JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Can Sparse Memory updates allow LLMs to continually learn?

CSCI 601-771 (NLP: Self-Supervised Models)

Jalil, Zhengguang, Dengjia, Jonathan

https://self-supervised.cs.jhu.edu/fa2025/

Challenge of Continual LLM Learning

- **Continuel Learning = models that can be taught like students**
  - Experience + human feedback -> makes model smarter overtime
- Two subproblems in continual learning of LLMs :

Generalization:
- What is the *right* update from new data?
- Need augmentations to disambiguate the intended concept.
- Real-world learning requires active self-supervision.

Integration:
- Update without forgetting
- Must overwrite outdated info but preserve reusable knowledge.
- Requires sparse, targeted parameter updates

What properties do we want?

Target Updates
- (touch minimal parameters)

High Capacity
- (lifetime of learning)

Adaptive integration
- (what to overwrite/preserve)

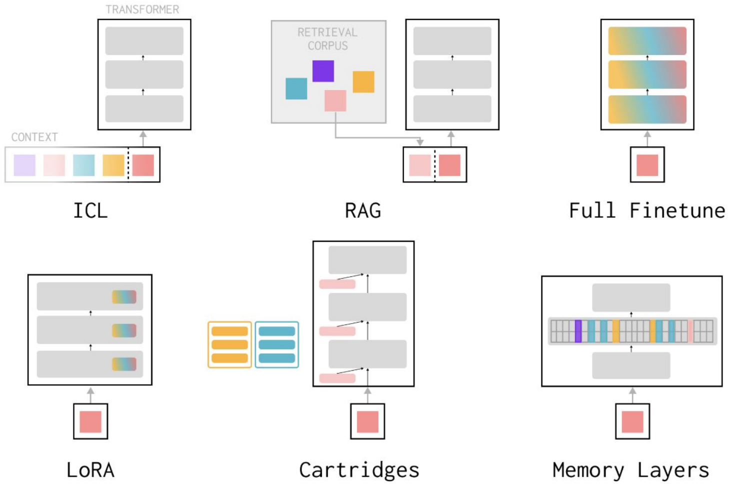
Current Continual Learning Approaches

Non-Parametric approaches:

- **ICl:** short-term, suffers from context rot.
- **RAG:** high capacity but no compression into weights.

Parametric approaches:

- **Finetuning+Replay:** prevents forgetting but not scalable.
- **Parameter-Efficient Finetuning (cartridge):** targeted but low capacity; unclear task boundaries.
- **MoE:** large capacity; finer-grained experts help.
- **Memory Layers:** many small experts enabling targeted, scalable lifelong learning.



Proposed Solution: Sparsity

Structural Sparsity: The Capacity Fix

Memory Layers at Scale:
A dedicated, sparse architecture providing billions of extra parameters to store new knowledge cheaply.

Learning Sparsity: The Update Fix

Sparse Memory Finetuning:
An intelligent, selective update rule that modifies only specific memory slots, mitigating interference.

Memory Layers at Scale

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, Gargi Ghosh

Problem Statement / Motivation

Motivation

Modern LLMs acquire factual knowledge by **absorbing it into dense parameters**, meaning:
To remember more facts:

- we must **increase model size**,
- which increases **training cost, inference cost, and energy consumption**.

Even with Mixture-of-Experts (MoE) architectures: memory capacity still scales with compute, routing is unstable,and experts overlap instead of storing clean, separable knowledge.
Yet, **a large portion of language model behavior is not reasoning** — it is:

- retrieving simple associative knowledge

This type of knowledge does **not require deep computation**,but it consumes expensive model parameters.
So the central motivation is:

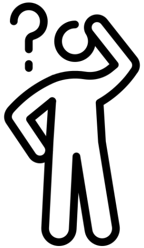
How can we expand knowledge capacity, without making models heavier or more expensive to run?

The authors' hypothesis:
Treat factual knowledge **not as computation**,
but as **memory lookup**.
If we replace some transformer layers with **trainable key–value memory**,
the model can store much more knowledge **without proportional FLOPs increase**.

Problem Statement / Motivation

Problem: How LLMs Store Facts

- LLMs memorize facts in **dense parameters**.
- More facts → l**arger models** → higher **training / inference cost** and **energy use**.
- Even with **Mixture-of-Experts** (MoE), memory capacity still roughly scales with compute; routing can be unstable and experts overlap.
- But a lot of what LLMs do is **simple fact retrieval**, not deep reasoning.



How can we expand knowledge capacity without making models heavier or more expensive to run?

Problem Statement / Motivation

The Author's Hypothesis:

- Treat factual knowledge **not as computation**, but as a **memory lookup**.
- Replace some transformer layers with **trainable key-value memory**, so the model can store much more knowledge **without proportional FLOPs increase**

Core Idea / Method

Core Ideas

- Replacing the feed-forward network (FFN) of one or more transformer layers with trainable memory layers
- Trainable Memory Layers work very similiarly with Attention: $q \in R^{N}, K \in R^{N \times n}, V \in R^{N \times n}$:
  N is hyperparameter of number of memory slots, n is hyperparameter of embedding vector
- $q = W^{q}h, \;\; s_i = qK^{i}, I = top\ indices\ of\ s, \alpha = softmax(S_I), y = aV$
- The memory layer searches over learned memory slots,ranks them by similarity to the query, and retrieves a weighted combination of the most relevant stored values.
  This output replaces the FFN output inside the Transformer block.

Intuitively Why Memory Layers Work?

Explicit Knowledge Storage

Memory layers let models store factual knowledge in dedicated slots, instead of diffusing it across dense parameters — enabling targeted recall rather than overfitting compute-heavy layers.

Sparse Retrieval

Only the top-k slots activate per query, so memory cells specialize automatically (coding patterns, capitals, entity templates) with almost no interference — enabling higher capacity without catastrophic forgetting.

Scaling Capacity Without Increasing Compute

Unlike dense scaling, adding more memory slots increases knowledge capacity dramatically without increasing FLOPs, because retrieval stays sparse — giving exponential memory at near-constant cost.

Intuitively Why Memory Layers Work?

Explicit Knowledge Storage

- Facts live in **dedicated memory slots**, not tangled in all the weights.

- Easier to **store, update, and recall** specific pieces of knowledge.

Sparse Retrieval

- For each query, only **top-k slots** are activated.

- Slots naturally **specialize** (e.g., capitals, entities, code patterns) with **little interference**.

Scaling Capacity, Not Compute

- We can add more **memory slots** to store more facts.

- Retrieval stays **sparse**, so **FLOPs barely increase** while knowledge capacity grows.

Scaling / Implementation Details

Product-Key Lookup (Efficient Retrieval)

- Scaling memory layers is limited by expensive nearest-neighbour search over large key spaces.
- The model avoids this by factorizing keys into two smaller “half-key” sets.
  Queries are split and matched against these smaller sets to retrieve top-k candidates efficiently.
- Final key scores are computed by combining matches from the two half-key sets, approximating full-space lookup without instantiating it.

Parallel Memory (sharding across GPUs)

- Memory layers are large, so lookup is sharded across GPUs to scale efficiently.
- Each GPU stores only part of the embeddings, performs lookup on its shard, and aggregates partial results.
- This avoids materializing full embeddings on any device, keeping activation memory manageable.
- The approach runs in its own parallel group, independent of other model-parallel schemes.

Scaling / Implementation Details

Product-Key Lookup (Efficient Retrieval)

- Naïve lookup over a huge key space is too expensive.
- Factorize keys into **two smaller key tables**; split each query accordingly.
- Search each table separately, then **combine pairs of matches** to approximate full-space top-k at much lower cost.

Parallel Memory (sharding across GPUs)

- The memory table is too large for one GPU, so we **shard it across GPUs**.
- Each GPU stores a slice of the table, does local lookup, then we **aggregate partial results**.
- Avoids materializing the full table on one device and works in its **own parallel group**, alongside other model-parallel schemes.

Scaling / Implementation Details

Shared Memory Across Layers

- They use a **shared pool of memory parameters** across all memory layers in the network. That is — multiple memory-augmented layers reference the **same** key/value tables.
- They find empirically that replacing more than a few FFN layers with memory helps, but beyond a certain point, further replacement hurts performance (suggesting a balance between dense + sparse layers). In their experiments, up to 3 memory layers was beneficial; beyond that it degraded performance.

Performance & Stability Improvements (Engineering)

- They also introduce an enhanced variant called Memory+, which adds an additional small projection + gating + non-linearity (e.g. SiLU) after retrieval to stabilize training and improve performance.
- For backward pass, gradient updates to the huge embedding tables (values, keys) can collide (many outputs may map to the same slot). They compare different strategies: atomic-add accumulation, row-level locks, and a “reverse-indices / atomic-free” method that maps token IDs to embedding indices to aggregate gradients safely. For high-dimensional embeddings (>128 dims), reverse-indices or lock-based updates are faster than atomic-add.

Scaling / Implementation Details

Shared Memory Across Layers

- All memory layers share **one global key–value table** (a common memory pool).
- This cuts parameters and encourages reuse of the same stored facts.
- Empirically, replacing **~1–3 FFN layers** with memory helps; more than that hurts, so a **mix of dense + memory layers** works best.

Performance & Stability Improvements

- **Memory+** adds a tiny projection + gate (e.g., SiLU) after lookup to stabilize training and improve accuracy.
- For the huge embedding table, they use a batched, atomic-free gradient aggregation scheme instead of naive atomic adds, reducing contention and making updates more stable and efficient.

Experimental Setup

Baselines

- Dense Transformer Models: The paper primarily compares its method against standard dense Llama-style transformer baselines, trained at multiple scales (134M → 1.3B parameters).
- Mixture-of-Experts (MOE): Each feed-forward layer contains multiple “experts,” but only a subset of experts is activated for a given input.→ This increases model capacity without proportionally increasing compute.
- PEER Model (He, 2024):Works similarly to memory layers but retrieves a pair of embeddings that form a rank-1 dynamic feed-forward layer.→ Serves as an alternative parameter augmentation method.

Evaluation Benchmarks

- Factual Question Answering: NaturalQuestions, TriviaQA
- Multi-hop / reasoning QA: HotpotQA
- World knowledge & comprehension: MMLU, HellaSwag, OBQA, PIQA
- Programming / code generation: HumanEval, MBPP
- **Reporting Metrics:** Common accuracy measures are used—Exact Match or F1 for QA, and pass@1 for coding tasks.
- Negative log-likelihood (NLL) is also reported to analyze language modeling quality.

Experimental Setup

Baselines

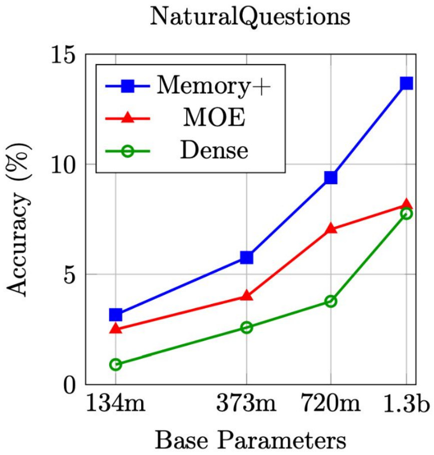
- **Dense Transformers:** LLaMA-style models at multiple sizes (≈134M → 1.3B parameters).
- **Mixture-of-Experts (MoE):** FFN layers with many experts; only a few are active per token → higher capacity at similar compute.
- **PEER (He, 2024):** Retrieves a pair of embeddings to form a rank-1 dynamic FFN layer → alternative way to add parameters.

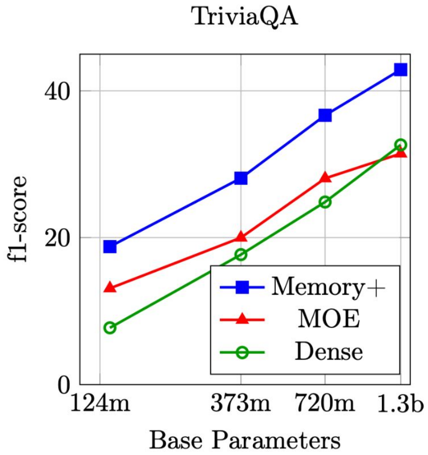
Evaluation Benchmarks

- **Factual QA:** NaturalQuestions, TriviaQA
- **Multi-hop / reasoning QA:** HotpotQA
- **World knowledge / comprehension:** MMLU, HellaSwag, OBQA, PIQA
- **Code generation:** HumanEval, MBPP
- **Metrics:** Exact Match / F1 for QA, pass@1 for coding, and NLL for language-model quality.

Scaling Results-Same # of parameters

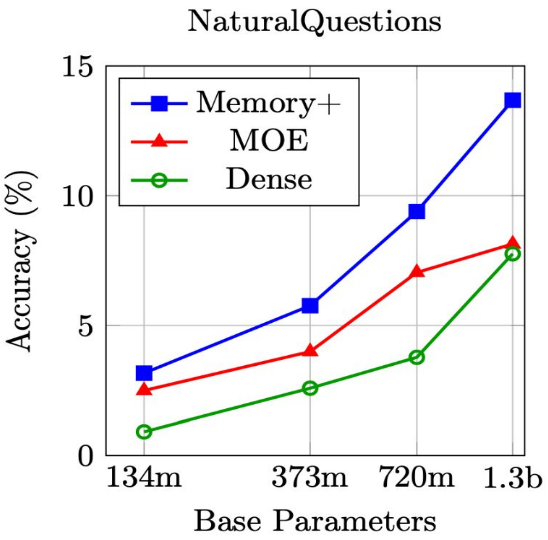
- Parameter budgets are deliberately matched across architectures.
- Memory models replace FFN layers with a shared memory table, keeping total parameters unchanged.
- Memory+ adds additional memory layers but reuses the same shared memory, so its footprint stays the same.
- PEER is configured with a slightly different half-key size to reach similar total parameters.
- MOE picks the minimum number of experts required to match Memory's parameter scale.
- Therefore, all models have nearly identical parameter counts and compute—the difference lies only in how the parameters are allocated and organized.

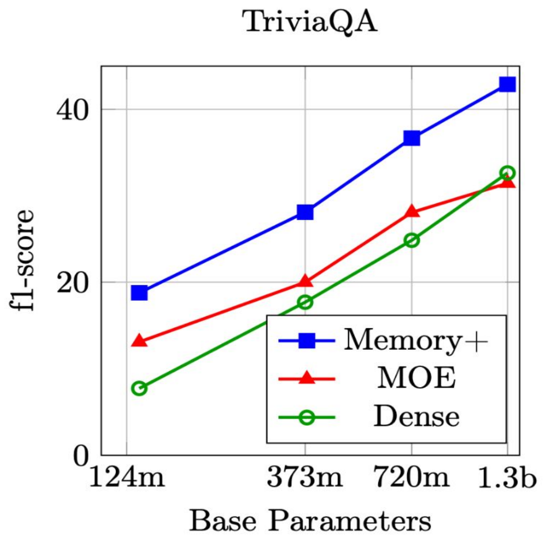




Scaling Results - Same # of parameters

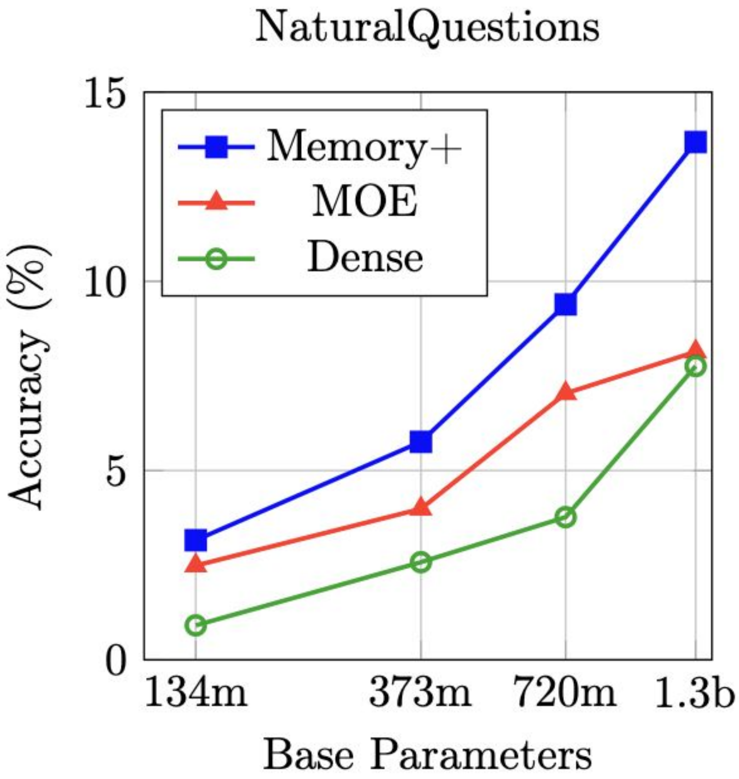
- All models are tuned to have **similar total parameters & FLOPs**.
- Memory models replace some FFN layers with a **shared memory table** (Memory+ adds more memory layers but reuses the same table).
- PEER and MoE are configured to **match this parameter scale** (adjusted key size / number of experts).
- Because budgets are matched, the curves on the right show **how we allocate parameters** (dense vs. MoE vs. memory), not how many we use. ***Memory+ consistently wins under the same parameter budget***.





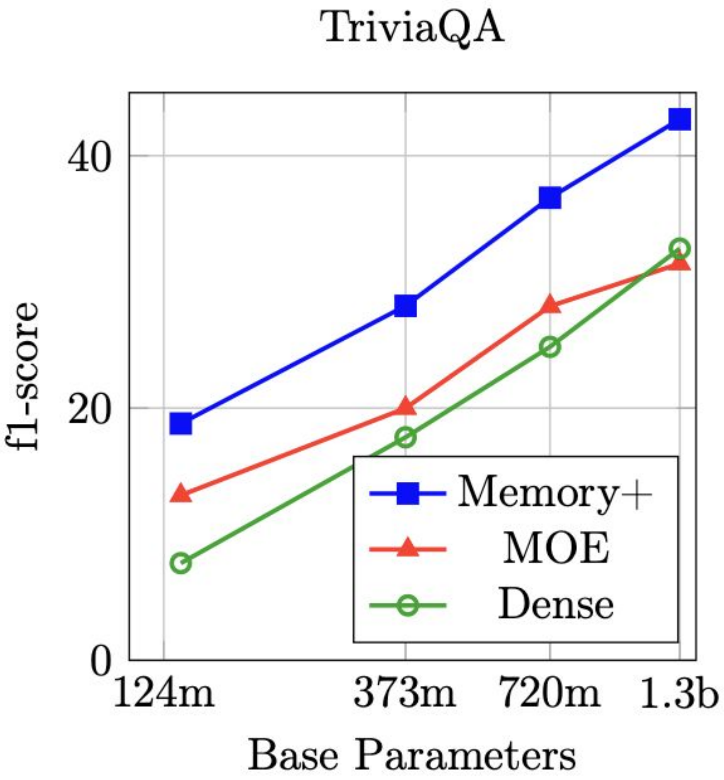
Scaling Results - Scaling Memory Params

- Scale MOE, Dense baseline, Memory+ to approximately-equal parameter counts
- Compare performance on datasets



Scaling Results - Scaling Memory Params

- Scale MOE, Dense baseline, Memory+ to approximately-equal parameter counts
- Compare performance on datasets



Scaling Results-Results at 8B Scale

Model	HellaS.	Hotpot	HumanE.	MBPP	MMLU	NQ	OBQA	PIQA	TQA
llama3.1 8B (15T)	60.05	27.85	37.81	48.20	66.00	29.45	34.60	79.16	70.36
dense (200B)	53.99	20.41	21.34	**30.80**	41.35	18.61	**31.40**	78.02	51.741
Memory+ (200B)	**54.33**	**21.75**	**23.17**	29.40	**50.14**	**19.36**	30.80	**79.11**	**57.64**
dense (1T)	58.90	25.26	29.88	**44.20**	59.68	25.24	34.20	**80.52**	63.62
Memory+ (1T)	**60.29**	**26.06**	**31.71**	42.20	**63.04**	**27.06**	**34.40**	79.82	**68.15**

Memory+ significantly improves data efficiency (models learn facts faster)

- Approaches Llama 3.1 (trained on 15T tokens) when trained on only 1T tokens
- When trained on 200B tokens, already outperforms baseline models (without memory layers, just standard FFN)

Model Ablations: FFN layers

Vanilla Memory paradigm: A single memory layer

Memory+: Multiple memory layers.

Best results: 3 memory layers at centered spaces, with large strides (layers 4, 12, 20)

“Sweet spot” - take advantage of faster learning, without losing too many dense layer parameters.

layer #	nll	NQ nll	TQA nll
12	2.11	12.13	8.34
12,16,20	2.08	11.60	7.54
8,12,16	2.07	11.79	7.64
4,12,20	**2.06**	**11.32**	**7.20**
5,8,11,14,17,21	2.11	11.79	7.73

Model Ablations: FFN layers

Various architectural tweaks

Authors preferred swilu as it gave consistent gains

	nll	NQ nll	TQA nll
Model			
PK base	2.11	12.13	8.34
+gated	2.11	12.24	8.17
+swilu	2.11	12.05	8.09
+random values	2.11	12.36	8.09
+softmax sink	2.11	12.19	8.04

Model Ablations: Key/value dimensions

v_dim	#values	nll	NQ nll	TQA nll
64	16m	2.15	12.86	8.75
256	4m	2.14	12.63	8.49
1024	1m	**2.11**	**12.13**	**8.34**
2048	512k	2.14	12.49	8.53

Value dimension tradeoff: Higher dimension, fewer value outputs in memory.

Default: Value dimension = model dimension (1024)

Authors find default to be optimal

Model Ablations: Key/value dimensions

Increasing key dimension to 2048 boosts performance, but adds more dense parameters, breaking parameter-matched comparisons

- Ambiguous: Better architecture, or more parameters?

Selected key dimension: Half of the base-model dimension (comparison fairness)

key_dim	nll	NQ nll	TQA nll
256	2.11	12.13	8.34
512	2.12	12.32	8.15
1024	2.11	12.37	8.25
2048	**2.09**	**11.98**	**7.83**

Ablations: Summary

Memory layer placement

- Optimal performance: 3 layers, spaced evenly at large stride

Architectural tweaks

- Swilu - most consistent gains; their only adopted tweak

Key/Value dimension choices

- Value dimension: Default (1024, same as model dimension) best.
- Key dimension: Increasing it boosts performance, but adds more dense parameters. Authors fix it at ½ base model dimension for fair comparison.

Recap - What This Paper Does

- **Problem:** LLMs store facts in dense weights → scaling knowledge = scaling compute & cost.
- **Idea:** Replace some FFN layers with trainable key–value memory layers.
- **Intuition:**
  - Explicit memory slots for facts
  - Sparse top-k retrieval per query
  - Add more memory slots without big FLOP increase
- **Engineering:** product-key lookup, sharded memory across GPUs, shared global memory table, and Memory+ tweaks for stability.

Recap - Key Results & Takeaways

- Under **matched parameter & FLOP budgets, Memory+ > dense and MoE** on QA benchmarks.
- Scaling memory size improves **factual accuracy** and **lowers NLL**.
- At **8B scale**, Memory+ trained on 1T tokens **approaches Llama-3.1 8B** trained on 15T.
- Best configuration: a few **well-placed memory layers** and default value dim; memory is a promising way to grow knowledge **without just making the model bigger**.

Continual Learning via Sparse Memory Finetuning

Jessy Lin , Luke Zettlemoyer, Gargi Ghosh, Wen-Tau Yih, Aram Markosyan, Vincent-Pierre Berges, Barlas Oğuz, ICLR 2025)

The Problem: Catastrophic Forgetting

Static Models after Deployment

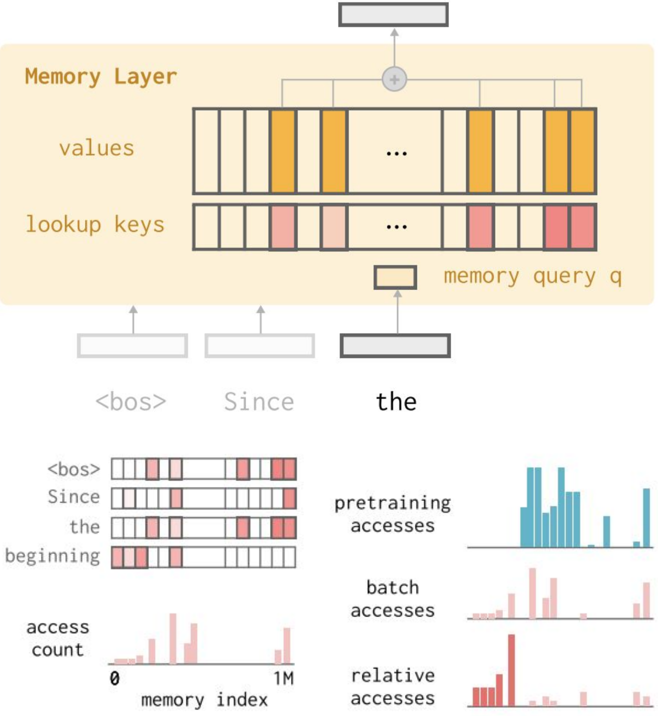
Once deployed, LLMs are typically frozen in time. Updating them on new data streams causes **Catastrophic Forgetting**.

This happens because standard training updates dense parameters shared across all knowledge, causing massive interference. New facts overwrite old ones.

What is needed: For models to learn how to organize its knowledge through end-to-end gradient updates, achieving selective token/parameter updates.

Solution: Sparse Memory Finetuning

- The whole method is based on emory layer(last paper)
- SMFT uses TF-IDF score to select memory slots in memory layer(this paper)



1. Get memory accesses in batch

2. Rank accesses relative to background corpus and train the top t

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

Methodology: Sparse Architecture

The Forward Pass:

Step 1: Retrieve top-k indices based on query projection.
Step 2: Compute scores using Softmax on retrieved keys.
Step 3: Compute weighted output and apply gating.

$$I = \mathrm{TopKIndices}(Kq(x), k)$$
$$s = \mathrm{softmax}(K_I q(x))$$
$$y = sV_I$$
$$\mathrm{output} = (y \odot \mathrm{silu}(x^T W_1))^T W_2$$

Given keys $\mathbf{K} \in \mathbb{R}^{N \times d}$, values $\mathbf{V} \in \mathbb{R}^{N \times d}$, and input $\mathbf{x} \in \mathbb{R}^n$:

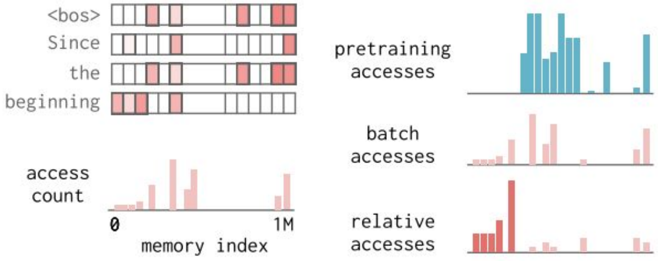
Methodology: The Update Rule

Intelligent Selection (TF-IDF)

- Does not just update *all* active parameters.
  - Filter them using a **TF-IDF score**.
- Prioritizes "knowledge-specific" slots
  - (high frequency in new data, low in general training data) and freezes "common" slots.

For a given memory slot $i \in M$ (where M is all memory slots)

$$\frac{c(i)}{\sum_{j \in M} c(j)} \cdot \log \frac{|B|+1}{\sum_{b \in B} \mathbf{1}_{c_b(i)>0} + 1}$$



1. Get memory accesses in batch

2. Rank accesses relative to background corpus and train the top t

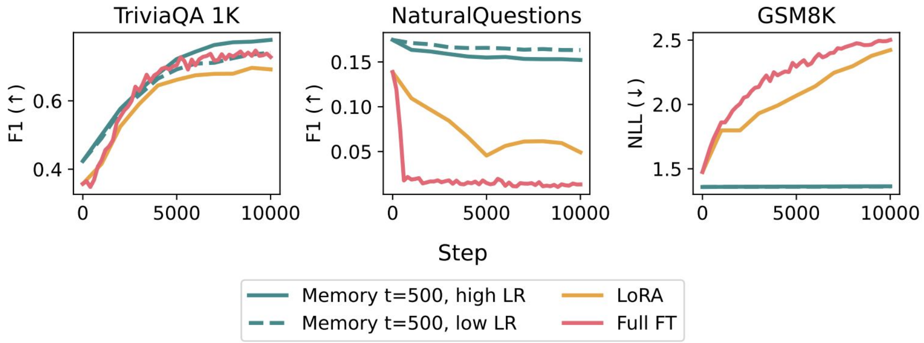
Experiment setup

Base Model Pretraining:
- Built on the **Memory Layers at Scale** model.
- Pretrained using the **DCLM dataset**
- DCLM includes large QA + retrieval-style data sources:
  - Wikipedia passages
  - WikiQA
  - NaturalQuestions (NQ)
  - GSM8k (Math)

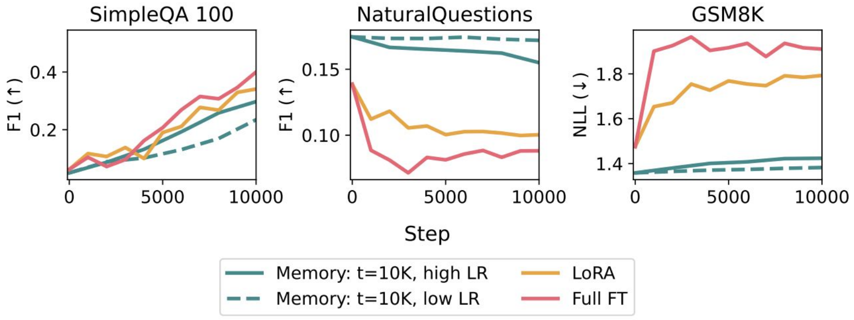
Model starts with strong factual + QA capabilities.

Continual Learning experiments:
1. **TriviaQA** Fact Stream
   a. 1,000 **TriviaQA** facts presented sequentially.
   b. Measures *acquisition* vs retention of facts.
2. Document Chunk Stream
   a. Sequential Wikipedia-style passage chuncks.
   b. Learning evaluated on **SimpleQA**.

Results: Mitigating Forgetting



How much old QA knowledge is forgotten while learning new TriviaQA facts



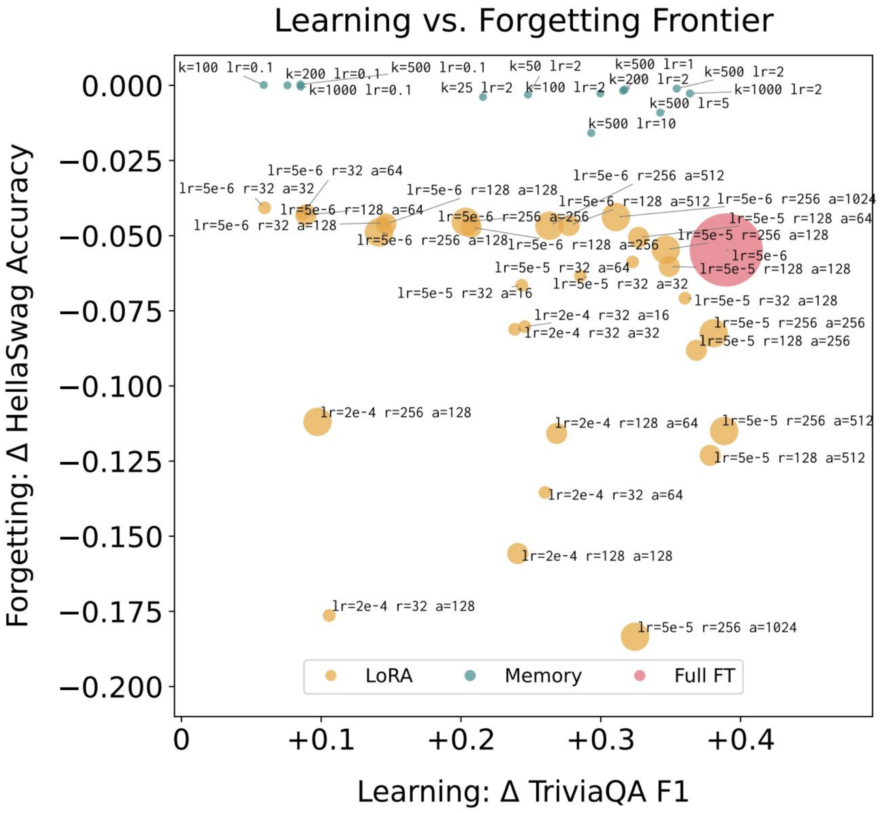
Stability of QA performance when ingesting a continuous stream of document chunks

Result: SMFT significantly reduces catastrophic forgetting compared to baseline methods, outperforms Full Finetuning by ~8x in retention metrics.

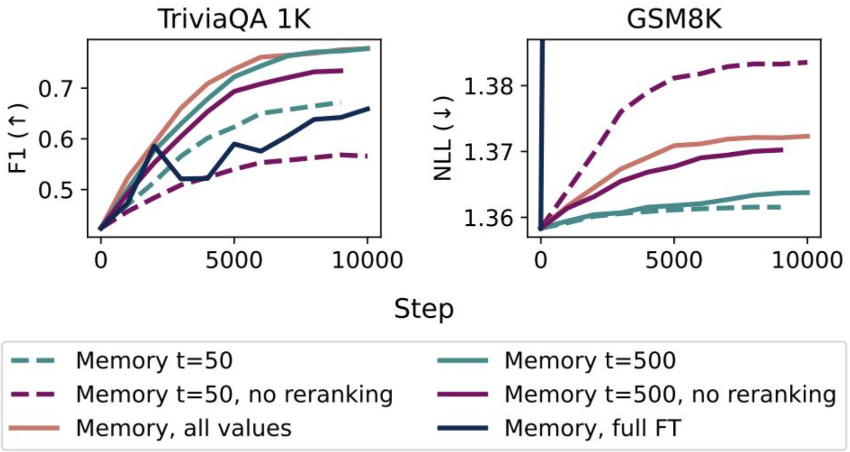
Pareto Efficiency: No Trade-off

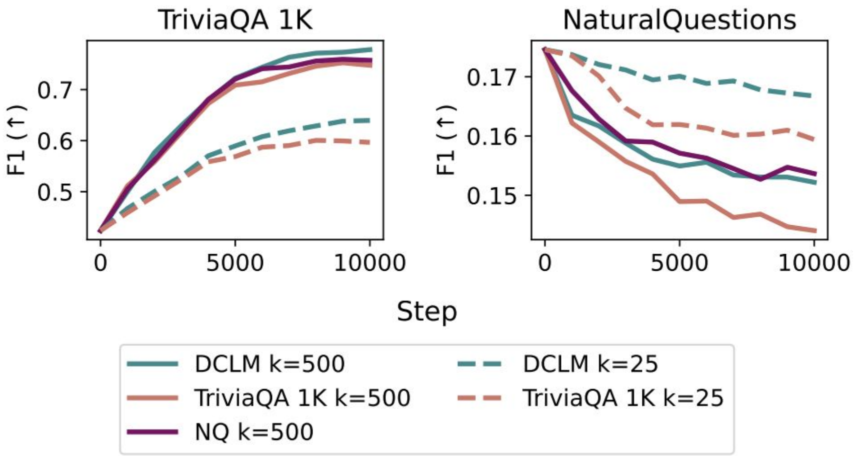
SMFT reaches Pareto dominance with no tradeoff between acquisition and retention

- Full Finetuning: High Acquisition, Low Retention
- LoRA: Moderate Acquisition, Moderate retention
- SMFT: High Acquisition, High Retention



Ablation Study





Retrieval Sparsity alone is not enough; the update must be intelligent (using IDF).

IDF needs a representative background to correctly detect "common" slots.

Understanding Memory Access

- Core & trainable hit **the same memory slots** → shared semantics
- Only **20–100 slots** matter per fact → small trainable subset
- Core meaning drives retrieval, not wording → **semantic indexing**
- Knowledge stored **sparsely and consistently**
- Enables SMFT: **targeted slots = targeted updates**

	Fact index: 174 477 indices in core set, 25 indices needed to answer
Question	
Core	How long was swimmer Michelle Smith-de Bruin banned for attempting to manipulate a drugs test? 4 years<eot>
Trainable	How long was swimmer Michelle Smith-de Bruin banned for attempting to manipulate a drugs test? 4 years<eot>
Paraphrases	
Core	Michelle Smith-de Bruin was given a 4-year ban for attempting to deceive in a drugs test. <eot>
Trainable	Michelle Smith-de Bruin was given a 4-year ban for attempting to deceive in a drugs test.<eot>
Core	Michelle Smith-de Bruin was suspended for 4 years after attempting to deceive in a drugs test.<eot>
Trainable	Michelle Smith-de Bruin was suspended for 4 years after attempting to deceive in a drugs test.<eot>
Core	A 4-year ban was handed down to Michelle Smith-de Bruin for attempting to cheat on a drugs test.<eot>
Trainable	A 4-year ban was handed down to Michelle Smith-de Bruin for attempting to cheat on a drugs test.<eot>

	Fact index: 592 169 indices in core set, 25 indices needed to answer
Question	
Core	What was the name of the cat in Rising Damp? Vienna<eot>
Trainable	What was the name of the cat in Rising Damp? Vienna<eot>
Paraphrases	
Core	A cat named Vienna appeared in the TV series Rising Damp.<eot>
Trainable	A cat named Vienna appeared in the TV series Rising Damp.<eot>
Core	Rising Damp features a notable feline character named Vienna.<eot>
Trainable	Rising Damp features a notable feline character named Vienna.<eot>
Core	The cat Vienna is a beloved part of Rising Damp.<eot>
Trainable	The cat Vienna is a beloved part of Rising Damp.<eot>

	Fact index: 83 193 indices in core set, 100 indices needed to answer
Question	
Core	Who was the first US-born winner of golf's British Open? Walter Hagen<eot>
Trainable	Who was the first US-born winner of golf's British Open? Walter Hagen<eot>
Paraphrases	
Core	The first US-born winner of the British Open was Walter Hagen.<eot>
Trainable	The first US-born winner of the British Open was Walter Hagen.<eot>
Core	Walter Hagen's British Open win was a historic moment for US golfers.<eot>
Trainable	Walter Hagen's British Open win was a historic moment for US golfers.<eot>
Core	Walter Hagen achieved a groundbreaking victory as the first American-born winner of the British Open.<eot>
Trainable	Walter Hagen achieved a groundbreaking victory as the first American-born winner of the British Open.<eot>

Takeaway + Discussion Questions

- SMFT enables continual learning by updating only small, TF-IDF selected subset of memory slots, achieving high plasticity while maintaining stability
- Paper demonstrates that forgetting comes from updating shared dense weights, not from lack of model capacity

Discussion Questions

- Would the sparse update method work for acquiring dense reasoning skills (e.g., coding, math)?
- Possible scores other than TF-IDF for filtering update slots?
- Sparse Memory vs RL continual learning
- What is different between MoE and memory layer
- Limits of memory based continual learning