Little By Little: Continual Learning via Incremental Mixture of Rank-1 Associative Memory Experts

Haodong Lu[1,2] · Chongyang Zhao[1] · Minhui Xue[2] · Lina Yao[1] · Kristen Moore[2] · Dong Gong[1,*]

[1] University of New South Wales [2] CSIRO * Corresponding author *https://artificer-ai-lab.github.io/MoRAM/*

Scan for project page

ICML International Conference On Machine Learning

1 Continual Learning

Setting. Large pre-trained models are fine-tuned on a sequence of new tasks via lightweight parameter updates (e.g. LoRA)

CATASTROPHIC FORGETTING

Pre-trained knowledge —base model's general capabilities degrade as incrementally learning on new tasks

Fine-tuned knowledge — what earlier task learned, got overwritten by learning on new incoming tasks

2 Current Pitfalls – MoE-LoRA

The expert is an **indivisible block** — three coupled pitfalls follow:

① **Interference:** Coarse experts *bundle* many *low-specialty subspaces*. Activating one for a specific input inevitably triggers irrelevant subspaces — *interfered with knowledge of others*.

② **Redundancy:** New experts cannot *reuse "atoms"* of old coarse experts. If the *existing combination* doesn't cover the new task, the whole block must be relearned — *wasted capacity*.

③ **Routing/Retrieval collapse:** *Under-specialized* experts *confuse* the router. As experts accumulate, router struggles to index them— *routing drifts, old experts misrouted, forgetting*.

*The router is where failure shows up — but the root cause is **coarse-grained, indivisible fixed-rank experts**.*

3 Weight Matrix as a Memory

Linear Associative Memory (LAM). A weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ stores pattern pairs $\{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}$ – keys $\mathbf{k}_i \in \mathbb{R}^{d_{\text{in}}}$ and values $\mathbf{v}_i \in \mathbb{R}^{d_{\text{out}}}$ – as single matrix of outer products

$$\mathbf{y} = \mathbf{W}\mathbf{x} \approx \sum_{i=1}^{m} \mathbf{v}_i(\mathbf{k}_i^\top \mathbf{x})$$

where the inner product $(\mathbf{k}_i^\top \mathbf{x})$ computes the *relevance* (or activation strength) of the i-th memory slot to the input, which weights the retrieval of the *value* vector $\mathbf{v}_i$.

*Each **weight matrix** in the Pre-trained model **is** already a storage of implicit (key, value) **memory atoms**.*

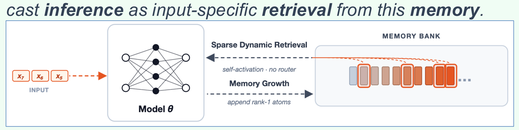
4 Fine-tuning as Memory Augmentation

Low-Rank Updates (LoRA) as Rank-1 Memory Augmentation. LoRA's update $\Delta\mathbf{W} = \mathbf{BA}$ is itself a key-value memory: column vectors $\mathbf{B}_{:,i}$ are values, row vectors $\mathbf{A}_{i,:}$ are keys.

$$\Delta\mathbf{W}\mathbf{x} = \sum_{i=1}^{r} \underbrace{\mathbf{B}_{:,i}}_{\text{Value } \mathbf{v}_i} \big(\underbrace{\mathbf{A}_{i,:}}_{\text{Key } \mathbf{k}_i^\top} \mathbf{x} \big)$$

A rank-r update inserts r new key-value memory atoms into the pre-trained model's intrinsic linear associative memory.

***Continue Learning** as maintaining and expanding a **parametric memory** — a growing collection of **fine-grained, atomic units** stored outside the base model's weights — and cast **inference** as input-specific **retrieval** from this **memory**.*



*Each **memory** unit encodes an incremental **learning snapshot**, and relevant units are retrieved at test time to **specialize** the model for the input—**recall** and **reuse** of memorized snapshots.*

5 Mixture of Rank-1 Memory (MoRAM)

MoRAM Formulation. Adaptation parameters become a dynamic memory bank of rank-1 atoms:

$$\mathcal{M}_t = \{(\mathbf{B}_{:,i}, \mathbf{A}_{i,:})\}_{i=1}^{r_t},$$

where r_t is the number of atoms accumulated by task t. For input $\mathbf{x}$, the update is a sparse, input-dependent mixture over $\mathcal{M}_t$.

$$\Delta\mathbf{W}^t = \sum_{i=1}^{r_t} \mathrm{w}_i \mathbf{B}_{:,i}\mathbf{A}_{i,:},$$

Self-Activation of Each Atom via Memory Keys

Each atom's own key $\mathbf{A}_{i,:}$scores its relevance to the input:

$$s_i = \frac{\mathbf{A}_{i,:}\mathbf{x}}{\sqrt{\sum_{j=1}^{r_t}(\mathbf{A}_{j,:}\mathbf{x})^2}}$$

✗ Auxiliary learned router module → Router induced forgetting

✓ Content-addressable retrieval → Atom knows its relevance

Sparse Memory Retrieval Drives Specialization

1. Top-k masking — keep only the $k$most relevant atoms; mask the rest:

$$[\mathrm{TopK}(\mathrm{s}, k)]_i = \begin{cases} \mathrm{s}_i, & \text{if } \mathrm{s}_i \in \text{top-}k(\mathrm{s}) \\ -\infty, & \text{otherwise.} \end{cases}$$

2. Sharpening — temperature softmax concentrates mass on specialists:

$$\mathrm{w}_i = \mathrm{softmax}\left(\frac{\mathrm{TopK}(\mathrm{s}, k)}{\tau_{\text{MoRAM}}}\right)_i$$

3. Test-time thresholding — prune weak signals at inference:

$$\mathrm{w}_i := \mathbb{1}\{\mathrm{s}_i \geq \delta\} \odot \mathrm{w}_i$$

Forward: focused activation. Backward: gradients flow only to winning atoms — faster specialization, frozen atoms stay frozen.

Learning "*Little by Little*"

For each new task: add r new atoms (Keys $\mathbf{A}$, Values $\mathbf{B}$), freeze all prior atoms, let self-activation route over the union.

Each task becomes an atomic learning snapshot in memory — recalled and reused at test time, never overwritten.

Training: standard loss only. No regularization. No load-balancing. Fine-grained atoms specialize naturally; the self-activated memory retrieval stays stable as memory grows.

5 Experiments

X-TAIL — CLIP (Vision-Language)
Backbone: CLIP ViT-B/16
Tasks: Aircraft, Caltech101, DTD, EuroSAT, Flowers, Food, MNIST, OxfordPet, StanfordCars, SUN397.
Metrics:
Transfer ↑ — zero-shot accuracy on unseen domains
Average ↑ — running mean across all stages
Last ↑ — accuracy after the full stream

TRACE — LLMs (language)
Backbones: LLaMA-2-7B, Gemma-2B, LLaMA-3.2-1B
Tasks: stance detection, finance QA, scientific summarisation, code generation, science QA, math word problems, German news, dialogue.
Metrics:
OP ↑ — overall performance (mean accuracy after the stream)
BWT ↓ — backward transfer (accuracy drop on prior tasks; 0 = no forgetting)

Results — X-TAIL (CLIP)

Method	Transfer ↑	Average ↑	Last ↑
CLIP zero-shot	62.4	—	—
LwF	47.7	53.2	64.0
WiSE-FT	52.3	54.2	58.0
iCaRL	61.7	61.0	64.0
ZSCL	59.0	60.0	63.4
MoE-Adapter	56.0	63.0	70.5
RAIL-Primal	62.4	70.7	79.1
CoDyRA	63.2	71.3	79.2
MoRAM (ours)	**63.3**	**72.7**	**80.9**

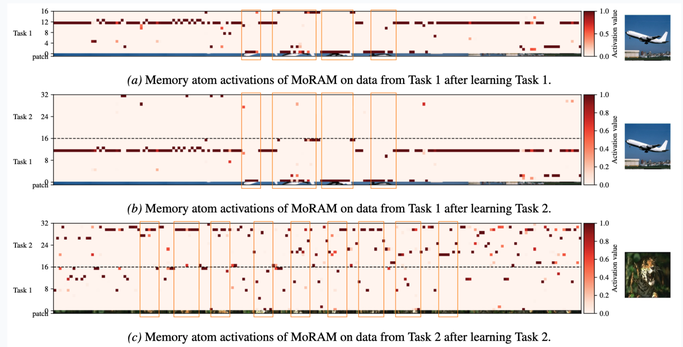
- **Transfer** — learns new tasks without losing pre-trained general capability.
- **Average & Last** — fine-grained atoms specialize without overwriting prior knowledge.

Ablation of Memory Retrieval Strategies

Routing	Transfer	Average	Last
Coarse-Grained (Rank-r Experts)			
MoE-LoRA (Baseline)	62.56	69.45	74.53
w/ Temperature Scaling (τ^*)	62.48	69.40	74.57
Fine-Grained (Rank-1 Memory Experts)			
External Router (Learned $\mathbf{W}_r$)	60.09	65.97	69.76
Self-Activated Retrieval	60.26	65.94	69.85
w/ Sparsity Constraint (Top-k)	60.69	66.52	70.62
w/ Temperature Scaling (τ_{MoRAM})	62.07	71.15	79.62
w/ Threshold-based Selection (δ)	60.78	66.83	71.08
MoRAM (Full)	**63.30**	**72.70**	**80.90**

- **Sparse retrieval drives specialization on fine-grained atoms** — top-k, temperature, and threshold each reduce interference and redundancy.
- **No gain on coarse MoE-LoRA** — the structural win is fine-grained memory, not the sparsity machinery on top.

Analysis — What Do Atoms Learn?



(a) Memory atom activations of MoRAM on data from Task 1 after learning Task 1.

(b) Memory atom activations of MoRAM on data from Task 1 after learning Task 2.

(c) Memory atom activations of MoRAM on data from Task 2 after learning Task 2.

Visualization. Per-atom mixture weights in an activation map: rows = atom indices (by task), columns = input patches, colour = activation value. (More detailed visualizations in the paper Fig. 6 – Forgetting mitigation and Fig. 7 – Knowledge reuse.)

Specialization. A *concentrated* subset of specialized atoms fire strongly on specific patches; the rest stay near zero.
Forgetting mitigation. Activation patterns for Task-1 inputs stay *invariant* after learning new tasks.
Knowledge reuse. Prior-task atoms *re-fire* on later-task inputs when similar semantics recurs (e.g., sky in task 1 re-fired for blue-sky task 9, see Fig. 7 in paper for more details) — no redundant expert/memory needed.

Results — TRACE (LLMs)

	FIX(ICL)	SeqLoRA	OGD	GEM	EWC	L2P	DualPrompt	HiDeLoRA	O-LoRA	TreeLoRA	MoRAM
meta-llama / LLaMA-2-7B-Chat											
OP	38.94 ± 0.3	34.3 ± 1.2	42.09 ± 1.6	40.08 ± 1.6	42.36 ± 1.2	36.23 ± 0.8	37.69 ± 1.2	41.60 ± 0.8	42.78 ± 0.8	43.52 ± 1.0	**44.54 ± 0.9**
BWT	–	18.5 ± 0.8	8.06 ± 1.2	6.77 ± 1.2	5.97 ± 0.8	8.25 ± 0.8	8.03 ± 0.8	7.12 ± 0.4	7.16 ± 0.4	3.46 ± 0.4	**1.37 ± 0.3**
google / Gemma-2B-it											
OP	32.3 ± 0.2	31.89 ± 0.8	32.85 ± 1.4	26.48 ± 1.5	28.35 ± 1.6	31.14 ± 1.2	32.42 ± 1.0	33.25 ± 0.9	33.73 ± 0.8	33.41 ± 0.9	**36.27 ± 0.7**
BWT	–	15.28 ± 0.4	12.27 ± 0.9	18.25 ± 0.9	16.96 ± 1.2	15.77 ± 0.7	14.25 ± 0.5	13.66 ± 0.5	12.36 ± 0.4	8.50 ± 0.5	**2.74 ± 0.4**
meta-llama / LLaMA-3-1B-Instruct											
OP	31.16 ± 0.4	29.73 ± 1.6	30.12 ± 2.0	32.19 ± 2.0	31.96 ± 1.6	29.38 ± 1.2	30.76 ± 1.2	33.73 ± 1.2	32.94 ± 0.8	36.14 ± 0.7	**37.77 ± 0.8**
BWT	–	17.03 ± 1.2	15.2 ± 1.6	10.74 ± 1.6	11.62 ± 1.2	13.57 ± 0.8	11.34 ± 0.8	12.36 ± 0.8	12.89 ± 1.2	7.36 ± 0.8	**3.12 ± 0.8**

Best OP and lowest BWT on every backbone. Frozen keys + content-addressable retrieval over specialized memory atoms with the memory bank — old atoms don't drift as new tasks arrive.

Results — Supervised Fine-Tuning (SFT)

Method	HumanEval (Pass@1)	Out-of-Domain (Acc.)								Params (M)	%Params
		Logic	Phil.	Reli.	Econ.	Pub. Rel.	STEM	Phys.	ML	MMLU		
Llama-3.1-8B	38.40	42.06	71.06	83.63	70.17	68.18	54.84	39.22	40.18	63.45	—	—
LoRA ($r = 4$)	41.46	39.68	70.09	81.87	71.43	65.45	54.77	45.10	40.17	63.28	10.5	0.13%
LoRA ($r = 8$)	44.51	39.68	70.74	81.87	71.85	64.54	54.17	42.16	39.29	63.03	21.0	0.26%
LoRA ($r = 16$)	45.73	41.27	68.49	80.70	72.69	66.36	54.96	44.11	38.39	63.35	41.9	0.52%
LoRA ($r = 32$)	47.56	42.85	69.45	81.87	72.27	66.36	55.44	45.10	39.29	63.59	83.9	1.03%
MoRAM	47.56	48.41	70.09	82.46	73.53	68.18	55.53	46.08	41.96	63.70	41.9/**26.2**	0.52%/**0.32%**

Retained (and possibly improved) Out-of-Domain Generalization After SFT on Code-Alpaca.