, g_2) \in \{(+1, +1), (-1, +1), (+1, -1), (-1, -1)\}$, corresponding to nominal control, reversed joint 1, reversed joint 2, and both joints reversed. A Soft Actor-Critic (SAC) policy (Haarnoja et al., 2018) is pretrained on the nominal task. For each task, the base policy is frozen, and only its model embedding is adapted, which contains 128 trainable parameters.

Figure 2(a) shows that the learned policies successfully adapt to the target tasks, while Figure 2(b) shows that they generate task-dependent commands that compensate for the corresponding actuator transformations. Figure 2(c) visualizes parts of model embeddings, and we find they contribute to different joints, including joint 1 (J_1), joint 2 (J_2) and their interaction $J_1 \times J_2$. This demonstrates that structured changes in physical dynamics produce corresponding joint-specific and interaction-specific structure in parameter space.

3 Memory and Retrieval

3.1 Model Embedding Memory (MEM)

The compactness of model embeddings enables efficient storage of task-specific knowledge. We construct a *model embedding memory (MEM)* that stores embeddings corresponding to N tasks:

$$\mathcal{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_N\}, \tag{13}$$

where each $\mathbf{M}_i$ represents the model embedding associated with task i.

3.2 Retrieval Mechanism

Given a query task, our goal is to retrieve the most relevant model embedding from the model embedding memory. Let $\mathbf{M}_q$ denote the embedding associated with the query. Using the similarity metric (6), we compute the cosine similarity between $\mathbf{M}_q$ and each stored embedding $\mathrm{sim}(\mathbf{M}_q, \mathbf{M}_i)$.

We then select the embedding with the highest similarity:

$$\mathbf{M}_s = \arg\max_{\mathbf{M}_i \in \mathcal{M}} \mathrm{sim}(\mathbf{M}_q, \mathbf{M}_i). \tag{14}$$

The retrieved embedding $\mathbf{M}_s$ is subsequently applied to the pretrained model to adapt it to the query task.

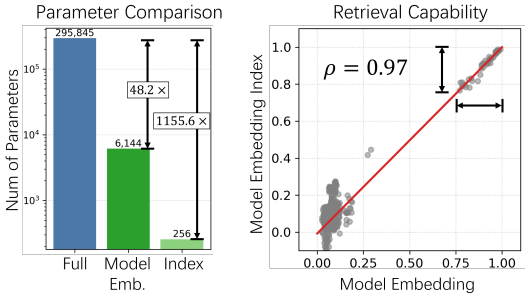


Figure 3: The model embedding index significantly reduces the required parameters while preserving the same retrieval capability as the original model embedding. This is evidenced by high Spearman correlation, i.e., $\rho = 0.97$.

Model embedding index Consider the block-diagonal structure of model embedding $\mathbf{M}_i = \mathrm{blockdiag}(\mathbf{M}_i^{(1)}, \ldots, \mathbf{M}_i^{(H)})$, we define the model embedding index using a single block, *e.g.*,

$$\boldsymbol{\iota}_i := \mathrm{vec}(\mathbf{M}_i^{(h)}). \tag{15}$$

In practice, we find that different blocks work equally well for model retrieval. The model embedding index is in the space $\boldsymbol{\iota}_i \in \mathcal{I}$. We can obtain the corresponding model embedding from the index using a mapping $\boldsymbol{\iota}_i \to \mathbf{M}_i$. More empirical choices of model embedding index are provided in Appendix B.3.

Retrieval is then performed using

$$\mathrm{sim}(\boldsymbol{\iota}_q, \boldsymbol{\iota}_i) = \frac{\boldsymbol{\iota}_q^\top \boldsymbol{\iota}_i}{\|\boldsymbol{\iota}_q\|_2 \|\boldsymbol{\iota}_i\|_2}. \tag{16}$$

Table 1: Model-embedding capacity and parameter efficiency. **(a)** Model embedding can efficiently learn new tasks. It achieves a comparable success rate while using substantially fewer parameters than LoRA. **(b)** Model embedding further reduces the parameters by selecting fewer blocks.

(a) Adaptation capability vs. number of parameters

	Model Embedding		LoRA (Hu et al., 2022)	
r_h or r_l	Params. ↓	Success (%) ↑	Params. ↓	Success (%) ↑
4	6,144	87.90 ± 1.55	14,336	87.63 ± 0.60
8	8,192	87.67 ± 0.57	28,672	87.53 ± 0.90
16	12,288	88.90 ± 0.10	57,344	$\mathbf{91.77 \pm 1.58}$
32	20,480	$\mathbf{90.40 \pm 1.13}$	114,688	88.03 ± 0.31
Zero-shot		0 Params.		25.20

(b) Block selection: $H_{max} = 8$, which corresponds to $r_h = 16$. The effective rank is Hr_h

H	Params. ↓	Success (%) ↑
2	3,072	85.37 ± 1.00
4	6,144	87.17 ± 0.64
6	9,216	$\mathbf{89.70 \pm 0.87}$
8	12,288	88.90 ± 0.50

As illustrated in Figure 3, the proposed model embedding index significantly reduces the number of parameters while preserving retrieval performance comparable to the full model. Empirically, we observe that task similarities measured in the compressed index space closely match those in the original embedding space, achieving a Spearman correlation of 0.97.

3.3 Retrieval Using Language Instruction

When task descriptions or natural language instructions are available, we can leverage language-model-based embeddings k^{text}. To enable comparison within the model embedding space, we learn a projection function ψ to map the text embedding into model embedding space,

$$\tilde{\boldsymbol{\iota}} = \psi(k^{\text{text}}) \in \mathcal{I}, \tag{17}$$

where $\tilde{\boldsymbol{\iota}}$ is the projected language guidance in the model embedding index space. This enables instruction-guided retrieval via

$$\mathbf{M}_s = \arg \max_{\boldsymbol{\iota}_i \to \mathbf{M}_i \in \mathcal{M}} \text{sim}(\tilde{\boldsymbol{\iota}}, \boldsymbol{\iota}_i). \tag{18}$$

to retrieve relevant model embeddings $\mathbf{M}_s$ from memory. Therefore, the memory can be controlled not only by parameter similarity, but also by semantic task descriptions.

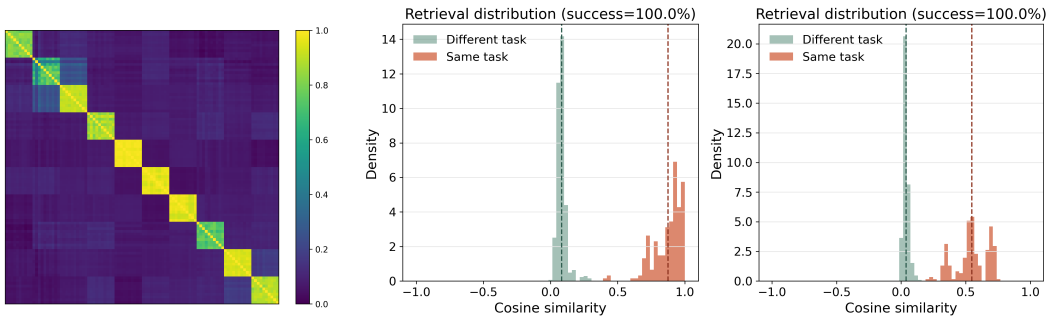
4 Experiments

4.1 Efficient Model Embedding Memory and Retrieval

In this section, we utilize Meta-World (Yu et al., 2019) benchmark to demonstrate the effectiveness of model embedding memory and retrieval mechanism.

Experimental settings. We evaluate memory efficiency and retrieval behavior on Meta-World MT50 using a pretrained causal policy Transformer backbone. The backbone is first pretrained on MT10 and then kept frozen while adapting to the remaining 40 MT50 tasks using either LoRA (Hu et al., 2022) adapters or model embeddings with 10 random seeds. Both methods use identical demonstration data and are evaluated under closed-loop rollout conditions, isolating the effect of the task representation from training or data differences.

Efficient model embedding memory. **(1)** As shown in Table 1 (a), model embedding adaptation achieves performance comparable to LoRA while using substantially fewer trainable parameters. This indicates that model embeddings provide a compact representation of task-specific adaptation. **(2)** Due to the more structured geometry of the model embedding space, we can further reduce the storage cost and enable a more compact model embedding memory by selecting part of the blocks, as shown in Table 1 (b). **(3)** As model embeddings require much fewer parameters, they introduce a much slower growth in memory size, making them more suitable for constructing a scalable model embedding memory.



(a) Inter/Intra Task Similarity (b) Model Embedding (c) LoRA

Figure 4: **Comparative analysis of task discriminability.** We visualize pairwise cosine similarity between task-specific adaptations across 10 tasks and 10 random seeds per task. (a) The model embedding space exhibits high intra-task clustering and distinct inter-task separation. (b) Model embeddings demonstrate a more task-consistent geometry. Compared to (c) LoRA, same-task pairs of model embedding exhibit higher cosine similarity, yielding sharper separation from different-task pairs and increased retrieval confidence.

Table 2: Effective model embedding memory and efficient retrieval: The model embedding index requires only 0.09% of the parameters used by LoRA and enables $17\times$ faster retrieval.

	Stored Parameters	Retrieval Time (ms)	FLOPs	Accuracy
LoRA (Hu et al., 2022)	262,144	1.93	$(2r_l + 4N)Ld^2$	93.9
Model Embedding	6,144	0.13	$4NHLr^2$	93.4
Model Embedding Index	256	0.11	$4Nr^2$	-

Demonstration of retrieval mechanism. In this experiment, following (13), model embeddings are used as queries to retrieve similar embeddings from memory. Specifically, checkpoints from 7 random seeds are stored in the memory bank, while checkpoints from the remaining 3 seeds are used as queries. In Figure 4 (a), we visualize pairwise cosine similarity between task-specific adaptations across all tasks and seeds. It shows that model embeddings form well-separated, seed-consistent task clusters with high intra-task similarity and low inter-task similarity. In Figure 4(b-c), we compare the cosine similarity distributions of same-task and different-task retrieval pairs across the checkpoints. Model embeddings exhibit a more task-consistent geometry than LoRA: same-task pairs are concentrated at high cosine similarity, leading to clearer separation from different-task pairs and higher retrieval confidence. After reconstructing LoRA updates into weight space, LoRA also captures task identity to some extent; however, its similarity scores are lower and more dispersed.

Model embedding index leads to much faster retrieval. Table 2 compares LoRA and model embeddings, where r_l denotes the LoRA rank and L denotes the number of layers. Compared with LoRA, model embeddings require fewer parameters for storage, lower retrieval time, and fewer FLOPs, while achieving comparable accuracy. Using the model embedding index further reduces retrieval cost in both time and computation.

4.2 Demonstration on LIBERO and RoboCasa with Language Instruction

Experimental settings. We evaluate the end-to-end effectiveness of model embedding memory on all five LIBERO (Liu et al., 2023) suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, LIBERO-10, and LIBERO-90. For each suite, an MTACT (Bharadhwaj et al., 2024) backbone is pretrained on LIBERO-90, and per-task model embeddings are obtained by fine-tuning only the model embedding parameters on each task's 50 expert demonstrations. The resulting embeddings are stored in a model embedding memory and can be retrieved using model embedding indices. We compare against MTACT (Bharadhwaj et al., 2024), Diffusion Policy (Chi et al., 2025), OpenVLA (Kim et al., 2024), and Octo (Octo Model Team et al., 2024). Success is measured as binary

Table 3: Retrieval accuracy using language instruction.

	Explicit	Conversational	Polite	Goal	Synonym	Overall
Top-1 (%)	97.5	85.0	95.0	95.0	97.5	**94.0**
Top-3 (%)	100	97.5	100	100	100	**99.5**

Table 4: Comparison on LIBERO dataset.

	LIBERO 90	LIBERO 10	Spatial	Object	Goal	Params. ↓
Number of tasks (N)	90	10	10	10	10	-
MTACT (Bharadhwaj et al., 2024)	54	34.3	71.2	76.8	83.1	53M
Diffusion Policy (Chi et al., 2025)	-	50.5	78.3	**92.5**	68.3	89M
Octo (Octo Model Team et al., 2024)	-	51.1	78.9	85.7	<u>84.6</u>	93M
OpenVLA (Kim et al., 2024)	-	<u>53.7</u>	<u>84.7</u>	88.4	79.2	7B
Ours (MTACT+MEM)	**86.6** (+32.6)	**71** (+40.7)	**91.5** (+20.3)	<u>88.5</u> (+11.7)	**90.5** (+7.4)	53M+N*0.9M

task completion averaged over 20 rollouts per task following the standard LIBERO protocol. More experimental details are in Appendix C.

Retrieval using language instruction. Task descriptions are embedded using a frozen MiniLM (Wang et al., 2020) text encoder and projected into the model embedding index space via a learned *linear map* trained on paired of {text embedding, index} examples from the training tasks. At test time, only the projected language embedding is used as the query to retrieve the model embedding from the memory. We evaluate this retrieval mechanism on 200 held-out instructions. More details are in Appendix C.7. Table 3 shows that the proposed language-token router achieves 94.0% top-1 and 99.5% top-3 retrieval accuracy.

Improved performance with model embedding memory and retrieval. As shown in Table 4, our approach (MTACT + model embedding memory) consistently outperforms or matches all baselines across every LIBERO suite while using fewer task-specific parameters. We select explicit language instructions to retrieve the target adapters. Compared with MTACT, model embedding memory achieves an average success-rate gain of $+22.54$ by storing compact task-specific embeddings for each subtask, confirming that task-specific retrieval substantially improves performance.

Demonstration on RoboCasa dataset. To further evaluate the effectiveness of model embedding memory, we conduct experiments on RoboCasa (Nasiriany et al., 2026), which contains atomic tasks and compositional tasks constructed from combinations of these atomic skills. We fine-tune a model embedding for each atomic task and store the resulting embeddings in the model embedding memory. At test time, a lightweight router uses features from the frozen policy backbone to score candidate model embeddings, and retrieves the highest-scoring embedding to combine with the pretrained model for task execution. The highest-scoring embedding is selected at each policy step. We illustrate the rollout in Figure 5. We also include the demo videos in the supplementary materials.

5 Related Work

Pretrained models and adaptation. Parameter-efficient fine-tuning (PEFT) keeps the backbone fixed while training a small number of additional parameters (Houlsby et al., 2019; Hu et al., 2022; Lester et al., 2021), supported by evidence that adaptation operates in a low intrinsic subspace (Aghajanyan et al., 2021; Frankle et al., 2020; Li et al., 2018; Wortsman et al., 2022). A complementary line exploits spectral structure: several methods apply SVD to pretrained matrices or task-induced deltas to tune singular directions, analyze interference, or guide merging (Gargiulo et al., 2025; Han et al., 2023; Zhang & Pilanci, 2024), while others study linear task composition or reduce interference via sparsification and adaptive coefficients (Ilharco et al., 2023; Yadav et al., 2023; Yang et al., 2024; Yu et al., 2024). In contrast, we use pretrained-induced spectral bases to define a shared coordinate system across adaptations, treating each task-specific update as an entry in a reusable model memory that supports comparison, retrieval, and reuse without modifying the pretrained backbone.

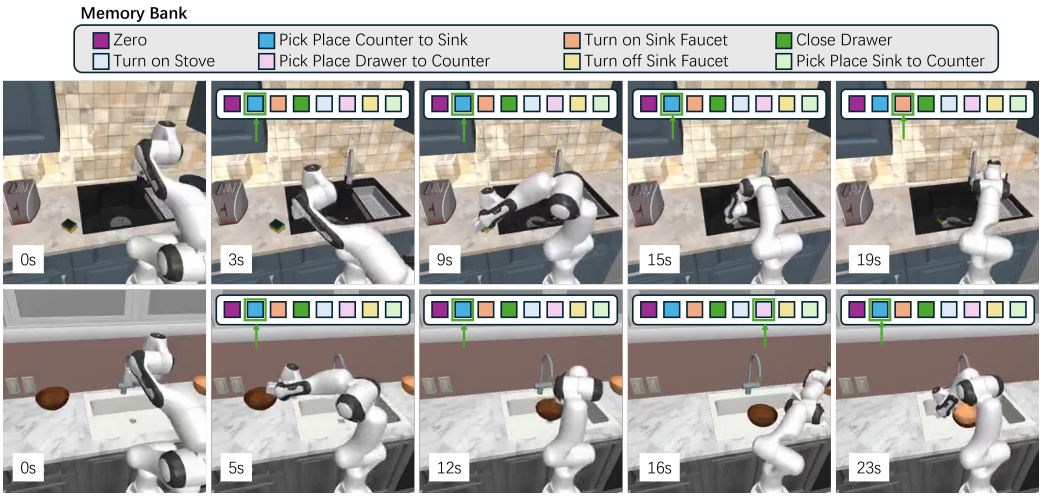


Figure 5: **Demonstration of model embedding memory retrieval.** Each model embedding is fine-tuned on an atomic task, such as "turn on the sink faucet," and the resulting embeddings are stored in a memory bank. The evaluation tasks are compositional; for example, a task requires moving an object from the counter to the sink and then turning on the faucet. At test time, the retrieval mechanism selects relevant model embeddings from memory to augment the pretrained model for the target task. We include demo videos in supplementary.

Policy learning for robot manipulation. A range of recent imitation learning methods learn robot manipulation policies from demonstrations, spanning diffusion-based (Chi et al., 2025), transformer-based (Bharadhwaj et al., 2024), and large vision-language-action architectures (Kim et al., 2024; Octo Model Team et al., 2024; Physical Intelligence et al., 2025). These models define the class of pretrained policy backbones targeted by our framework: each task adaptation is expressed as a fine-tuned residual over a shared backbone and stored as a model embedding. Rather than proposing a new policy architecture, our work complements these approaches by providing a structured memory over the task-specific adaptations they produce.

Memory and retrieval in robot learning. Observation-based methods retrieve demonstrations or sub-trajectories matched via visual features (Lin et al., 2024; Memmel et al., 2025) or object-centric geometric prompts (Zhu et al., 2023), while others index experience via language (Sridhar et al., 2026). Most closely related are model-based approaches that tokenize model parameters into vectors for efficient selection (Zhang et al., 2023) or retrieve and adapt specific skill modules and adapter libraries (Guo et al., 2025). However, these rarely provide a principled geometric coordinate system grounded in the pretrained model's weight structure. In contrast, our embedding space is constructed directly from pretrained-induced spectral structure, yielding task-commensurable embeddings that support efficient retrieval without a model zoo or retraining.

6 Conclusion

We present a framework for organizing task-specific adaptations as structured model memory. By grounding the embedding space in the intrinsic spectral structure of the pretrained weights, we obtain a shared coordinate system in which task adaptations are directly comparable by construction, without relying on semantic features or learned similarity metrics. Experiments across simulated robot manipulation tasks demonstrate that model embedding memory enables effective multi-task policy retrieval, and that language-guided retrieval is achievable by mapping LLM embeddings into the index space. More broadly, this work suggests that the weight structure of pretrained models is itself a basis for organizing and reusing learned task knowledge.

7 AI DISCLOSURE

In this work, we used generative AI tools for providing feedback on research methodology or experiments, and to assist in the writing of proofs. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, generate synthetic data sets, assist with translation, clean and reformat datasets, support qualitative and thematic data analysis, or interpret results are not applicable to this work. Additionally, we used generative AI tools for brainstorming, sourcing/searching for information, edit a research paper to improve readability. We have reviewed all AI-assisted work. We have checked the proof (assumptions, inequalities, etc.) used by AI. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL-IJCNLP*, 2021.

Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *ICRA*, 2024.

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*, 2025.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *CVPR*, 2025.

Yijie Guo, Bingjie Tang, Iretiayo Akinola, Dieter Fox, Abhishek Gupta, and Yashraj Narang. Srsa: Skill retrieval and adaptation for robotic assembly tasks. In *ICLR*, 2025.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Ligong Han, Yinxiao Li, Han Zhang, Peyman Milanfar, Dimitris Metaxas, and Feng Yang. Svdiff: Compact parameter space for diffusion fine-tuning. In *ICCV*, 2023.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

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 *ICLR*, 2022.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *ICLR*, 2023.

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. In *CoRL*, 2024.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *ICLR*, 2018.

Li-Heng Lin, Yuchen Cui, Amber Xie, Tianyu Hua, and Dorsa Sadigh. Flowretrieval: Flow-guided data retrieval for few-shot imitation learning. In *CoRL*, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *NeurIPS*, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *ICML*, 2024.

Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. Strap: Robot sub-trajectory retrieval for augmented policy learning. In *ICLR*, 2025.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *ICLR*, 2026.

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.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *CoRL*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. Scaling up memory for robotic control via experience retrieval. In *ICLR*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *NeurIPS*, 2020.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In *NeurIPS*, 2023.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *ICLR*, 2024.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*, 2024.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, 2019.

Fangzhao Zhang and Mert Pilanci. Spectral adapter: Fine-tuning in spectral space. *NeurIPS*, 2024.

Yi-Kai Zhang, Ting-Ji Huang, Yao-Xiang Ding, De-Chuan Zhan, and Han-Jia Ye. Model spider: Learning to rank pre-trained models efficiently. *NeurIPS*, 2023.

Yifeng Zhu, Zhenyu Jiang, Peter Stone, and Yuke Zhu. Learning generalizable manipulation policies with object-centric 3d representations. *CoRL*, 2023.

A A PEDAGOGICAL SCENARIO

In this section, we use the linear layer case as a toy scenario to discuss how different parts of model parameters contribute to different adaptation outcomes.

Pretrained model. Consider a linear generator with parameters $\mathbf{W} \in \mathbb{R}^{d\times r}$ and $\mathbf{b} \in \mathbb{R}^d$

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \mathbf{b}. \tag{19}$$

It maps the input $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_r)$ to the data distribution with $\mu_0 = \mathbf{b}$ and $\mathbf{\Sigma}_0 = \mathbf{W}\mathbf{W}^\top$,

$$p_0(\mathbf{x}) = \mathcal{N}(\mu_0, \mathbf{\Sigma}_0). \tag{20}$$

Downstream task. For downstream task i, we represent the target distribution with mean and covariance $(\mu_i, \mathbf{\Sigma}_i)$, *i.e.*,

$$p_i(x) = \mathcal{N}(\mu_i, \mathbf{\Sigma}_i). \tag{21}$$

Let the adapted model be

$$\mathbf{x} = (\mathbf{W} + \Delta\mathbf{W}_i)\mathbf{z} + (\mathbf{b} + \Delta\mathbf{b}_i). \tag{22}$$

Adaptation requires $\Delta\mathbf{b}_i \approx \mu_i - \mu_0$ and $(\mathbf{W} + \Delta\mathbf{W}_i)(\mathbf{W} + \Delta\mathbf{W}_i)^\top \approx \mathbf{\Sigma}_i$.

Adaptation capability. Let the covariance change of the task be

$$\Delta\mathbf{\Sigma}_i := \mathbf{\Sigma}_i - \mathbf{\Sigma}_0, \tag{23}$$

and let the adapted covariance change be $\Delta\hat{\mathbf{\Sigma}}_i := (\mathbf{W}+\Delta\mathbf{W}_i)(\mathbf{W}+\Delta\mathbf{W}_i)^\top - \mathbf{W}\mathbf{W}^\top$, expanded in (25) below.

The adaptation error is defined as $\mathcal{E}_i = \left\|\Delta\hat{\mathbf{\Sigma}}_i - \Delta\mathbf{\Sigma}_i\right\|_F^2$. The forgetting is measured via the KL divergence under pretrained inputs $\mathcal{F}_i = D_{KL}(p_0\|\hat{p}_i)$, where $\hat{p}_i$ is the distribution generated by the adapted model.

Discussion. We characterize the structure of the pretrained model through the singular value decomposition (SVD) of its linear mapping, $\mathbf{W} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{d\times r}$, $\mathbf{V} \in \mathbb{R}^{r\times r}$, and $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$ is diagonal with singular values $\lambda_1 \geq \cdots \geq \lambda_r > 0$. The pretrained covariance $\mathbf{\Sigma}_0 = \mathbf{W}\mathbf{W}^\top = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^\top$ shows that p_0 is supported on the subspace $\mathcal{S} = \mathrm{span}(\mathbf{U})$. Let $\mathbf{P} = \mathbf{U}\mathbf{U}^\top$ be the projector onto $\mathcal{S}$. Then $\Delta\mathbf{\Sigma}_i$ admits the decomposition

$$\Delta\mathbf{\Sigma}_i = \underbrace{\mathbf{P}\Delta\mathbf{\Sigma}_i\mathbf{P}}_{\text{in-subspace}} + \underbrace{(\mathbf{I}-\mathbf{P})\Delta\mathbf{\Sigma}_i(\mathbf{I}-\mathbf{P})}_{\text{out-of-subspace}} + \underbrace{\mathbf{P}\Delta\mathbf{\Sigma}_i(\mathbf{I}-\mathbf{P}) + (\mathbf{I}-\mathbf{P})\Delta\mathbf{\Sigma}_i\mathbf{P}}_{\text{cross-term}}. \tag{24}$$

Since $\{\mathbf{U}, \mathbf{U}_\perp\}$ is an orthonormal basis of $\mathbb{R}^d$, *any* update admits the corresponding coordinate decomposition $\Delta\mathbf{W}_i = \mathbf{U}\mathbf{M}_i\mathbf{V}^\top + \mathbf{U}_\perp\mathbf{N}_i\mathbf{V}^\top$, with $\mathbf{M}_i \in \mathbb{R}^{r\times r}$ and $\mathbf{N}_i \in \mathbb{R}^{(d-r)\times r}$ collecting the in- and out-of-subspace coordinates. Substituting gives

$$\Delta\hat{\mathbf{\Sigma}}_i = \Delta\mathbf{W}_i\mathbf{W}^\top + \mathbf{W}\Delta\mathbf{W}_i^\top + \Delta\mathbf{W}_i\Delta\mathbf{W}_i^\top \tag{25}$$

$$= \underbrace{\mathbf{U}(\mathbf{M}_i\mathbf{\Lambda} + \mathbf{\Lambda}\mathbf{M}_i^\top + \mathbf{M}_i\mathbf{M}_i^\top)\mathbf{U}^\top}_{\text{in-subspace}} + \underbrace{\mathbf{U}_\perp\mathbf{N}_i\mathbf{N}_i^\top\mathbf{U}_\perp^\top}_{\text{out-of-subspace}} + \text{cross-term}. \tag{26}$$

Comparing (24) and (25) block by block shows only that in-subspace targets are matched *exactly* by $\mathbf{M}_i^\star = (\mathbf{\Lambda}^2 + \mathbf{U}^\top\Delta\mathbf{\Sigma}_i\mathbf{U})^{1/2} - \mathbf{\Lambda}$ with $\mathbf{N}_i = 0$ (so $\mathcal{E}_i = 0$ with $r \times r$ parameters), whereas an out-of-subspace block forces $\mathbf{N}_i\mathbf{N}_i^\top = \mathbf{U}_\perp^\top\Delta\mathbf{\Sigma}_i\mathbf{U}_\perp \neq 0$. Being covariance identities, (24)–(25) cannot bound $\mathcal{F}_i$; we establish forgetting separately.

A.1 ANALYSIS

Intuition. The directions in $\mathbf{U}$ encode the output structure that supports the pretrained task. Therefore, an adaptation that remains within $\mathrm{span}(\mathbf{U})$ reuses the representation established during pretraining and avoids introducing unnecessary changes along previously inactive directions. In contrast, updates outside $\mathrm{span}(\mathbf{U})$ introduce additional distributional drift and may lead to forgetting of pretrained knowledge.

Theoretical analysis. We formalize this intuition using the forgetting measure

$$\mathcal{F}_i = D_{\mathrm{KL}}(p_0 \,\|\, p_i). \tag{27}$$

For the Gaussian distributions, considering only the covariance change, we have

$$\mathcal{F}_i = \frac{1}{2}\left[\mathrm{tr}\left((\boldsymbol{\Sigma}_0 + \Delta\boldsymbol{\Sigma}_i)^{-1}\boldsymbol{\Sigma}_0\right) - d + \log\frac{\det(\boldsymbol{\Sigma}_0 + \Delta\boldsymbol{\Sigma}_i)}{\det(\boldsymbol{\Sigma}_0)}\right]. \tag{28}$$

For a sufficiently small covariance perturbation, the Gaussian KL divergence admits the second-order approximation

$$\mathcal{F}_i \approx \frac{1}{4}\left\|\boldsymbol{\Sigma}_0^{-1/2}\Delta\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_0^{-1/2}\right\|_F^2 + O\left(\left\|\boldsymbol{\Sigma}_0^{-1/2}\Delta\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_0^{-1/2}\right\|_F^3\right). \tag{29}$$

Proposition A.1 (In-subspace target changes minimize unnecessary forgetting)**.** *For a sufficiently small covariance perturbation, the local KL divergence $\mathcal{F}_i = D_{\mathrm{KL}}(p_0 \,\|\, p_i)$ is minimized, for a fixed in-subspace covariance component, when the target covariance change lies entirely within the pretrained subspace, i.e.,*

$$\Delta\boldsymbol{\Sigma}_i = \mathbf{P}\Delta\boldsymbol{\Sigma}_i\mathbf{P}, \tag{30}$$

where $\mathbf{P} = \mathbf{U}\mathbf{U}^\top$ denotes the orthogonal projector onto $\mathrm{span}(\mathbf{U})$ and $\Delta\boldsymbol{\Sigma}_i$ is the covariance change required by the target task.

Proof. To ensure that the Gaussian distribution is nondegenerate and that the standard KL divergence is well defined, we introduce a small isotropic variance $\epsilon\mathbf{I}$, with $\epsilon > 0$. The pretrained covariance is therefore

$$\boldsymbol{\Sigma}_0 = \mathbf{W}\mathbf{W}^\top + \epsilon\mathbf{I}. \tag{31}$$

Let

$$\mathbf{S} = \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top + \epsilon\mathbf{I}_r. \tag{32}$$

Using $\mathbf{W} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^\top$, we obtain

$$\boldsymbol{\Sigma}_0 = \mathbf{U}\mathbf{S}\mathbf{U}^\top + \epsilon\mathbf{U}_\perp\mathbf{U}_\perp^\top. \tag{33}$$

Hence,

$$\begin{aligned}\boldsymbol{\Sigma}_0^{-1/2}\Delta\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_0^{-1/2} = {}& \mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top\Delta\boldsymbol{\Sigma}_i\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top \\ &+ \epsilon^{-1}(\mathbf{I}-\mathbf{P})\Delta\boldsymbol{\Sigma}_i(\mathbf{I}-\mathbf{P}) \\ &+ \epsilon^{-1/2}\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top\Delta\boldsymbol{\Sigma}_i(\mathbf{I}-\mathbf{P}) \\ &+ \epsilon^{-1/2}(\mathbf{I}-\mathbf{P})\Delta\boldsymbol{\Sigma}_i\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top.\end{aligned} \tag{34}$$

Because these four components occupy mutually orthogonal block subspaces, their squared Frobenius norms add. Therefore, up to higher-order terms,

$$\begin{aligned}\mathcal{F}_i \approx {}& \frac{1}{4}\left\|\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top\Delta\boldsymbol{\Sigma}_i\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top\right\|_F^2 \\ &+ \frac{1}{4\epsilon^2}\left\|(\mathbf{I}-\mathbf{P})\Delta\boldsymbol{\Sigma}_i(\mathbf{I}-\mathbf{P})\right\|_F^2 \\ &+ \frac{1}{2\epsilon}\left\|\mathbf{U}\mathbf{S}^{-1/2}\mathbf{U}^\top\Delta\boldsymbol{\Sigma}_i(\mathbf{I}-\mathbf{P})\right\|_F^2.\end{aligned} \tag{35}$$

The last two terms are nonnegative. Thus, for a fixed in-subspace component $\mathbf{P}\Delta\boldsymbol{\Sigma}_i\mathbf{P}$, the KL divergence is minimized by setting both the out-of-subspace and cross-subspace components to zero, yielding

$$\Delta\boldsymbol{\Sigma}_i = \mathbf{P}\Delta\boldsymbol{\Sigma}_i\mathbf{P}. \tag{36}$$

□

Remark A.2. *The out-of-subspace and cross-subspace contributions to the local KL divergence are scaled by $1/\epsilon^2$ and $1/\epsilon$, respectively. Since ϵ is small, covariance changes along directions in which the pretrained model has negligible variance incur a disproportionately large KL penalty.*

Remark A.3. *Our work focuses on the regime in which the covariance shift required by the target task lies primarily within* $\mathrm{span}(\mathbf{U})$*. Under this assumption, restricting adaptation to the pretrained subspace avoids introducing additional sources of distributional drift through out-of-subspace components.*

Proposition A.4 (In-subspace parameterization realizes an in-subspace target without unnecessary covariance drift)**.** *Suppose the target covariance change satisfies*

$$\Delta\mathbf{\Sigma}_i = \mathbf{P}\Delta\mathbf{\Sigma}_i\mathbf{P}. \tag{37}$$

Consider a parameter update of the form

$$\Delta\mathbf{W}_i = \mathbf{U}A\mathbf{V}^\top + \mathbf{U}_\perp B\mathbf{V}^\top. \tag{38}$$

If the adapted model exactly realizes the target covariance change,

$$\Delta\hat{\mathbf{\Sigma}}_i = \Delta\mathbf{\Sigma}_i, \tag{39}$$

then $B = \mathbf{0}$*. Consequently, the update must lie entirely within the pretrained output subspace:*

$$\Delta\mathbf{W}_i = \mathbf{U}A\mathbf{V}^\top. \tag{40}$$

Proof. The covariance change induced by the parameter update is

$$\begin{aligned}\Delta\hat{\mathbf{\Sigma}}_i = \mathbf{U}\left(A\mathbf{\Lambda}^\top + \mathbf{\Lambda}A^\top + AA^\top\right)\mathbf{U}^\top \\ + \mathbf{U}_\perp BB^\top\mathbf{U}_\perp^\top \\ + \mathbf{U}_\perp B(\mathbf{\Lambda}+A)^\top\mathbf{U}^\top \\ + \mathbf{U}(\mathbf{\Lambda}+A)B^\top\mathbf{U}_\perp^\top.\end{aligned} \tag{41}$$

Since the target covariance change satisfies

$$\Delta\mathbf{\Sigma}_i = \mathbf{P}\Delta\mathbf{\Sigma}_i\mathbf{P}, \tag{42}$$

its out-of-subspace block is zero. Exact covariance matching therefore requires

$$\mathbf{U}_\perp^\top\Delta\hat{\mathbf{\Sigma}}_i\mathbf{U}_\perp = BB^\top = \mathbf{0}. \tag{43}$$

Since $BB^\top = \mathbf{0}$ implies $B = \mathbf{0}$, we obtain

$$\Delta\mathbf{W}_i = \mathbf{U}A\mathbf{V}^\top. \tag{44}$$

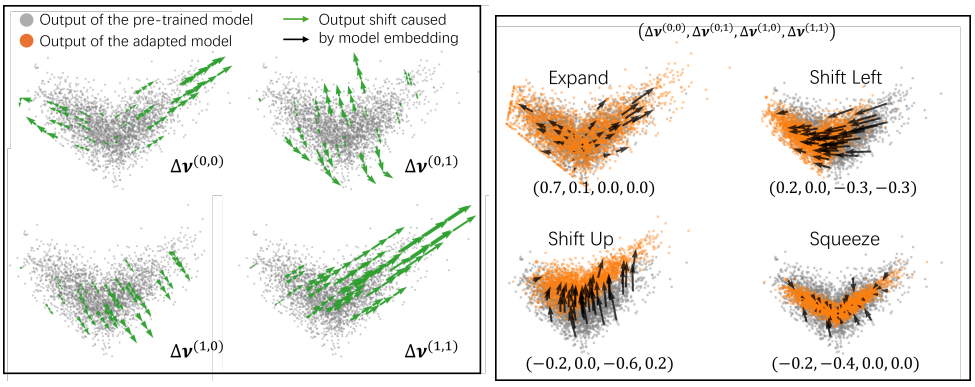
Thus, an in-subspace target covariance change can be realized without introducing out-of-subspace or cross-subspace covariance drift. □

Remark A.5. *Under our assumption that the target task lies primarily within* $\mathrm{span}(\mathbf{U})$*, the desired covariance adaptation can be realized through an in-subspace parameter update. Combined with Proposition R1, this shows that such a parameterization realizes the target while avoiding unnecessary sources of forgetting.*

A.2 Demonstration of Model Embedding Space Structure

To validate the structure of the model embedding space, we design a simple synthetic density estimation task using a two-layer ReLU MLP that maps a latent Gaussian $z \sim \mathcal{N}(0, I)$ to a 2D banana-shaped distribution. We extract a low-dimensional model embedding $\mathbf{M} \in \mathbb{R}^4$ from selected components across layers and study how perturbations in $\mathbf{M}$ affect the generated distribution.

As shown in Figure 6, we evaluate this by visualizing the induced displacement field between outputs before and after applying an embedding, and by systematically exploring the embedding space. Each embedding is associated with a qualitative transformation (e.g., shift, expansion, squeezing) based on the resulting distribution change. This experiment demonstrates that the model embedding space is geometrically structured: nearby embeddings induce similar transformations, while different regions correspond to distinct, interpretable behaviors.



(a) (b)

Figure 6: **Functional relationship between model embeddings and task space.** (a) An example of model embedding in the linear layer. Given an input $\mathbf{X}$, the output $\mathbf{O}$ is changed according to the model embedding $\mathbf{M}$ change. (b) Visualizing the influence of individual embedding dimensions on the pre-trained model (gray). For each layer, the impact of different model embeddings, e.g., $\mathbf{M}^{(0,0)}$ and $\mathbf{M}^{(0,1)}$, provides orthogonal change in output space, due to the structure enforced by $\mathbf{U}$ and $\mathbf{V}$. (c) Different task identities emerge from the combined influence of specific model embedding vectors.

A.3 Model Embedding Space and Task Space

Task space. We consider a family of robotic tasks indexed by $\tau \in \mathcal{T}$. Each task is characterized by task-dependent dynamics P_τ and objective R_τ,

$$\tau = (P_\tau, R_\tau), \tag{45}$$

where P_τ specifies how the robot and environment evolve under actions, and R_τ specifies the desired behavior. The task space $\mathcal{T}$ therefore represents variations in what the robot should accomplish and under what execution conditions the behavior is performed.

Let π_i and π_j denote the policies adapted to tasks τ_i and τ_j, respectively. We characterize the distance between two tasks through their induced behaviors,

$$d_{\mathcal{T}}(\tau_i, \tau_j) = \left(\mathbb{E}_{s\sim\mu}\left[\|\pi_i(s) - \pi_j(s)\|_2^2\right]\right)^{1/2}, \tag{46}$$

where μ is a shared state distribution over which the two task-specific policies are compared.

Task adaptation in the model embedding space. Let $\mathbf{W}_0$ denote the pretrained model parameters. For task τ_i, adaptation produces

$$\mathbf{W}_i = \mathbf{W}_0 + \Delta\mathbf{W}_i. \tag{47}$$

Given the singular-vector bases $\mathbf{U}$ and $\mathbf{V}$ obtained from the pretrained parameters, we represent the task-specific update as

$$\Delta\mathbf{W}_i = \mathbf{U}\mathbf{M}_i\mathbf{V}^\top, \tag{48}$$

where $\mathbf{M}_i$ is the model embedding of task τ_i. The bases $\mathbf{U}$ and $\mathbf{V}$ are fixed across tasks, providing a shared coordinate system inherited from the pretrained model.

For two tasks τ_i and τ_j, their parameter difference is

$$\mathbf{W}_i - \mathbf{W}_j = \Delta\mathbf{W}_i - \Delta\mathbf{W}_j \tag{49}$$

$$= \mathbf{U}(\mathbf{M}_i - \mathbf{M}_j)\mathbf{V}^\top. \tag{50}$$

Task-space distance is bounded by model-embedding distance. We assume that the policy is locally smooth with respect to the model parameters.

Assumption 2.1 (Local parameter smoothness). For all parameters in a neighborhood of the pretrained model containing the considered task adaptations, the policy satisfies

$$\left\|\pi_{\mathbf{W}_i}(s) - \pi_{\mathbf{W}_j}(s)\right\|_2 \leq L(s) \left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F, \tag{51}$$

for some state-dependent constant $L(s)$. Let

$$C = \left(\mathbb{E}_{s\sim\mu}[L(s)^2]\right)^{1/2}. \tag{52}$$

Proposition 2.2 Under Assumption 2.1, the behavioral distance between two tasks is bounded by the distance between their model embeddings:

$$d_{\mathcal{T}}(\tau_i, \tau_j) \leq C \left\|\mathbf{M}_i - \mathbf{M}_j\right\|_F. \tag{53}$$

Proof. From (51),

$$\left\|\pi_{\mathbf{W}_i}(s) - \pi_{\mathbf{W}_j}(s)\right\|_2^2 \leq L(s)^2 \left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F^2. \tag{54}$$

Taking expectation over the shared state distribution μ gives

$$d_{\mathcal{T}}(\tau_i, \tau_j)^2 \leq \mathbb{E}_{s\sim\mu}\left[L(s)^2\right] \left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F^2 \tag{55}$$

$$= C^2 \left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F^2. \tag{56}$$

Therefore,

$$d_{\mathcal{T}}(\tau_i, \tau_j) \leq C \left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F. \tag{57}$$

Using the model embedding parameterization,

$$\left\|\mathbf{W}_i - \mathbf{W}_j\right\|_F = \left\|\mathbf{U}(\mathbf{M}_i - \mathbf{M}_j)\mathbf{V}^\top\right\|_F. \tag{58}$$

Since $\mathbf{U}$ and $\mathbf{V}$ consist of orthonormal singular vectors,

$$\mathbf{U}^\top\mathbf{U} = \mathbf{I}, \qquad \mathbf{V}^\top\mathbf{V} = \mathbf{I}, \tag{59}$$

and the Frobenius norm is invariant under multiplication by orthonormal matrices. Hence,

$$\left\|\mathbf{U}(\mathbf{M}_i - \mathbf{M}_j)\mathbf{V}^\top\right\|_F = \left\|\mathbf{M}_i - \mathbf{M}_j\right\|_F. \tag{60}$$

Substituting (60) into (57) yields

$$d_{\mathcal{T}}(\tau_i, \tau_j) \leq C \left\|\mathbf{M}_i - \mathbf{M}_j\right\|_F. \tag{61}$$

□

Interpretation. The result shows that the model embedding preserves the distance of task-specific parameter updates within the pretrained singular-vector subspace,

$$\left\|\Delta\mathbf{W}_i - \Delta\mathbf{W}_j\right\|_F = \left\|\mathbf{M}_i - \mathbf{M}_j\right\|_F. \tag{62}$$

Combined with the local smoothness of the policy, this yields

$$d_{\mathcal{T}}(\tau_i, \tau_j) \leq C \left\|\mathbf{M}_i - \mathbf{M}_j\right\|_F. \tag{63}$$

Therefore, two task adaptations that are close in model embedding space cannot induce arbitrarily different behaviors. This provides a direct connection between the geometry of the model embedding space and the local geometry of the induced task behaviors, supporting the use of model embeddings for organizing and retrieving task-specific knowledge.

B Additional Discussion About Model Embedding Space

B.1 Model Embedding in Attention

The multi-head attention layer Vaswani et al. (2017) transforms the input sequence $\mathbf{X} \in \mathbb{R}^{N\times d}$ to the output sequence $\mathbf{O} \in \mathbb{R}^{N\times d}$, where N denotes the sequence length, d is the dimension of input and output features. It allows multiple attention mechanisms to work in parallel, with each head independently learning attention patterns.

For H attention heads, each attention head calculates attention maps as $\mathbf{A}^{(h)} = \texttt{softmax}(\mathbf{X}\,\mathbf{W}_q^{(h)}\mathbf{W}_k^{(h)\top}\mathbf{X}^\top)$, where projection matrices $\mathbf{W}_q^{(h)}, \mathbf{W}_k^{(h)} \in \mathbb{R}^{d\times d_h}$, $h = 1, \cdots, H$, $d_h = \frac{d}{H}$. The multi-head attention represents as,

$$\texttt{MHA}(\mathbf{X}) = \sum_{h=1}^{H} \mathbf{A}^{(h)}\mathbf{X}\mathbf{W}_v^{(h)}\mathbf{W}_o^{(h)\top} \tag{64}$$

where projection matrices $\mathbf{W}_v^{(h)}, \mathbf{W}_o^{(h)} \in \mathbb{R}^{d\times d_h}$.

Attention and per-head outputs:

$$\mathbf{Z}^{(h)} = \mathbf{A}^{(h)}\mathbf{X}\mathbf{W}_v^{(h)} \in \mathbb{R}^{N\times d_h}. \tag{65}$$

Each attention head applies a projection from d to d_h.

Concatenate heads and apply the fixed output projection:

$$\mathbf{O} = \sum_{h=1}^{H} \mathbf{Z}^{(h)}\mathbf{W}_o^{(h)}, \tag{66}$$

where $\mathbf{W}_o^{(h)} \in \mathbb{R}^{d_h\times d}$ maps $\mathbf{Z}^{(h)}$ back to a higher-dimensional representation.

Given *fixed* pre-trained weights $\mathbf{W}_q^{(h)}$, $\mathbf{W}_k^{(h)}$, $\mathbf{W}_v^{(h)}$, $\mathbf{W}_o^{(h)}$, for each head h, we insert a learnable model embedding $\mathbf{M}^{(h)} \in \mathbb{R}^{d_h\times d_h}$ to adapt to the downstream task:

$$\begin{aligned} \mathbf{Z}'^{(h)} &= \mathbf{Z}^{(h)} + \mathbf{Z}^{(h)}\,\mathbf{M}^{(h)}, \\ \widetilde{\mathbf{O}}^{(h)} &= \mathbf{Z}'^{(h)}\,\mathbf{W}_o^{(h)\top} \in \mathbb{R}^{n\times d}. \end{aligned} \tag{67}$$

Initialize each model embedding to zero: $\mathbf{M}^{(h)} = \mathbf{0}_{d_h}\,\forall h$, and it causes *zero* deviation from the pre-trained model at initialization.

B.2 Toy Reacher with Compositional Dynamics Variations

Environment. We consider a planar two-link reaching problem with two actuated joints. The arm has link lengths 0.10 and 0.11, and each episode lasts 100 control steps with a simulation time step of 0.05. The observation is a 10-dimensional vector

$$s = [\cos(q_1), \cos(q_2), \sin(q_1), \sin(q_2), x_{\text{target}}, y_{\text{target}}, \dot{q}_1, \dot{q}_2, x_{\text{tip}} - x_{\text{target}}, y_{\text{tip}} - y_{\text{target}}]\,. \tag{68}$$

Importantly, neither the task identity nor the actuator gains are included in the observation.

The policy produces a two-dimensional bounded command $\mathbf{a}_t \in [-1, 1]^2$. Each task applies a diagonal transformation before the command is passed to the arm:

$$\widetilde{\mathbf{a}}_t = \begin{bmatrix} g_1 & 0 \\ 0 & g_2 \end{bmatrix} \mathbf{a}_t, \tag{69}$$

where $\mathbf{a}_t$ is the commanded action and $\widetilde{\mathbf{a}}_t$ is the applied action. The approximate joint dynamics are

$$\ddot{\mathbf{q}}_t = 8\widetilde{\mathbf{a}}_t - 1.2\dot{\mathbf{q}}_t + \mathbf{c}(\mathbf{q}_t, \dot{\mathbf{q}}_t), \tag{70}$$

where $\mathbf{c}$ is a weak velocity-dependent coupling term. The reward is shared by all tasks:

$$r_t = -\left\|\mathbf{x}_{\text{tip},t} - \mathbf{x}_{\text{target}}\right\|_2 - 0.01\left\|\widetilde{\mathbf{a}}_t\right\|_2^2. \tag{71}$$

A state is considered successful when the fingertip is within 0.025 of the target.

Table 5: The four reacher dynamics tasks. A negative gain reverses the effect of the corresponding policy command.

Task	g_1	g_2	Dynamics interpretation
0	$+1$	$+1$	nominal dynamics
1	-1	$+1$	joint 1 is reversed
2	$+1$	-1	joint 2 is reversed
3	-1	-1	both joints are reversed

Table 6: Performance of the task-specific model embedding adapters. Higher return and success rates are better.

Task	Mean return	Success	Final success
0	-5.90	0.35	0.30
1	-8.19	0.50	0.35
2	-8.27	0.15	0.05
3	-14.76	0.05	0.00
Mean	-9.28	0.26	0.18

Dynamics tasks. The four tasks form a complete 2×2 factorial design over the signs of the two actuator gains in Table 5. Task 0 has the same actuator dynamics as the pretraining task. Tasks 1–3 systematically reverse one or both control channels. Thus, the four tasks isolate the effect of joint 1, joint 2, and their interaction while keeping the observation space, target distribution, reward, and underlying arm identical.

Pretraining and task adaptation. We train the base policy using Soft Actor–Critic (SAC) on the nominal Task 0 dynamics. The base actor contains two hidden layers of width 128, followed by separate two-dimensional mean and log-standard-deviation heads. The base training uses $100{,}000$ environment steps, and the checkpoint selected at step $90{,}000$ is reused for all subsequent task adaptations. For each dynamics task, the pretrained actor is frozen and only a model embedding adapter in the second hidden layer is optimized. The input layer and both output heads remain unchanged.

Control performance. Table 6 reports deterministic evaluation over 20 episodes for the run with training seed 0. The "success" metric denotes the fraction of episodes that enter the success region at least once, while "final success" requires the arm to remain in the success region at the last step.

The experiment demonstrates successful compensation in several evaluation episodes, but performance is not uniform across the four tasks. In particular, reversing both joints is substantially harder and Task 3 remains poorly solved under the current training budget. Therefore, these results should be interpreted primarily as evidence that the learned adapter space tracks structured dynamics changes, rather than as evidence that all tasks have converged. Reloading an adapter reproduces its original mean return exactly, with a measured retention ratio of 1.0, because adapters are stored independently and recalled without further optimization.

Adapter embeddings. To study whether physical differences between tasks induce corresponding differences in parameter space, we flatten the 128 active model embeddings of each task adapter and treat the resulting vector as a task embedding. A two-dimensional PCA projection explains 42.3% and 34.7% of the variance in its first two components, respectively, for a total of 77.0%. Moreover, the Pearson correlation between pairwise task distance in gain space and pairwise Euclidean distance in adapter space is 0.744.

B.3 Model Embedding Index Selection

To construct a compact retrieval index, we analyze how much of the full model-embedding similarity structure is preserved by smaller subsets of the model embedding. For each task adapter, the

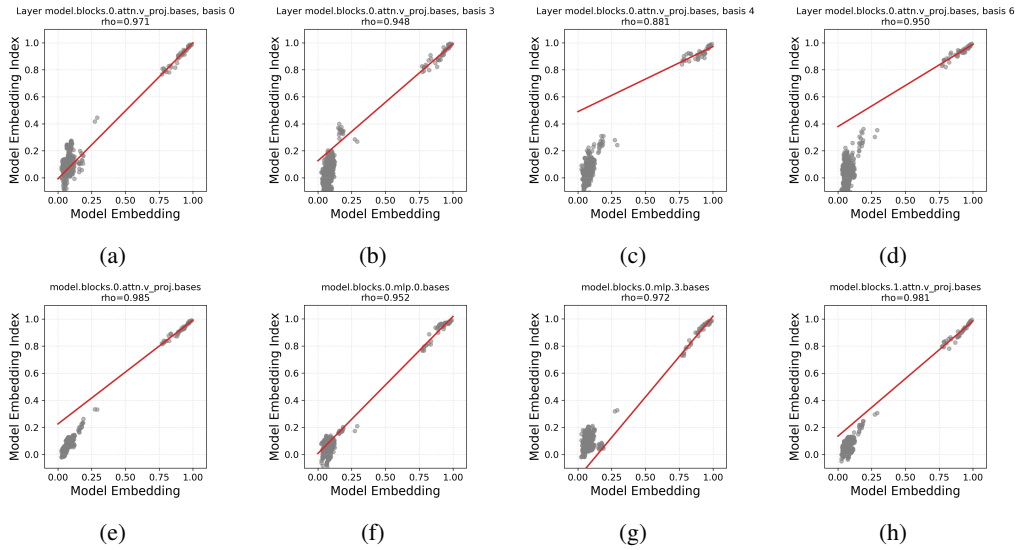


Figure 7: **Spearman correlation between index and full model embeddings.** Scatter plots compare cosine similarities of the index space (x-axis) to full embedding space (y-axis). (a–e) $\mathrm{v_proj}$ basis blocks and aggregate in block 0 ($\rho > 0.94$). (f–g) MLP layers in block 0. (h) $\mathrm{v_proj}$ in block 1. High ρ values confirm the index faithfully preserves full model retrieval rankings.

full model embedding is defined as the concatenation of all learned model embedding tensors across layers. We then evaluate three classes of candidate indices: the embedding from a single layer, an individual basis/head within a layer, and the same basis/head index concatenated across all layers. For each candidate index, we compute the pairwise cosine similarity matrix over adapters and compare it with the similarity matrix induced by the full model embedding using Pearson and Spearman correlation.

Figure 7 reveals that task-relevant similarity is nearly uniformly distributed across all embedding components. These results suggest that model embeddings admit an efficient retrieval index: a small, structured subset of embedding coordinates can approximate the full embedding's task similarity relationships, reducing memory and query cost without discarding the core retrieval signal.

C Experiment Details

C.1 Meta-World Dataset

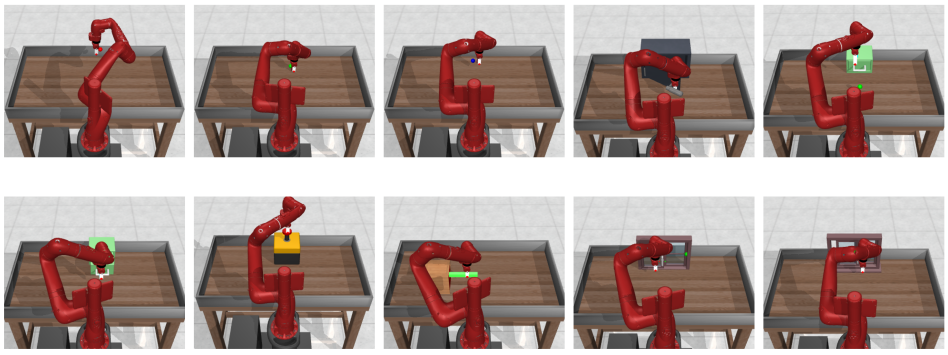


Figure 8: Representative tasks from the Meta-World MT10 benchmark. The grid displays rendered MuJoCo frames for the 10 manipulation tasks, demonstrating the diversity of environments from reaching to articulated-object manipulation.

Meta-World (Yu et al., 2019) is a suite of 10 tabletop manipulation tasks implemented in MuJoCo, designed to benchmark multi-task and meta-reinforcement learning. Each task defines a distinct manipulation behavior, including reaching, pushing, pick-and-place, drawer opening and closing, button pressing, peg insertion, and articulated-object manipulation, with randomized goal and object positions at each episode reset, as shown in Figure 8.

We use the MT50 multi-task split, which consists of 50 tasks. Observations are low-dimensional state vectors comprising end-effector position, gripper aperture, object positions, and goal positions; no pixel-based observations are used. For our experiments, we collect N demonstrations per task using a scripted expert policy and fine-tune a shared pretrained policy backbone on each task independently to obtain a task-specific actor. The resulting collection of 40 task-specific fine-tuned policies constitutes the model memory evaluated in our retrieval experiments. Evaluation follows the standard Meta-World protocol: a rollout is deemed successful if the task completion criterion is satisfied within a fixed episode horizon, and we report mean success rate averaged over all 40 tasks.

C.2 LIBERO Dataset

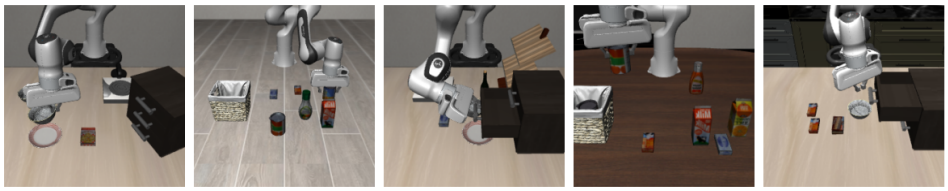


Figure 9: Task examples from the five LIBERO suites used in our experiments. From left to right: LIBERO-Spatial (spatial arrangement variation), LIBERO-Object (object type variation), LIBERO-Goal (goal predicate variation), LIBERO-90 (large-scale pre-training split), and LIBERO-10 (long-term reasoning tasks). Each task is paired with a natural-language instruction shown below the rendered frame.

LIBERO (Liu et al., 2023) (Figure 9) is a language-conditioned robot manipulation benchmark built on the Robosuite simulation framework. It comprises 130 tasks grouped into four task suites, each probing a distinct type of knowledge transfer:

- **LIBERO-Spatial** (10 tasks): object identities and scene layouts are fixed; spatial predicates and arrangements vary across tasks, probing transfer of relational knowledge.
- **LIBERO-Object** (10 tasks): spatial configurations are fixed; object categories and instances vary, probing transfer of object-centric knowledge.
- **LIBERO-Goal** (10 tasks): the scene and objects are held constant; goal predicates (e.g., *open*, *place inside*, *push*) vary, probing transfer of task-goal knowledge.
- **LIBERO-100** (100 tasks): The combination of LIBERO-90 and LIBERO-10, where 90 tasks provide a large-scale diverse distribution for multi-task pre-training, and the remaining 10 tasks serve as a distinct split to evaluate zero-shot or few-shot transfer and long-term reasoning capabilities.

Each task is paired with a natural-language instruction and 50 high-quality human-teleoperated demonstrations. Observations consist of RGB images from two fixed cameras (agentview and wrist), concatenated with robot proprioceptive state. We evaluate our framework on all four task suites, using the per-suite task collections as the model memory; the pretrained backbone is shared across all tasks within a suite. Success is measured as binary task completion averaged over 20 rollouts per task, following the standard LIBERO evaluation protocol.

C.3 MTACT

We adopt MTACT (Bharadhwaj et al., 2024) as the shared pretrained policy backbone for all LIBERO experiments. For our framework, the backbone is first pretrained on the full suite-level demonstration corpus, then frozen. Per-task fine-tuning optimizes only the model-embedding parameters.

C.4 Training Hyperparameters

Table 7: Training hyperparameters for backbone pre-training, per-task fine-tuning, and model embedding index training. Values marked with † were selected by grid search on a held-out validation split; all others are fixed across experiments.

Stage	Hyperparameter	Meta-World	LIBERO
Pre-training	Optimizer	AdamW	AdamW
	Learning rate	3×10^{-4}	1×10^{-4}
	LR schedule	Cosine decay	Cosine decay
	Batch size	512	64
	Training steps	10 (epochs)	100,000
Per-task fine-tuning	Optimizer	AdamW	AdamW
	Learning rate	5×10^{-5}	5×10^{-5}
	LR schedule	Cosine decay	Cosine decay
	Batch size	512	64
	Training steps	500 (epochs)	50,000
Model embedding index	Index dimension	128	512

All models are trained on a single NVIDIA RTX 5090 32 GB GPU. Gradient clipping with a maximum norm of 1.0 is applied in all stages.

Meta-World experiment. We construct the environment suite from Meta-World MT10 using scripted expert policies (50 episodes per task) for behavioral cloning. The policy backbone is a causal transformer with observation dimension 39, action dimension 4, context length 40, hidden size 128, 2 transformer blocks, 8 attention heads, MLP ratio 2, and dropout 0.1. Training hyperparameters follow Table 7. We compare LoRA (rank 16, $\alpha = 16$) against model-embedding adapters ($H = 8$), injected into all attention projections and MLP layers. Evaluation uses 25 closed-loop rollouts per task.

LIBERO experiment. We pretrain one MTACT backbone on the LIBERO-90 suite using the hyperparameters in Table 7, saving the checkpoint with the highest held-out success rate. Per-task adaptation freezes the backbone and fine-tunes only the model-embedding parameters ($H = 8$, all policy modules); the best checkpoint per subtask is selected by evaluation success. Evaluation reports mean binary success rate over 20 rollouts per task. To quantify inter-task interference, we evaluate the pretrained backbone alone and with a single task's embedding applied across all other tasks in the suite, using the per-task performance delta as a forgetting diagnostic.

C.5 Inference Details

Table 8: Inference-time parameters for task retrieval and policy execution. Retrieval latency is measured on an NVIDIA RTX 5090 GPU averaged over 100 query trials.

Parameter	Meta-World	LIBERO
Retrieval modality (default)	Model embedding index	Model embedding index
Similarity metric	Cosine	Cosine
Top-k candidates retrieved	1	1
LLM embedding projection	N/A	Linear
Evaluation rollouts per task	20	20
Episode horizon (steps)	500	600
Retrieval latency (ms, approx.)	< 1	< 1

Retrieval procedure. At inference, queries are formed either directly from a task's model embedding (weight-space retrieval) or via the learned projection ψ from a frozen LLM embedding

Table 9: Five held-out instruction variants for a single LIBERO-Object task. None of these instructions is used to train the language-to-adapter mapping.

Variant	Held-out instruction	Target token
Explicit paraphrase	Grab the ketchup bottle and drop it into the basket.	$\langle \mathrm{ADAPTER}_{O,04} \rangle$
Conversational shorthand	I want this robot arm to pick up the ketchup.	$\langle \mathrm{ADAPTER}_{O,04} \rangle$
Polite request	Could you please have the robot pick up the ketchup and place it in the basket?	$\langle \mathrm{ADAPTER}_{O,04} \rangle$
Goal-state description	The desired final arrangement is achieved after the robot can grab the ketchup and set it in the container.	$\langle \mathrm{ADAPTER}_{O,04} \rangle$
Synonym instruction	Robot, grab the ketchup and set it in the container.	$\langle \mathrm{ADAPTER}_{O,04} \rangle$

(language retrieval; see Section 3.2). In both cases the query index $\boldsymbol{\iota}_q$ is compared against all stored entries $\{\boldsymbol{\iota}_i\}$ in $\mathcal{M}$ using cosine similarity and the top-1 entry is selected. The full model embedding $\mathbf{M}_s$ is then reconstructed via the mapping $\boldsymbol{\iota}_s \rightarrow \mathbf{M}_s$ and injected into the frozen backbone. No gradient updates are performed at test time.

Policy execution. The retrieved actor operates in open-loop action-chunking mode: at each decision step the policy predicts a chunk of k future targets tracked by a low-level PD controller, with k given in Table 8.

Language-guided retrieval. Task descriptions are embedded with a frozen MiniLM encoder to produce k^{text}, then projected into the index space via $\tilde{\boldsymbol{\iota}} = \psi(k^{\text{text}}) \in \mathcal{I}$, where ψ is a linear map trained on paired $\{\text{description}, \text{index}\}$ examples from training tasks only. At test time, the weight delta is never observed; only $\tilde{\boldsymbol{\iota}}$ serves as the retrieval query.

C.6 Language-to-Index Mapping

Given 40 LIBERO tasks with task-specific model embedding adapters, we encode each canonical task description using a frozen language encoder and learn a lightweight mapping $\psi : \mathbb{R}^{384} \rightarrow \mathbb{R}^{16}$ from language space to an adapter-index embedding space, where each discrete token $\langle \mathrm{ADAPTER}_i \rangle$ addresses the corresponding adapter checkpoint. At test time ψ is frozen and only the projected embedding $\tilde{\boldsymbol{\iota}}$ is used as the retrieval query; the task-specific weight delta is never observed.

C.7 Held-Out Language Instructions

For each of the 40 LIBERO tasks, we construct five held-out instruction variants: an explicit paraphrase, a conversational shorthand request, a polite request, a goal-state description, and a synonym-based instruction. These instructions are not used to train the language-to-adapter mapping; training uses only the canonical task description.

Table 9 illustrates the five variants generated for the task *pick up the ketchup and place it in the basket*. Although the instructions differ in wording and may omit information implied by the task library, they are all associated with the same adapter token, $\langle \mathrm{ADAPTER}_{O,04} \rangle$.

Table 10 presents additional examples spanning LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10. The spatial examples test fine-grained relational distinctions, whereas the LIBERO-10 examples require the router to identify multi-stage or multi-object behaviors.

Given an instruction, the language model first predicts a discrete task token. The learned mapping ψ subsequently maps the selected language token into the model embedding index space,

$$\tilde{\boldsymbol{\iota}} = \psi(k^{\text{text}}).$$

The resulting index is used to load the corresponding model embedding checkpoint.

Table 10: Representative held-out instructions from the four LIBERO suites. The target token indexes the task-specific model embedding adapter used to execute the instruction.

Suite	Variant	Held-out instruction	Target token
Spatial	Explicit paraphrase	Move the black bowl located between the plate and ramekin onto the plate.	$\langle \mathrm{ADAPTER}_{S,00} \rangle$
Spatial	Conversational shorthand	Can you plate the bowl that is sitting by the ramekin?	$\langle \mathrm{ADAPTER}_{S,01} \rangle$
Spatial	Synonym instruction	Robot, grab the black bowl on the stove and set it on the plate.	$\langle \mathrm{ADAPTER}_{S,07} \rangle$
Object	Explicit paraphrase	Put the can of alphabet soup into the basket.	$\langle \mathrm{ADAPTER}_{O,00} \rangle$
Object	Conversational shorthand	Could the arm pick up the cream cheese for me?	$\langle \mathrm{ADAPTER}_{O,01} \rangle$
Object	Goal-state description	The desired final arrangement is achieved after the robot can move the orange juice into the container.	$\langle \mathrm{ADAPTER}_{O,09} \rangle$
Goal	Conversational shorthand	Open up the middle cabinet drawer.	$\langle \mathrm{ADAPTER}_{G,00} \rangle$
Goal	Explicit paraphrase	Open the upper drawer, then place the bowl inside it.	$\langle \mathrm{ADAPTER}_{G,03} \rangle$
Goal	Conversational shorthand	Fire up the stove.	$\langle \mathrm{ADAPTER}_{G,07} \rangle$
Goal	Synonym instruction	Robot, set the wine bottle on the rack.	$\langle \mathrm{ADAPTER}_{G,09} \rangle$
LIBERO-10	Explicit paraphrase	Turn on the stove and then place the moka pot on the active burner.	$\langle \mathrm{ADAPTER}_{L10,00} \rangle$
LIBERO-10	Conversational shorthand	Bowl into the lower drawer, then close it.	$\langle \mathrm{ADAPTER}_{L10,01} \rangle$
LIBERO-10	Conversational shorthand	Basket both the alphabet soup and the tomato sauce.	$\langle \mathrm{ADAPTER}_{L10,04} \rangle$
LIBERO-10	Explicit paraphrase	Put the white mug on the left plate and the yellow-and-white mug on the right one.	$\langle \mathrm{ADAPTER}_{L10,06} \rangle$
LIBERO-10	Conversational shorthand	File the book in the caddy's back slot.	$\langle \mathrm{ADAPTER}_{L10,09} \rangle$

D Limitations and Future Work

The framework has several limitations worth noting. First, embedding quality is bounded by the pre-trained backbone, *i.e.*, if the backbone has not been exposed to relevant task distributions, the induced spectral bases may not align with task-relevant adaptation directions. Second, our method assumes downstream adaptation is concentrated within the pretrained subspace; tasks requiring substantial out-of-subspace adaptation will incur higher representational bias. Third, the language projection ψ requires paired (description, index) supervision from training tasks, so generalization to out-of-distribution task descriptions depends on the capability of LLMs. Finally, all experiments are conducted in simulation; validating on a physical robot platform and studying how real-world distribution shift affects the embedding geometry are important directions for future work.