SARAM: Sparse Adapter Allocation with Reuse and Merging for Lifelong Robot Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lifelong robot learning requires a robot to acquire new tasks sequentially without forgetting previously learned ones. Adapter-based methods have recently addressed this problem effectively. They train task-specific adapters and freeze them, so learning a new task cannot overwrite past parameters. A selection module then infers which task's adapters to activate at deployment. However, their cost grows with the task stream. Per-task adapters and selection modules accumulate on disk, stay resident beside the backbone at deployment, and raise GPU memory with the number of learned tasks against the fixed budget of onboard hardware. To address this, we propose SARAM, an adapter-based continual learning method that maintains performance while keeping storage growth and deployment cost low. Specifically, each new task first fits combination coefficients over the frozen adapters of past tasks. A sparse allocation then adds new adapters at only a few layers, and a duplication penalty discourages them from re-expressing what reuse already provides. After each task is learned, its training demonstrations are passed once through the frozen backbone to build a compact task signature from activation statistics. At deployment, this signature selects the task once per episode, and the selected update is merged into the backbone weights before execution. On four LIBERO suites, SARAM matches or exceeds baselines while adding only a small amount of state per task, and keeps GPU memory and control frequency at the backbone's level regardless of task count.

1 Introduction

Robots deployed in real environments need to learn new tasks continually under the fixed resource budgets of onboard hardware. Replay methods Chaudhry et al. [2019] mitigate forgetting by retaining past demonstrations, but their storage grows with the task stream and raises privacy concerns. Regularization methods Kirkpatrick et al. [2017] avoid storing data but provide limited protection against forgetting. Adapter-based approaches Liu et al. [2024], Lei et al. [2025], Römer et al. [2026] instead freeze task-specific parameters after each task, so a new task cannot overwrite previous ones. Freezing, however, shifts the difficulty from parameter interference to resource accumulation. Since the oracle task ID is unavailable at deployment, these methods attach per-task selection modules to infer which adapters to activate Römer et al. [2026], Lei et al. [2025], and both adapters and selection modules accumulate as the task stream grows.

We identify two resource axes along which this accumulation grows. The first is disk storage: each task leaves its adapters, selection modules, and associated state on disk, so persistent storage grows with the task stream. The second is GPU memory and computation at deployment: selection runs alongside the backbone and all candidate adapters, so the resident model grows as tasks are learned, and GPU memory and per-step inference cost increase against the fixed budgets of onboard hardware.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

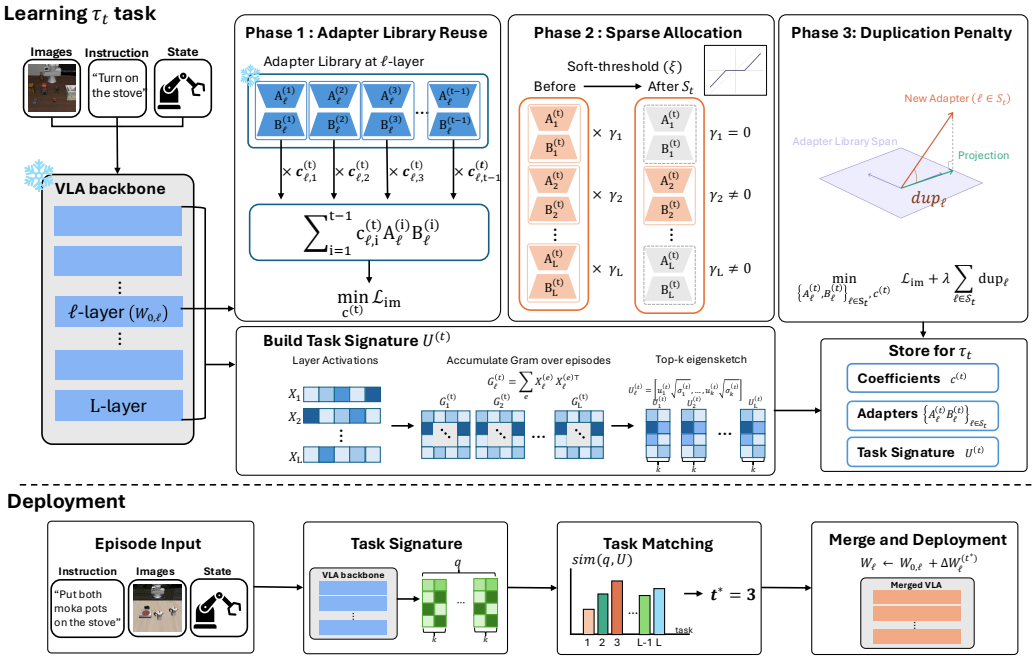


Figure 1: Overview of SARAM.

This raises our central question: *Can a continual learning method limit its storage growth, and keep deployment memory and control cost independent of the task count?*

To address this, we propose SARAM, an adapter-based continual learning method that reuses past adapters to add new parameters sparingly, and merges each task's update into the backbone weights. A new task first learns a linear combination of the frozen LoRA adapters Hu et al. [2021] of past tasks, reusing behavior already acquired. A sparse adapter allocation then adds new adapters only at the layers where this reuse falls short, and a duplication penalty keeps them from reproducing what the reused adapters already express. Both the new adapters and the combination coefficients are frozen afterwards, so previously learned task parameters cannot be overwritten. For deployment, no selection module is learned. After each task is trained, its demonstrations are passed once through the frozen backbone, and second-order activation statistics are summarized into a compact task signature; at the start of an episode, the first observation and instruction are summarized in the same way, and the most similar signature selects the task. Because the update is linear, it is merged into the backbone weights before acting, and the policy executes as a single backbone whose memory and per-step cost remain independent of the number of learned tasks.

We evaluate SARAM on four LIBERO suites against replay, regularization, and modular baselines, under the benchmark's continual learning metrics. SARAM attains the best or comparable performance among the compared methods while adding the fewest parameters and the least on-disk state per task among adapter-based methods. At deployment, measured GPU memory and control frequency stay at the backbone's own level regardless of how many tasks have been learned, where prior adapter-based methods grow and slow down.

2 Method

2.1 Problem Setting

We follow the lifelong imitation learning setup of LIBERO Liu et al. [2023]. A robot learns T manipulation tasks $\{\tau_1, \ldots, \tau_T\}$ sequentially with a single policy π conditioned on the task. Each task τ_t comes with a language instruction ι_t and a demonstration dataset $\mathcal{D}_t = \{(o_i, \iota_t, a_i)\}_{i=1}^N$ of observation–action pairs. Past datasets are no longer fully available once the stream moves on. The

policy is evaluated throughout the stream, on the task being learned and on all earlier tasks, so a method must balance acquiring new tasks against retaining old ones.

2.2 Per-Task Updates as Linear Deltas

SARAM adapts a pre-trained VLA backbone with frozen weights $W_0 = \{W_{0,\ell}\}_{\ell=1}^{L}$, where ℓ indexes the linear layers subject to adaptation. After learning task τ_j, its new adapters are kept as frozen rank-r LoRA Hu et al. [2021] adapters, with $A_\ell^{(j)} \in \mathbb{R}^{d_\ell \times r}$ and $B_\ell^{(j)} \in \mathbb{R}^{r \times d'_\ell}$, where d_ℓ and d'_ℓ are the input and output widths of layer ℓ. We refer to the adapters kept by $\tau_1, \ldots, \tau_{t-1}$ as the library available to task τ_t. The update of task τ_t at layer ℓ combines this library with at most one new adapter,

$$\Delta W_\ell^{(t)} \;=\; \sum_{j<t} c_{\ell,j}^{(t)}\, A_\ell^{(j)} B_\ell^{(j)} \;+\; A_\ell^{(t)} B_\ell^{(t)}, \tag{1}$$

where $c_{\ell j}^{(t)} \in \mathbb{R}$ is a combination coefficient on the j-th library adapter at layer ℓ. The per-task parameters of τ_t are thus $\theta_t = \{c^{(t)}, (A_\ell^{(t)}, B_\ell^{(t)})_{\ell \in \mathcal{S}_t}\}$, where $\mathcal{S}_t$ is the set of layers that receive a new adapter. The expression is linear in the weights and contains no nonlinearity, so $\Delta W_\ell^{(t)}$ can be added to $W_{0,\ell}$ rather than evaluated as a separate module. Learning τ_t fills in Eq. (1) in three phases: Phase 1 fits the combination coefficients $c^{(t)}$, Phase 2 determines $\mathcal{S}_t$, and Phase 3 trains the new adapters jointly with $c^{(t)}$.

2.3 Phase 1: Adapter Library Reuse

The first phase measures how much of the new task the library already covers. All new adapters are disabled, so that Eq. (1) reduces to a linear combination of past adapters, and only $c^{(t)}$ is trained under the backbone's imitation loss $\mathcal{L}_{\mathrm{im}}$, stopping when the loss plateaus. What the coefficients cannot fit is the reuse residual that the later phases represent.

2.4 Phase 2: Sparse Adapter Allocation

The second phase determines at which layers a new adapter is worth keeping. A candidate adapter and a scalar gate γ_ℓ are attached to every layer, and the gate scales the layer's candidate output, so $\gamma_\ell = 0$ removes the adapter from the update. With the coefficients of Phase 1 held fixed, all candidates train jointly for a short selection horizon under an ℓ_1-gated imitation objective,

$$\min_{\{A_\ell^{(t)},\, B_\ell^{(t)},\, \gamma_\ell\}} \quad \mathcal{L}_{\mathrm{im}} \;+\; \xi \sum_{\ell} |\gamma_\ell|, \tag{2}$$

where ξ is a single sparsity level shared by all layers. The objective is optimized by proximal gradient descent Beck and Teboulle [2009]: after a short warm-up during which the gates train without thresholding, every gradient step on $\mathcal{L}_{\mathrm{im}}$ is immediately followed by the proximal operator of the ℓ_1 term at step size η, the soft-threshold

$$\gamma_\ell \;\leftarrow\; \mathrm{sign}(\gamma_\ell)\, \max\big(|\gamma_\ell| - \eta\,\xi,\; 0\big). \tag{3}$$

A layer's gate survives only if the gradient it receives outweighs ξ, so the layers where the reuse residual is largest keep their adapters. A gate driven to zero may regrow while the horizon lasts, and the support $\mathcal{S}_t = \{\ell : \gamma_\ell \neq 0\}$ is read off only at its end. The gate magnitudes are then absorbed into the corresponding $B_\ell^{(t)}$, and candidates outside $\mathcal{S}_t$ are discarded. Since ξ is the only quantity controlling how many layers survive, it controls the storage added per task.

2.5 Phase 3: Training with a Duplication Penalty

In the third phase the surviving adapters and the combination coefficients are trained together. The objective adds to $\mathcal{L}_{\mathrm{im}}$ a duplication penalty that measures how much of the new adapter lies in the

span of the library,

$$\min_{\{A_\ell^{(t)},\, B_\ell^{(t)}\}_{\ell\in\mathcal{S}_t},\, c^{(t)}} \quad \mathcal{L}_{\text{im}} \;+\; \lambda \cdot \frac{1}{|\mathcal{S}_t|} \sum_{\ell\in\mathcal{S}_t} \text{dup}_\ell, \qquad \text{dup}_\ell = \frac{b_\ell^\top M_\ell^{-1} b_\ell}{\left\| A_\ell^{(t)} B_\ell^{(t)} \right\|_F^2}, \tag{4}$$

where $M_\ell[i,j] = \langle A_\ell^{(i)} B_\ell^{(i)},\, A_\ell^{(j)} B_\ell^{(j)} \rangle$ collects the pairwise inner products among library adapters at layer ℓ and $b_\ell[i] = \langle A_\ell^{(i)} B_\ell^{(i)},\, A_\ell^{(t)} B_\ell^{(t)} \rangle$ their inner products with the new adapter. The numerator is the squared norm of the projection of the new adapter onto the library span, so $\text{dup}_\ell \in [0,1]$ is the fraction of the new adapter that reuse could already express, and λ sets how strongly this fraction is discouraged. The penalty routes reusable structure into $c^{(t)}$ and leaves the new adapter for directions the library does not contain. For numerical stability M_ℓ is inverted with a small ridge proportional to its largest diagonal entry, and since the library is frozen this inverse is computed once per task.

When training ends, τ_t publishes θ_t together with the task signature described next, and these are kept frozen for the remainder of the stream. Since a later task only attaches its own coefficients and adapters on top of them, the parameters of past tasks are never modified, and their policies remain exactly as trained.

2.6 Task Recognition from Task Signatures

SARAM operates without the oracle task ID: at deployment an episode provides only an observation and an instruction, and the task is identified from these inputs using statistics that the frozen backbone already produces. For a given input, the Gram matrix of a layer's input activations records which directions of the layer's input space are excited and how strongly. These second-order statistics differ across tasks with different scenes and instructions, while episodes of the same task excite similar directions. To build the task signature of τ_t, the first frame and instruction of each training episode in $\mathcal{D}_t$ are passed once through the backbone with no adapters attached, and every token-level input that layer ℓ receives across these passes is added into one running matrix,

$$\begin{aligned} G_\ell^{(t)} &= \textstyle\sum_x x\, x^\top \in \mathbb{R}^{d_\ell \times d_\ell}, \\ U_\ell^{(t)} &= \left[\, u_1\sqrt{\sigma_1},\, \ldots,\, u_k\sqrt{\sigma_k} \,\right] \in \mathbb{R}^{d_\ell \times k}, \end{aligned} \tag{5}$$

where the sum runs over all token inputs from all episodes of τ_t, and (σ_i, u_i) are the top-k eigenpairs of $G_\ell^{(t)}$. Only the sketch $U_\ell^{(t)}$ is stored, which keeps the task signature at $\sum_\ell k\, d_\ell$ values, a size set by the backbone's widths and independent of how many tasks have been learned. At the start of a deployment episode, the same sketch q_ℓ is computed from the single first observation and instruction, and the task is selected by

$$t^* = \arg\max_t \; \frac{1}{L} \sum_\ell \text{sim}\big(q_\ell,\, U_\ell^{(t)}\big). \tag{6}$$

The similarity is the uncentered variant of linear CKA Kornblith et al. [2019], computed between the Gram matrices reconstructed from the two sketches; centering is omitted since Eq. (5) keeps no first-moment statistics The comparison requires one adapter-free forward pass and T products of k-column matrices, and it is performed once per episode, with no selection computation inside the control loop. Because the backbone is never updated, task signatures collected for earlier tasks remain valid as the stream continues.

2.7 Deployment by Merging into the Backbone

Given t^*, the update of Eq. (1) is evaluated from the stored combination coefficients and adapters and merged into the backbone,

$$W_\ell \;\leftarrow\; W_{0,\ell} + \Delta W_\ell^{(t^*)}. \tag{7}$$

The merge is exact because $\Delta W_\ell^{(t^*)}$ is a single linear weight update that can be exactly merged into the backbone, and it holds for the whole episode because t^* is fixed at the start and $c^{(t^*)}$ does not depend on the input. On a task switch the current update is subtracted and the next one added, using

Table 1: Results on the four LIBERO suites (3 seeds, mean$_{\pm \text{sd}}$). Bold marks the best value per column.

	LIBERO-Long			LIBERO-Goal			LIBERO-Spatial			LIBERO-Object		
Method	FWT↑	NBT↓	AUC↑	FWT↑	NBT↓	AUC↑	FWT↑	NBT↓	AUC↑	FWT↑	NBT↓	AUC↑
SeqFFT	70.1 ±2.0	74.7 ±0.8	20.9 ±0.0	**90.7 ±1.0**	87.6 ±1.4	25.6 ±0.5	**87.0 ±0.8**	82.7 ±0.7	26.2 ±0.5	**93.2 ±1.7**	78.4 ±0.3	36.2 ±0.6
SeqLoRA	57.0 ±0.8	66.5 ±1.9	15.7 ±0.4	82.4 ±0.3	83.7 ±0.3	23.4 ±0.1	75.6 ±0.7	77.9 ±2.0	22.4 ±0.5	82.9 ±0.2	75.9 ±1.5	31.1 ±1.1
PackNet	12.4 ±2.8	26.3 ±4.0	1.5 ±0.4	25.9 ±1.8	40.8 ±2.9	3.8 ±0.2	23.5 ±1.7	35.7 ±2.8	2.8 ±0.1	15.2 ±1.4	26.3 ±4.9	1.7 ±0.2
EWC	60.3 ±0.4	70.3 ±1.5	16.8 ±0.9	83.6 ±1.2	85.3 ±0.7	22.9 ±1.0	83.8 ±0.9	83.1 ±0.6	25.3 ±0.2	90.7 ±0.7	74.1 ±1.4	38.7 ±0.6
ER	64.5 ±3.4	18.2 ±1.6	61.6 ±3.5	89.1 ±0.3	10.3 ±2.1	85.8 ±1.6	85.8 ±0.3	11.6 ±2.0	81.8 ±1.7	92.0 ±0.9	6.5 ±1.7	91.8 ±1.4
CLARE	71.4 ±1.6	25.5 ±8.3	61.9 ±4.9	90.1 ±1.0	11.6 ±5.5	84.9 ±5.3	82.4 ±0.3	7.4 ±0.9	83.0 ±0.4	88.4 ±0.8	9.0 ±1.5	88.2 ±0.8
DMPEL	**73.4 ±0.6**	11.1 ±0.7	74.8 ±1.1	88.3 ±0.8	8.7 ±3.4	86.9 ±3.2	83.2 ±1.5	**6.5 ±1.7**	84.3 ±0.7	78.6 ±5.4	9.1 ±1.4	80.6 ±6.3
SARAM (ours)	73.0 ±1.2	**7.5 ±0.9**	**80.9 ±1.7**	86.5 ±2.2	**6.6 ±0.9**	**89.1 ±2.1**	83.8 ±1.9	6.8 ±2.4	**86.4 ±3.1**	87.2 ±8.0	**3.1 ±1.4**	**92.9 ±2.4**

deltas kept in host memory; W_0 is also retained off-GPU so that the weights can be re-based from it when needed.

After Eq. (7), no adapter, coefficient, or selection module is present in the control loop, and the resident model is a single backbone with modified weights. Its memory footprint and per-step cost are therefore those of the backbone, regardless of T.

3 Experiments

3.1 Experimental Setup

Benchmark and protocol. We evaluate on the four LIBERO suites Liu et al. [2023]: Long, Goal, Spatial, and Object, each a stream of $T{=}10$ tasks learned sequentially. Following the LIBERO protocol, each task is probed with rollout episodes during training and its best checkpoint is deployed as the final policy. Every method runs three seeds under one shared schedule: 10k training steps per task, 20-episode probes every 800 steps, and 40 episodes for every retention evaluation, executed under each method's own task selection without the oracle task ID.

Metrics. Let $s_{j,t}$ be the success rate of task τ_t evaluated after the stream has learned τ_1 through τ_j. Let $\bar{s}_t$ and $\hat{s}_t$ be the mean and best success rate of the rollout probes recorded while τ_t itself trains with known task identity; the deployed checkpoint achieves $\hat{s}_t$. We report the LIBERO metrics Liu et al. [2023]: $\mathrm{FWT} = \frac{1}{T}\sum_{t\in[T]} \bar{s}_t$, how well each task is learned during training; $\mathrm{NBT} = \frac{1}{T}\sum_{t\in[T]} \mathrm{NBT}_t$ with $\mathrm{NBT}_t = \frac{1}{T-t}\sum_{j=t+1}^{T}(\hat{s}_t - s_{j,t})$, the average drop from the deployed score when earlier tasks are revisited, so lower is better; and $\mathrm{AUC} = \frac{1}{T}\sum_{t\in[T]} \frac{1}{T-t+1}\big(\bar{s}_t + \sum_{j>t} s_{j,t}\big)$, which summarizes the whole learn-then-retain trajectory.

Baselines. We consider sequential adaptation baselines that overwrite previously learned parameters: SeqFFT, which fine-tunes the whole model for each new task, and SeqLoRA, which trains LoRA adapters and merges them into the backbone after each task. We also consider EWC Kirkpatrick et al. [2017], which penalizes changes to parameters important for earlier tasks, PackNet Mallya and Lazebnik [2018], which freezes the most important weights after each task and learns the next within the remaining capacity, and ER Chaudhry et al. [2019], which retains past demonstrations and mixes them evenly with new data. Finally, we include two modular methods: CLARE Römer et al. [2026], which trains task-specific adapters with per-task discriminators for selection, and DMPEL Lei et al. [2025], which builds an expert library with a router fine-tuned by expert-coefficient replay.

Implementation. All methods share one pre-trained backbone: a DiT-based flow-matching VLA policy Dasari et al. [2025] pretrained on LIBERO-90 Liu et al. [2023], with a DINOv2 vision encoder Oquab et al. [2023], a CLIP text encoder Radford et al. [2021], and an action head predicting 16-step chunks of which 8 are executed. SARAM uses rank-8 LoRA adapters and one setting on every suite: sparsity level $\xi{=}2$ with an 800-step selection horizon, duplication penalty $\lambda{=}0.3$, and a coefficient pre-fit that stops when the loss plateaus. Task signatures keep the top-2 eigenpairs per observation-side layer. We build on the official CLARE implementation Römer et al. [2026], which also provides the shared backbone and the sequential, replay, and regularization baselines; DMPEL is adapted from its official code Lei et al. [2025] with only the backbone replaced.

Table 2: Component ablations on LIBERO-Long (mean±sd over 3 seeds). Δ is the AUC change from the full method.

	FWT↑	NBT↓	AUC↑	Δ
SARAM (full)	73.0±1.2	7.5±0.9	80.9±1.7	
w/o coeff. pre-fit	68.7±1.7	6.8±0.9	79.0±0.5	−1.9
w/o gate compet.	68.3±3.7	9.4±4.1	74.1±7.1	−6.7
w/o per-task place.	72.5±2.3	10.8±3.4	77.4±3.8	−3.5
w/o library reuse	67.3±0.7	8.5±3.2	76.1±2.0	−4.7
w/o dup. penalty	67.9±10.5	7.0±2.8	76.8±9.1	−4.1

Table 3: Effect of the sparsity level ξ and penalty weight λ on LIBERO-Long (default in bold; 3 seeds). Params is cumulative storage after ten tasks.

ξ	λ	FWT↑	NBT↓	AUC↑	Params (M)
2	**0.3**	**73.0±1.2**	**7.5±0.9**	**80.9±1.7**	6.9±2.1
2	0.6	68.1±8.1	10.0±2.7	75.3±4.1	6.8±1.9
2	0	67.9±10.5	7.0±2.8	76.8±9.1	6.8±2.0
3	0.3	68.5±5.6	7.4±2.1	75.2±4.4	2.4±0.4
4	0.3	59.8±5.0	11.6±3.5	64.3±4.2	1.0±0.2

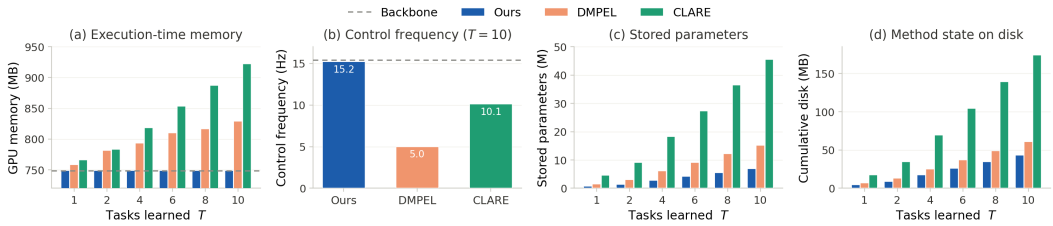


Figure 2: **Deployment cost versus tasks learned.** GPU memory and control frequency stay at the backbone's level for SARAM, and its stored parameters and on-disk state grow far more slowly than modular baselines.

3.2 Results

Main results. Table 1 reports the four LIBERO suites. SARAM achieves the highest AUC on all four suites and the lowest NBT on three, while keeping FWT comparable to the strongest baselines: it retains earlier tasks more effectively than the modular methods without reducing acquisition, and matches or exceeds ER without retaining any demonstrations.

Deployment cost. Figure **??** compares resource cost as the stream grows. GPU memory and control frequency of SARAM stay at the backbone's level because the selected update is merged before execution, whereas CLARE and DMPEL grow as their task-specific modules accumulate; SARAM also adds the least on-disk state per task.

Component ablations. Table 2 ablates SARAM on LIBERO-Long. In *w/o gate competition*, the selected layer locations are replaced with randomly chosen ones of equal count, isolating placement from capacity; in *w/o per-task placement*, the first task's placement is reused throughout; *w/o duplication penalty* sets λ=0. Random placement causes the largest AUC drop, per-task placement matters because suitable locations differ across tasks, and removing library reuse, the duplication penalty, or the coefficient pre-fit each also reduces performance.

Effect of ξ and λ. Table 3 varies the sparsity level and duplication strength. Increasing ξ trades continual-learning performance for storage, and at fixed ξ a moderate λ performs best: too small gives insufficient pressure against duplicating reusable directions, too large overly constrains the new adapters.

4 Conclusion

We presented SARAM, an adapter-based continual learning method that reuses frozen adapters from past tasks and sparsely allocates new adapters where additional capacity is needed. At deployment, a task signature selects the task without a learned selection module, and the corresponding update is merged into the backbone so that only the backbone runs during control. On four LIBERO suites, SARAM achieves the highest AUC across all suites while adding less task-specific state than the modular baselines, and keeps GPU memory and control frequency close to the backbone's level as the task stream grows. Our current evaluation focuses on simulated task streams, and future work will evaluate SARAM on longer streams and physical robots.

References

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15617–15625. IEEE, 2025.

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. *arXiv preprint arXiv:2106.09685*, 2021.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMlR, 2019.

Yuheng Lei, Sitong Mao, Shunbo Zhou, Hongyuan Zhang, Xuelong Li, and Ping Luo. Dynamic mixture of progressive parameter-efficient expert library for lifelong robot learning. *arXiv preprint arXiv:2506.05985*, 2025.

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

Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. Tail: Task-specific adapters for imitation learning with large pretrained models. In *International Conference on Learning Representations*, volume 2024, pages 16330–16353, 2024.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7765–7773. IEEE, 2018.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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.

Ralf Römer, Yi Zhang, Yuming Li, and Angela P Schoellig. Clare: Continual learning for vision-language-action models via autonomous adapter routing and expansion. *IEEE Robotics and Automation Letters*, 2026.