Memory Adapters Enable Fast, Flexible Knowledge Unlearning in LLMs

Keltin Grimes [1] **Kevin Kuo** [2] **Zhiwei Steven Wu** [2] **Virginia Smith** [2] **Marissa Connor** [1]

Abstract

We introduce *memory adapters*, a fine-tunable adapter for LLMs that allows isolating sequence-level gradient updates to small modular sets of parameters. The modularity of memory adapters enables instantaneous unlearning of any combination of documents, with empirically strong unlearning performance on the TOFU benchmark. Iterative unlearning is costless, and documents can even be flexibly included or excluded for individual queries in a batch. These unique properties are especially valuable for domains where data-removal requests may be unpredictable, granular, or user- or input-dependent.

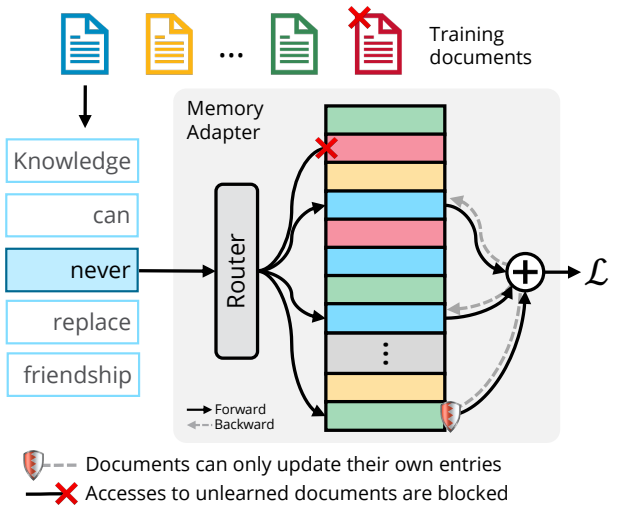

Figure 1. Memory adapters enable instantaneous unlearning of individual fine-tuning sequences. A frozen LLM is augmented with a trainable memory layer, and each document is assigned a set of memory entries which only it can update. After fine-tuning, any document can be unlearned by blocking accesses to its entries.

1. Introduction

The current paradigm of LLM training is not well-suited to the realities of privacy and fair-use regulations. LLMs are monolithic, memorize and reproduce training data (Carlini et al., 2021; Nasr et al., 2025), and are not responsive to individual data-deletion requests or copyright claims – rights enshrined by various regulations worldwide (Bonta, 2022; EU, 2016; Coble, 1998). Ideally, model architectures would be modular, such that individual user data corresponded to particular sub-components of a model that could be easily removed as needed.

Modern LLM architecture research often focuses on a related problem: increasing the sparsity of models, so that parameter count can be scaled without a significant increase in compute. Sparsely activated mixture-of-experts (MoE) models (Shazeer et al., 2017), for example, have famously seen significant adoption (Jiang et al., 2024; DeepSeek-AI, 2024; Qwen Team, 2025). Shi et al. (2025) exploit the sparsity of MoEs for unlearning by training each expert on a single data source, allowing sources to be unlearned simply by removing the corresponding expert. However, this approach is constrained by the scalability of MoE architectures, which typically have tens or hundreds of experts per layer, not enough to represent individual training documents.

To bridge this gap, we introduce *memory adapters*, a fine-tuning technique which leverages memory layers to enable document-level modularity and fast, flexible unlearning. Memory layers (Lample et al., 2019; Berges et al., 2024), push sparsity to the extreme, allowing MoEs to scale to tens of millions of experts. We augment a frozen pretrained LLM with a memory layer and assign each individual data source an independent set of memory entries. During training, we mask gradient updates such that each source updates only its assigned entries. This localizes the information learned from a particular data source into an isolated component. Any document can be subsequently unlearned by blocking accesses to its associated entries.

Memory adapters have several unique advantages over traditional unlearning algorithms:

1. Unlearning is *effective*; we achieve state-of-the-art privacy scores on the TOFU benchmark while maintaining high utility on retained data.

[1]Carnegie Mellon University Software Engineering Institute, Pittsburgh, USA [2]Carnegie Mellon University, Pittsburgh, USA. Correspondence to: Keltin Grimes <kgrimes@andrew.cmu.edu>.

Published at ICML 2026 Workshop on the Impact of Memorization on Trustworthy Foundation Models, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

2. Unlearning is *costless*; data sources can be unlearned by adding their assigned memory indices to a block-list from which memory accesses are disallowed.
3. Unlearning is *flexible*; individual data sources can be included or excluded by editing the block-list, and different block-lists can be used within a single batch.

Our results demonstrate that effective and efficient unlearning can be achieved by rethinking how we architect and train LLMs, and motivate future work on full-stack approaches to AI safety.

2. Related Works

Sparse LLM Architectures. LLMs often employ sparsely activated mixture-of-expert (MoE) architectures to increase parameter count while minimizing inference compute (Jacobs et al., 1991; Jordan & Jacobs, 1994; Shazeer et al., 2017). Domain specialization of experts is common, whether occurring naturally (Lewis et al., 2021) or induced intentionally (Shi et al., 2025), indicating feasibility of isolating behaviors into localized components. A central challenge in increasing the sparsity of MoEs is effectively routing inputs to the needed experts (Shazeer et al., 2017; Lewis et al., 2021; Zhou et al., 2022). Several recent approaches, including memory layers, leverage product keys (Lample et al., 2019), which enable $\mathcal{O}(\sqrt{N})$ lookup over N experts. These experts can be rank-1 MoE experts as in He (2024), but in memory layers are simply vectors (Weston et al., 2014; Lample et al., 2019; Berges et al., 2024). Sparsity is not only helpful for inference performance; Lin et al. (2025) show that sparsely updating memory layers can prevent catastrophic forgetting in continual learning tasks.

Unlearning by Construction. Most unlearning algorithms apply a post-hoc procedure to a pretrained language model (Fan et al., 2026; Li et al., 2024; Dong et al., 2025; Mekala et al., 2025; Maini et al., 2024; Zhang et al., 2024), a model not built with subsequent unlearning in mind. Some techniques, however, augment model architectures and training pipelines to better support unlearning, often by adopting a 'decentralized training, centralized execution' framework. Decentralized training trains independent components for each data source, and centralized execution may involve ensembling over independent models (Bourtoule et al., 2021), merging model weights (Kuo et al., 2025), or combining experts into a MoE model (Shi et al., 2025). In these cases, each data source is an entire task or domain; we support much more granular modularity. When the unlearning target is known ahead of time, gradient-routing techniques can isolate updates for those samples to a removable subcomponent of the network (Cloud et al., 2024; Shilov et al., 2025). Our approach uses a similar gradient masking technique, but does not require pre-specified unlearning targets.

3. Memory Adapters

We consider the task of fine-tuning a pretrained LLM on a set of documents. We expect to receive data-deletion requests which may include any combination of documents. The goal of unlearning is to satisfy these data-deletion requests by removing the influence of the documents while maintaining overall model performance.

To facilitate unlearning after finetuning, we 1) augment one or more MLP layers in a transformer-based LLM with a memory adapter, 2) assign each data source a set of memory values, and 3) mask gradients during fine-tuning such that each data source only updates its associated values. Any data source can then be unlearned by preventing accesses to its associated values. We describe each step in detail below.

Memory Layers. Memory layers (Lample et al., 2019) are a trainable parametric memory that can be queried via an attention-like mechanism. A basic memory layer contains N entries, consisting of learnable keys $K \in \mathbb{R}^{N \times d}$ and values $V \in \mathbb{R}^{N \times d}$, and a projection operator q that maps the input $x \in \mathbb{R}^n$ to $\mathbb{R}^d$. Intuitively, V is a large pool of learned vectors from which k entries can be accessed via their corresponding keys. The similarity between the query $q(x)$ and the keys ($Kq(x)$) determines which keys are selected. We define the layer $\text{Memory}(x) = y$ where:

$$\begin{aligned}
&\mathbb{I} = \text{TopKIndices}(Kq(x), k) && \text{\# Retrieve top-k indices} \\
&s = \text{softmax}(K_{\mathbb{I}} q(x)) && \text{\# Compute scores} \\
&y = sV_{\mathbb{I}} && \text{\# Compute weighted output}
\end{aligned}$$

To make key retrieval more efficient, modern memory layers use product keys to accelerate the expensive TopK step (Lample et al., 2019; Berges et al., 2024). Product keys use two sets of keys $K_1, K_2 \in \mathbb{R}^{\sqrt{N} \times d/2}$ and split $q(x)$ into two sub-queries $q_1(x), q_2(x) \in \mathbb{R}^{d/2}$. Each set computes top-k similarity scores $s_i = \text{TopK}(K_i q_i(x), k)$ and corresponding indices $\mathbb{I}_i$. The overall top-k indices are selected as $\text{TopK}_{i_1 \in \mathbb{I}_1, i_2 \in \mathbb{I}_2}(s_1[i_1] + s_2[i_2])$, which index into the full set of $(\sqrt{N})^2 = N$ values with $i = i_1\sqrt{N} + i_2$.

Memory Adapters. Instead of assuming access to a pretrained model with memory layers, we add a memory layer as an adapter to one or more MLP layers in an LLM. We define $\text{MemAdapt}(x) = \text{MLP}(x) + \text{Memory}(x)$ using the memory layer defined above with product key routing. We randomly initialize the router (query and key matrices), but initialize all value vectors to zero. This ensures that $\text{MemAdapt}(x) = \text{MLP}(x)$ at the beginning of training, similar to how LoRA adapters are initialized (Hu et al., 2022).

Memory Entry Assignment. Prior to training, we assign n memory values to each data source. This partitions the

values such that no two data sources share a memory value; therefore, removing the values associated with a document affects only that document. We construct the assignment using the TF-IDF metric from Lin et al. (2025). Specifically, we first pass each sequence through the memory adapter and record the values it accesses. We then compute a TF-IDF score for every pair of data sources and entries. Finally, we greedily select the highest-scoring pairs until each data source has been assigned n entries, removing each selected entry from further consideration to enforce non-sharing. Unassigned values are kept frozen as zero vectors.

Gradient Masked Training. When training a memory adapter, we mask the gradient so that each data source only updates its assigned entries. The key operation is the embedding-bag, which fuses the embedding lookup ($V_{\mathbb{I}}$) and weighted sum ($sV_{\mathbb{I}}$) into a single operation which does not need to materialize the intermediate embeddings. As described in Lin et al. (2025), gradients can be masked to update only a defined set of entries with the following pseudocode, where `mem` is the table of memory values of shape `(memory_size, value_dim)` and `trainable_mask` is a mask of length `memory_size`:

```
# Gradient only flows through mask
mem = mem * trainable_mask +
   (1 - trainable_mask) * mem.detach()

result = embedding_bag(
   indices, mem, weights=scores)
```

However, this only applies when the entire batch comes from the same data source, which may not be feasible or effective in many situations. To allow for sequence-level gradient masks, we use a more complicated masking approach, which requires an additional forward pass of the embedding-bag operation. Here, `trainable_mask` is of shape `(batch_size, memory_size)`:

```
masked_scores = scores *
   trainable_mask.gather(1, indices)

# Correct gradients but masked scores
out_grad = embedding_bag(
   indices, mem, weights=masked_scores)

# Correct scores but no gradients
with no_grad():
   out_real = embedding_bag(
      indices, mem, weights=scores)

# Gradient only flows through out_grad
output = out_grad +
   (out_real - out_grad).detach()
```

This allows for a more natural training setup, and the increase in compute is minimal since memory layers are generally only used in a small percentage of all layers (Lample et al., 2019; Berges et al., 2024).

Note that gradient masking does *not* strictly make unlearning exact. Exact unlearning requires perfect independence between the removed data and the final model. Although we prevent any data source from updating the memory entries of other sources, the information learned by one source is dependent on other sources. As a simplified example, consider two documents, A and B, which both contain some fact f. A is seen first in training, and encodes f in its entries. When B is reached, it can read f from A's entries, and therefore doesn't need to encode f. Thus, B's entries depend on whether or not A is present in the training data. Furthermore, if A (and therefore f) is unlearned, an attacker with knowledge of B could observe that f was not learned and infer that the unlearned document contained f. The TF-IDF-based assignment is designed to minimize accesses to other documents and therefore limit shared knowledge. We freeze the router after initialization to prevent further information leakage to the key and query matrices, although empirically this makes little difference.

Memory Unlearning. When a document needs to be unlearned, we add its associated memory values to a global block-list. During a forward pass, the similarity scores for any entries in the block-list are set to $-\infty$, so they are never accessed. The parameters for those values can also be zeroed out for extra security in case model weights are leaked or the model is otherwise released open-weights.

4. Experimental Results

We report results on the TOFU benchmark (Maini et al., 2024), a popular benchmark for evaluating unlearning knowledge from LLMs.

Setting. The TOFU training set is based on 200 synthetically generated author profiles, with 20 question-answer pairs for each profile. A model is fine-tuned on the full set of Q/A pairs and then unlearning is tested using a forget/retain split of 10%/90% of the profiles. We run all experiments on `Llama-3.1-1B-IT`.

Training Setup. We add a single memory adapter to the 9th layer (out of 16) of the model with 1024^2 values and a value dimension of 2048. This adds 2B parameters to a 1B parameter model, but only adds 0.5% additional FLOPs for a forward pass thanks to the highly sparse memory accesses. We assign 256 values to each author profile. We train the memory layer for 15 epochs with a learning rate of 1×10^{-2}.

Metrics. We report four aggregate metrics: retain utility (Util._{R}), which measures how well the model learned

Table 1. Results of various unlearning algorithms on TOFU. Using memory adapters incurs a slight loss in overall model performance initially, but subsequent unlearning is highly effective.

Method	Util.$_R$	Util.$_G$	Mem.	Priv.	Agg.
Finetuned	1.00	1.14	0.07	0.38	0.21
MemAdapt FT	0.93	1.05	0.18	0.39	0.40
Retrained	1.00	1.11	0.58	1.00	0.87
MemAdapt	**1.00**	1.06	0.62	**0.98**	**0.87**
RMU	0.96	1.12	0.82	0.68	0.86
GradDiff	0.78	**1.26**	**0.98**	0.63	0.85
UNDIAL	0.95	1.16	0.63	0.65	0.79
SimNPO	0.78	1.11	0.56	0.88	0.78
NPO	0.73	1.16	0.66	0.62	0.75
IdkDPO	0.74	0.98	0.62	0.69	0.74
AltPO	0.55	0.96	0.52	0.61	0.62
IdkNLL	0.60	0.92	0.51	0.56	0.62

the retain-set data; general utility (Util.$_G$), to test for performance degradation on questions about real authors and world facts; memorization score (Mem.), which evaluates memorization of the forget-set data; and privacy score (Priv.), a set of membership inference attacks. We also report the harmonic mean of all four metrics (Agg.). All scores are reported such that higher is better. See Dorna et al. (2026) for details on the sub-components of each metric. We differ slightly by breaking utility into retain and general to better distinguish between knowledge acquisition versus loss of existing capabilities. We normalize the former by the finetuned model's score and the latter by the base model.

Baselines. We generate baseline results for a variety of unlearning algorithms following the protocol from Dorna et al. (2026). In particular, we train the full finetuned model, train the models retrained on the retain sets, and run a hyperparameter sweep for SimNPO (Fan et al., 2026), RMU (Li et al., 2024), UNDIAL (Dong et al., 2025), AltPO (Mekala et al., 2025), IdkNLL (Maini et al., 2024), NPO (Zhang et al., 2024), IdkDPO (Maini et al., 2024), and GradDiff (Maini et al., 2024). The best hyperparameters were chosen based on the harmonic mean of model utility and memorization score. See Dorna et al. (2026) for more details.

Results. We present the results on TOFU in Table 1. The initial memory adapter model performs slightly worse on the utility metrics compared to the standard fine-tuned model, likely due to training only a single adapter versus the entire model. Once unlearned, the memory adapter *increases* in utility as the removal of memory entries leads to fewer collisions. The memorization score of the unlearned memory adapter exceeds that of the retrained model, and the privacy score outperforms all of the post-hoc unlearning methods.

Computational Cost. Memory adapters trade off training and inference cost for unlearning cost. In Table 2 we compare the runtime of traditional fine-tuning versus memory adapter training. Memory adapter training is slower than full fine-tuning because we train for three times as many epochs, though the per-step time is actually faster because almost all layers are frozen. We also list the runtimes of selected baseline unlearning algorithms. All baselines take at least a third as long as retraining the entire model, and some take even longer than the initial fine-tuning. Unlearning with memory adapters only requires updating the block-list, which requires no training.

Table 2. Runtime for initial fine-tuning and subsequent unlearning. Each iteration of memory adapter training is faster than full fine-tuning, but we train for more steps, so is slower overall. Unlearning using memory adapters is free. We report the maximum, median, and minimum runtimes for baseline unlearning algorithms.

Training Time (s)		Unlearning Time (s)	
Method	Runtime	Method	Runtime
Finetuned	859.3	MemAdapt	Free!
MemAdapt FT	1106.9	AltPO (max)	973.1
Retrained	790.1	UNDIAL (med)	373.1
		RMU (min)	261.0

Ablations. The unlearning performance of memory adapters is quite robust to variations in implementation; the main challenge is maintaining model utility while aiming to minimize added compute costs or scaling to larger training sets. In Appendix A we present additional results as we vary key hyperparameters of the memory adapter. We show how model utility degrades as we decrease the number of entries given to each data source and when the total size of the memory layer decreases.

5. Conclusion

Memory adapters are a new approach to fine-tuning which supports rapid, granular unlearning by design. By leveraging the sparsity and modularity of memory layers, we enable unlearning to be costless, flexible, and effective, with minimal losses in overall model performance and efficiency.

Future work should validate memory adapters on larger and more realistic datasets to assess how well they will scale to real-world scenarios. Developing new techniques for increasing their scalability, quantifying information leakage across data sources, and exploring models with pre-trained memory layers are all interesting areas for additional study.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Unlearning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

Copyright 2026 Carnegie Mellon University.

This material is based upon work supported by the Department of War under Air Force Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This work is licensed under CC BY-NC 4.0 (https://creativecommons.org/licenses/by-nc/4.0/).

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

References

Berges, V.-P., Oğuz, B., Haziza, D., Yih, W.-t., Zettlemoyer, L., and Ghosh, G. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

Bonta, R. California consumer privacy act (ccpa). *Retrieved from State of California Department of Justice: https://oag. ca. gov/privacy/ccpa*, pp. 4–40, 2022.

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C. A., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. Machine unlearning. In *2021 IEEE symposium on security and privacy (SP)*, pp. 141–159. IEEE, 2021.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Cloud, A., Goldman-Wetzler, J., Wybitul, E., Miller, J., and Turner, A. M. Gradient routing: Masking gradients to localize computation in neural networks. *arXiv preprint arXiv:2410.04332*, 2024.

Coble, H. Digital millennium copyright act. *105th U.S. Congress*, 1998.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Dong, Y. R., Lin, H., Belkin, M., Huerta, R., and Vulić, I. UNDIAL: Self-distillation with adjusted logits for robust unlearning in large language models. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8827–8840, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.444. URL https://aclanthology.org/2025.naacl-long.444/.

Dorna, V., Mekala, A. R., Zhao, W., McCallum, A., Kolter, J. Z., Lipton, Z. C., and Maini, P. Openunlearning: Accelerating LLM unlearning via unified benchmarking of methods and metrics. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL https://openreview.net/forum?id=Gy67Zh5X1i.

EU. General data protection regulation. *Official Journal of the European Union*, 2016.

Fan, C., Liu, J., Lin, L., Jia, J., Zhang, R., Mei, S., and Liu, S. Simplicity prevails: Rethinking negative preference optimization for LLM unlearning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=JbvSQm5h1l.

He, X. O. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2): 181–214, 1994. doi: 10.1162/neco.1994.6.2.181.

Kuo, K., Setlur, A., Srinivas, K., Raghunathan, A., and Smith, V. Exact unlearning of finetuning data via model merging at scale. *arXiv preprint arXiv:2504.04626*, 2025.

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., and Jégou, H. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Li, N., Pan, A., Gopal, A., Yue, S., Berrios, D., Gatti, A., Li, J. D., Dombrowski, A.-K., Goel, S., Mukobi, G., Helm-Burger, N., Lababidi, R., Justen, L., Liu, A. B., Chen, M., Barrass, I., Zhang, O., Zhu, X., Tamirisa, R., Bharathi, B., Herbert-Voss, A., Breuer, C. B., Zou, A., Mazeika, M., Wang, Z., Oswal, P., Lin, W., Hunt, A. A., Tienken-Harder, J., Shih, K. Y., Talley, K., Guan, J., Steneker, I., Campbell, D., Jokubaitis, B., Basart, S., Fitz, S., Kumaraguru, P., Karmakar, K. K., Tupakula, U., Varadharajan, V., Shoshitaishvili, Y., Ba, J., Esvelt, K. M., Wang, A., and Hendrycks, D. The WMDP benchmark: Measuring and reducing malicious use with unlearning. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=xlr6AUDuJz.

Lin, J., Zettlemoyer, L., Ghosh, G., Yih, W.-T., Markosyan, A., Berges, V.-P., and Oğuz, B. Continual learning via sparse memory finetuning. *arXiv preprint arXiv:2510.15103*, 2025.

Maini, P., Feng, Z., Schwarzschild, A., Lipton, Z. C., and Kolter, J. Z. TOFU: A task of fictitious unlearning for LLMs. In *First Conference on Language Modeling*, 2024.

Mekala, A., Dorna, V., Dubey, S., Lalwani, A., Koleczek, D., Rungta, M., Hasan, S., and Lobo, E. Alternate preference optimization for unlearning factual knowledge in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3732–3752, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.252/.

Nasr, M., Rando, J., Carlini, N., Hayase, J., Jagielski, M., Cooper, A. F., Ippolito, D., Choquette-Choo, C. A., Tramèr, F., and Lee, K. Scalable extraction of training data from aligned, production language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vjel3nWP2a.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Shi, W., Bhagia, A., Farhat, K., Muennighoff, N., Morrison, J., Walsh, E. P., Schwenk, D., Longpre, S., Poznanski, J., Ettinger, A., et al. Flexolmo: Open language models for flexible data use. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Shilov, I., Cloud, A., Gema, A. P., Goldman-Wetzler, J., Panickssery, N., Sleight, H., Jones, E., and Anil, C. Beyond data filtering: Knowledge localization for capability removal in llms. *arXiv preprint arXiv:2512.05648*, 2025.

Weston, J., Chopra, S., and Bordes, A. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Zhang, R., Lin, L., Bai, Y., and Mei, S. Negative preference optimization: From catastrophic collapse to effective unlearning. *First Conference on Language Modelling*, 2024. URL https://openreview.net/pdf?id=MXLBXjQkmb.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

A. Memory Adapter Ablations

We find that the unlearning performance of memory adapters is quite robust to variations in implementation, including the number of memory layers, whether the router is trainable or frozen, and the addition of gating or projection steps (as in Berges et al. (2024)). The main challenge is maintaining model utility while limiting the number of parameters needed for each data source, which is important for both minimizing compute costs and scaling to larger training sets. In the left column of Figure 2, we plot retain and general utility as a function of the number of memory entries assigned to each document (out of 1024^2 total values). We see that retain-set utility degrades rapidly with fewer than 128 values per document due to the smaller number of updatable parameters and fewer accesses to non-zero entries.

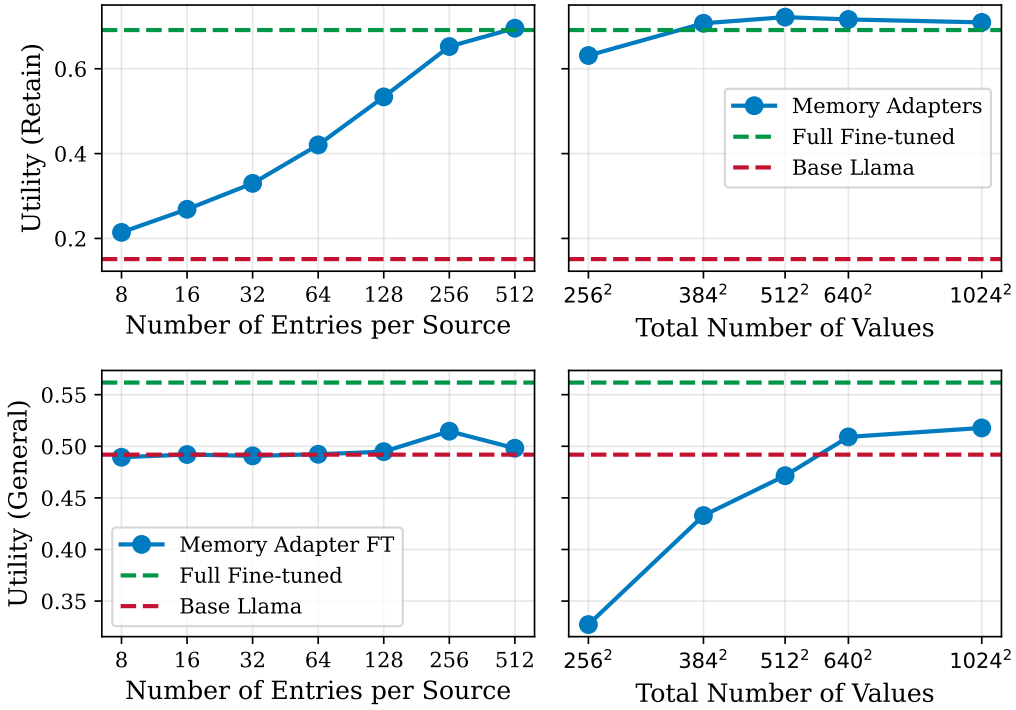

Figure 2. (Left column) Reducing the number of entries per data source increases the maximum allowable training set size, but reduces the model's ability to learn each document. (Right column) Reducing the total number of values reduces parameter count, but degrades general model performance due to task-specific entries being accessed more frequently in other contexts.

In the right column of Figure 2, we vary the total number of values from 256^2 to 1024^2, fixing 256 values per document. Here, general utility degrades as the total number of values is reduced and approaches the number of assigned entries. The total size of the memory layer has an important relationship with routing behavior and entry assignment. As the size increases, TF-IDF scores tend to increase, but for a particular document the accesses will be spread across a larger number of entries, reducing the number of accesses to assigned entries. Scaling memory size can therefore improve the specificity of router decisions, but documents may need more entries to be properly learnt. Future work should explore these tradeoffs further. Also note that larger layers have a large fraction of values frozen to zero; these do not actually need to be materialized, and could be removed and replaced with a step that reindexes accesses for the compressed form.