Layered Unlearning for Adversarial Relearning

Timothy Qian, Vinith Suriyakumar, Ashia Wilson, Dylan Hadfield-Menell
MIT
{tcqian,vinithms,ashia07,dylanhm}@mit.edu

Abstract

Unlearning in machine learning aims to remove specific information from models, often to protect privacy or eliminate harmful knowledge. However, current methods remain vulnerable to adversarial relearning, where partial re-exposure to sensitive data (e.g., a subset of social security numbers) causes models to recover previously unlearned information (e.g., the full set of social security numbers). To address this vulnerability, we introduce *Layered Unlearning*, a general framework that divides data into folds and unlearns them sequentially. This process creates directional barriers that prevent relearning on later subsets from compromising earlier ones, and can be composed with arbitrary unlearning algorithms to enhance their robustness. We motivate Layered Unlearning through synthetic experiments that serve as a testbed for future mechanistic investigations. We then evaluate our approach on the Years, WMDP, and MMLU datasets, demonstrating significantly improved resistance to adversarial relearning attacks. Our experiments also identify a more effective variant of adversarial relearning that finetunes directly on corpus data rather than multiple-choice prompts. Together, our contributions improve on state-of-the-art defenses and attacks, while offering new insight into the dynamics of post-training models.

1 Introduction

Machine unlearning—selectively removing specific data from trained models—has become increasingly important due to growing privacy, ethical, and legal concerns surrounding large language models (LLMs). As LLMs are increasingly deployed in high-stakes settings such as healthcare (He et al., 2025) and finance (Nie et al., 2024), their privacy and security risks have become urgent research challenges (Carlini et al., 2022; Das et al., 2024; Kandpal et al., 2022; Neel & Chang, 2023). One solution proposed by the research community is *machine unlearning* (Cao & Yang, 2015; Suriyakumar & Wilson, 2022; Thudi et al., 2022; Kurmanji et al., 2024; Chen & Yang, 2023), which aims to remove specific data (or concepts) from trained models for legal, ethical, or safety reasons.

Ideally, one would ensure forgetting by retraining from scratch without the sensitive data. Unfortunately, for many models such as LLMs, this is economically infeasible. Thus, a great effort has been put into building algorithms that approximate retraining (Zhang et al., 2024b; Chen & Yang, 2023; Pawelczyk et al., 2023; Li et al., 2024a; Maini et al., 2024). A requirement of these algorithms, especially for open-weight LLMs, is to ensure that unlearned information remains truly erased—or at least difficult to recover. However, this remains an open problem. Recent studies show that current unlearning techniques are fragile: partial re-exposure to unlearned data often leads to full recovery, revealing that models tend to suppress rather than erase learned information (Greenblatt et al., 2024; Deeb & Roger, 2025; Che et al., 2024; Suriyakumar et al., 2024).

Synthetic experiment code at: https://anonymous.4open.science/r/layered-unlearning-7AB7/

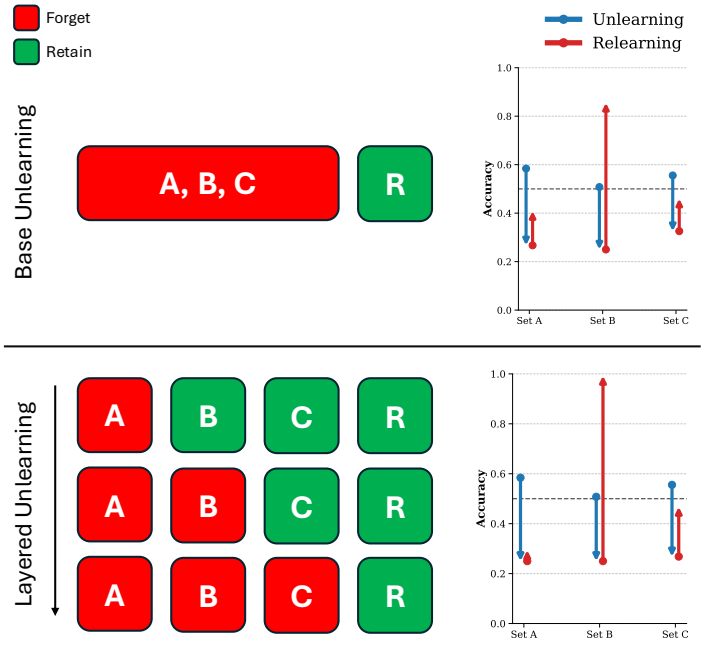

Figure 1: Left: An illustration of Layered Unlearning for three data folds (A, B, C) with a retain set R. Standard unlearning (top) jointly unlearns all three data folds together. In contrast, Layered Unlearning (bottom) unlearns each fold in sequence while retaining the tail of the sequence. Right: The performance on each data fold after unlearning (blue) and the performance after relearning B (red). For vanilla unlearning (top), relearning B improves performance on A and C. In contrast, relearning B after Layered Unlearning improves performance on C but not A. This illustrates how Layered Unlearning introduces a one-way barrier for relearning attacks.

In particular, these vulnerabilities are exposed through *adversarial relearning* attacks. Here, an attacker has white-box access to the model and partial exposure to the unlearned data, and attempts to recover forgotten knowledge by finetuning or leveraging model outputs (Che et al., 2024). Despite various proposed defenses, adversarial relearning remains a largely unsolved challenge.

We introduce *Layered Unlearning (LU)*, a sequential unlearning method that improves robustness to these relearning attacks. LU works by dividing the unlearned dataset into folds and applying unlearning incrementally, one fold at a time. At each step, the model forgets a new subset of data while retaining the remainder. This process introduces directional barriers: data forgotten in later folds cannot be used easily to recover data from earlier folds. LU is model-agnostic and can augment existing unlearning algorithms.

We evaluate LU across multiple datasets and attack strategies, showing that it offers stronger resistance to relearning than existing approaches, and that the order in which data is unlearned creates asymmetric vulnerabilities. Our experiments use a threat model where an adversary has white-box access to the model and partial exposure to previously unlearned data. The attacker finetunes the model on this subset to attempt recovery of additional information. LU reduces the effectiveness of this strategy. Specifically, we find that finetuning on fold i fails to recover information from previously unlearned folds $1, \ldots, i-1$, creating a systematic barrier to knowledge reconstruction.

In the course of evaluating LU, we also identify a stronger class of attack: *corpus-based finetuning*. This variant, which trains on corpus text rather than multiple-choice prompts, proves more effective against LU than prior attacks. We extend the RTT attack framework from Deeb & Roger (2025) and show that corpus-based finetuning is a stronger form of

adversarial relearning. Notably, this difference in attack effectiveness only emerges because LU creates meaningful variation in robustness across data subsets. As such, LU not only improves unlearning but also reveals new vulnerabilities—thereby advancing the state of evaluation.

Together, these contributions—a new method, a comprehensive evaluation, and a stronger attack—advance the understanding and practice of machine unlearning. More broadly, our results support the view that post-training reshapes model behavior not through erasure but through suppression and reprioritization (Che et al., 2024). We hypothesize that LU operates by activating distinct suppression mechanisms—what we refer to as *inhibitors*—within the model. To clarify this mechanism, we begin our analysis with synthetic experiments using a linear model trained for binary classification a one-layer attention-only transformer trained on toy data. These interpretable settings illustrates how relearning reactivates shared suppression and how LU avoids this by forcing the model to use more precise, independent inhibitors. This framing provides a conceptual link between LU's robustness and broader questions about how post-training modifies internal representations.

Contributions. Our key contributions are:

- **Layered Unlearning via sequential data partitioning:** We propose a novel k-fold unlearning framework, called *Layered Unlearning (LU)*. This framework partitions data into sequential folds and applies existing unlearning algorithms over these folds sequentially.
- **Investigating the mechanism of improved robustness:** We use both a logistic regression model for binary classification and a simple attention-only transformer trained on synthetic data to study how LU modifies internal representations. These controlled settings offer a mechanistic testbed for interpreting the source of LU's robustness and support future work in mechanistic interpretability.
- **Improved robustness against adversarial relearning:** We empirically demonstrate that LU improves resistance to relearning attacks. Finetuning on fold i fails to recover information from folds $1, \ldots, i-1$, introducing a one-way barrier that is absent in conventional joint unlearning.
- **Stronger adversarial relearning with corpus-based finetuning:** Through LU, we uncover a stronger RTT attack based on corpus fine-tuning compared to MCQ-based finetuning. Our comprehensive evaluation across datasets (Years, WMDP, MMLU) reveals nuanced vulnerabilities in existing unlearning techniques.

2 Background and related work

Unlearning for LLMs. Machine unlearning for large language models (LLMs) has become an active area of research (Lu et al., 2022; Jang et al., 2022; Kumar et al., 2022; Zhang et al., 2023; Pawelczyk et al., 2023; Eldan & Russinovich, 2023; Ishibashi & Shimodaira, 2023; Yao et al., 2023; Maini et al., 2024; Zhang et al., 2024b; Li et al., 2024b; Wang et al., 2024; Jia et al., 2024; Liu et al., 2024b;a; Thaker et al., 2024; Kadhe et al., 2024; Fan et al., 2025). Due to the difficulty of exact unlearning, most existing methods adopt approximate strategies, including model optimization (Ilharco et al., 2022; Liu et al., 2022; Yao et al., 2023; Eldan & Russinovich, 2023; Jia et al., 2024; Zhang et al., 2024b; Li et al., 2024b) and prompt-based or in-context learning techniques (Thaker et al., 2024; Pawelczyk et al., 2023; Liu et al., 2024a). However, recent work has shown that these models often remain vulnerable to adversarial attacks (Schwarzschild et al., 2024; Patil et al., 2024; Lynch et al., 2024) or to relearning from small fragments of previously seen data (Hu et al., 2024; Lynch et al., 2024). These findings highlight the persistent challenges in achieving robust unlearning in LLMs.

Adversarial relearning. Adversarial relearning exploits the model's residual knowledge after unlearning by finetuning on a small subset of forgotten data, with the goal of recovering information about the entire unlearned set. Che et al. (2024) showed that nearly all existing unlearning methods are vulnerable to this attack, exposing a fundamental weakness in

current approaches. Deeb & Roger (2025) further showed that even informationally distinct examples can induce relearning, suggesting failures that extend beyond direct memorization. While defenses have been proposed (Rosati et al., 2024; Zou et al., 2024; Tamirisa et al., 2025), none have proven consistently effective under adversarial relearning (Che et al., 2024). Our method, similar to latent adversarial training (LAT) (Sheshadri et al., 2025), introduces techniques that bootstrap on top of any existing unlearning methods. (Deeb & Roger, 2025) only finetunes on MCQ as that guarantees performance on the relearned set, but finetuning on corpus is a more natural form of adversarial relearning. Finetuning on corpus (Che et al., 2024) and MCQ (Deeb & Roger, 2025) have both been studied, but a comprehensive comparison of them has not.

Adversarial relearning. Adversarial relearning attacks exploit residual knowledge after unlearning by finetuning on a small subset of forgotten data, aiming to recover information about the full unlearned set. Che et al. (2024) showed that most existing unlearning methods are vulnerable to such attacks, revealing a fundamental limitation. Deeb & Roger (2025) further demonstrated that even informationally distinct examples can induce relearning, indicating failures beyond rote memorization. While several defenses have been proposed (Rosati et al., 2024; Zou et al., 2024; Tamirisa et al., 2025), none have consistently withstood adversarial relearning (Che et al., 2024). Our method, like latent adversarial training (LAT) (Sheshadri et al., 2025), augments existing unlearning algorithms to improve robustness.

Prior work has studied both corpus-based (Che et al., 2024) and MCQ (Deeb & Roger, 2025) finetuning. To our knowledge, no comprehensive comparison of the two strategies has been conducted; we find corpus-based finetuning to be a more natural and effective form of adversarial relearning.

Sequential unlearning. Sequential unlearning has been explored in various contexts, such as removing copyrighted information over time (Dou et al., 2025). Zhao et al. (2024) studied sequential unlearning as a means to improve forgetting efficiency but did not consider its impact on robustness against adversarial relearning. In contrast, our work investigates how the order and structure of sequential unlearning influence robustness, focusing on its potential to mitigate adversarial relearning. Specifically, we analyze the path dependence of unlearning and propose a novel framework that leverages structured forgetting to enhance resilience against information leakage.

3 Motivating Layered Unlearning

We propose a preliminary intuition to motivate the design of Layered Unlearning.

Consider the scenario where we attempt to unlearn two data folds, A and B. We introduce the concept of *inhibitors*, denoted as I_A, I_B, and I_{AB}, where an active inhibitor limits the model's performance on its corresponding folds. We hypothesize that I_{AB}, which jointly suppresses performance on both A and B, is easier to activate than the individual inhibitors I_A and I_B. Broadly, some of the reasons these inhibitors arise could be: localization of knowledge in different parts of the LLMs, inductive biases of optimizers such as SGD or Adam, or optimizing the average loss over the folds. We encourage future work to investigate these mechanisms.

When unlearning A and B jointly, the optimizer naturally activates I_{AB}, as it provides the simplest way to suppress both folds. However, if we later attempt to relearn either A or B, this deactivates I_{AB}, restoring performance on both sets simultaneously.

Ideally, we would like an algorithm that activates I_A and I_B separately. We hypothesize that this happens when applying sequential unlearning over the two data tolds, which we call Layered Unlearning. In the first stage, when we unlearn A while retaining B, activating I_{AB} is not an option, as it would degrade performance on B. Instead, the optimizer is forced to activate I_A, which selectively inhibits performance on A while preserving B. In the second stage, when we unlearn both A and B, either I_{AB} or I_B is activated.

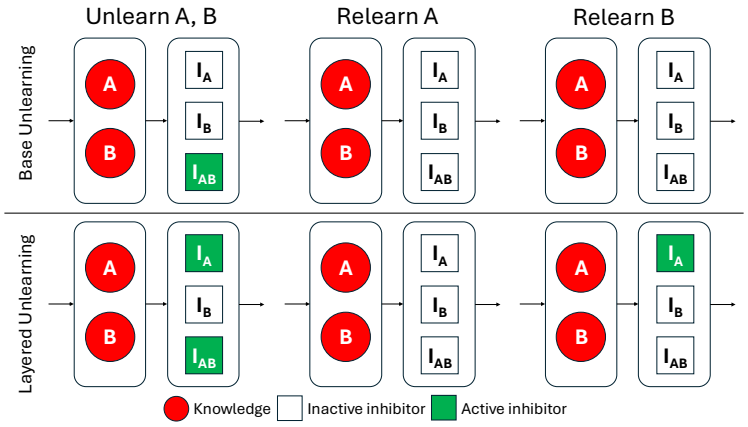

Figure 2: Effect of relearning with inhibitors. Relearning fold B keeps inhibitor I_A active, preventing recovery on A. In contrast, relearning fold A deactivates I_{AB} and I_A, improving performance on both A and B.

The key distinction emerges during relearning:

- Relearning on B only deactivates I_{AB} or I_B, leaving I_A intact and preserving the forgetting of A.
- Relearning on A deactivates both I_{AB} and I_A, restoring performance on both A and B. If I_B was activated during the second unlearning step, then the barrier will be bidirectional.

This suggests that Layered Unlearning would impose additional constraints, forcing the model to rely on more granular inhibitors (I_A, I_B) rather than a single joint inhibitor (I_{AB}). As a result, earlier folds in the unlearning process become more resistant to adversarial relearning from later folds.

4 Methods

Here we present the concrete version of our algorithm, Layered Unlearning (LU) based on previous intuition. Consider a dataset F that we wish to unlearn. For the purposes of our study, we will focus our experiments on settings where F is a multiple-choice question (MCQ) task. We partition F into k random disjoint subsets of equal size, $F_1, \ldots, F_k$. One could also partition the dataset based on adversarial risk, where F_k contains data most likely to be discovered by an adversary, and F_1 contains data least likely to be discovered. We leave the exploration of this partitioning to future work.

At a high level, we perform unlearning in k sequential steps. At step i, we unlearn $F_1, \ldots, F_i$ while retaining $F_{i+1}, \ldots, F_k$. The intuition is that this should result in easier independent control of the model's performance on each of these sets. This formulation will ideally progressively separate the model's performance on each fold at each stage. In comparison, existing techniques will combine all of the data in each fold together and apply unlearning all at once. While both can sufficiently unlearn, our intuition is that this sequential process will produce a more robust solution by forcing the model to learn more independent representations.

At each stage of unlearning, we train the model until it sufficiently forgets the information from F_i while retaining the information from $F_{i+1}, \ldots, F_k$. This is encoded explicitly through the forget and retain losses in the unlearning algorithm of choice. For MCQ, we expect a successful unlearning algorithm to achieve roughly 25% accuracy on the validation set for F_i, corresponding to random guessing. Additionally, we aim to maintain low accuracy on previously forgotten sets $F_1, \ldots, F_{i-1}$ to prevent relearning. For a graphical depiction of this

process, see Appendix Figure 6. In practice, since these algorithms approximate unlearning, we choose the sufficient unlearning threshold to be 35%.

Our approach is broadly applicable to any unlearning method that defines both a forget set and a retain set, encompassing most existing unlearning techniques. A detailed description of our algorithm is provided in Appendix Algorithm 1. For clarity, we denote *Algorithm*-LU as an unlearning algorithm augmented with Layered Unlearning. For instance, RMU with Layered Unlearning is referred to as RMU-LU.

Why not apply sequential unlearning directly? We observe that forgetting F_1 often leads to forgetting $F_2, \ldots, F_k$ as well. This effect suggests that unlearning can propagate beyond the intended target, making sequential unlearning unreliable for certain scenarios (Zhang et al., 2024a). However, this may not hold if the sets F_i are sufficiently distinct, and we leave this to future work. But this is a subset of our method, and as long as the forget accuracies of the forget sets remain low and the retain accuracies of the retain sets remain high at each stage, the coefficients on the losses in theory can be set to 0.

Why must previously forgotten sets be retained? We observe in our experiments that after forgetting a set F_i and advancing to the next unlearning stage, the model may sometimes regain knowledge of F_i unless it remains explicitly included in the forget set. This suggests that unlearning is not inherently persistent, as previously forgotten information can resurface unless continuously suppressed. However, this effect is inconsistent across different settings.

5 Synthetic experiments for understanding Layered Unlearning.

To empirically validate our intuition, we construct two synthetic experiments: one using logistic regression for binary classification, and another using a small, attention-only, one-layer transformer for bigram modeling. Main results are presented here; ablations are in Appendix B.2.

5.1 Binary classification

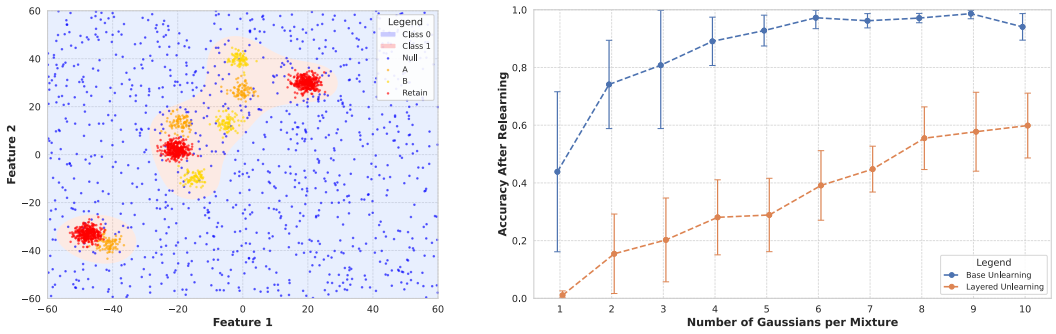

Figure 3: Left: Visualization of the experimental setup. Right: Accuracy after relearning on B for base unlearning and Layered Unlearning as a function of the number of Gaussians per mixture. Error bars indicate 2-std confidence intervals over 10 random seeds.

We begin with a binary classification task using logistic regression, a well-understood and interpretable setting. While simpler than our later experiments, it provides a useful foundation.

The input data lies in a 2D space. We sample from a uniform distribution over a rectangular region to form the Null dataset (class 0). We then generate three datasets, A, B, and C, each composed of a mixture of Gaussians with randomly chosen components, labeled as class 1. See Figure 3

Task A and Task B refer to classifying datasets A and B as class 1, respectively. The retain task is to classify Null as class 0 and C as class 1. Unlearning Task A requires classifying dataset A as class 0; similarly for Unlearning Task B.

We train a logistic regression model with radial basis function (RBF) features on the combined datasets.

As shown in Figure 3, Layered Unlearning exhibits greater robustness to adversarial relearning in this synthetic setup. This holds even as we increase the number of Gaussian components Figure 3. Notably, when the Gaussian means are generated completely at random, base unlearning is less robust compared to when the components are clustered to make A, B, and C more distinct. We hypothesize that clustering improves robustness by increasing separation between datasets, making adversarial relearning more difficult.

5.2 Bigram modeling

We next consider a more complex prediction task: bigram modeling with a small, attention-only, one-layer transformer. These networks are simple enough to be analytically studied (Elhage et al., 2021), yet still serve as a useful proxy for the larger architectures used in our later experiments.

We model a simple language with three tokens, a, b, c, where transitions are governed by bigram statistics. The initial transition matrix is:

$$\begin{bmatrix} \epsilon & \epsilon & 1-2\epsilon \\ \epsilon & \epsilon & 1-2\epsilon \\ \frac{1-\epsilon}{2} & \frac{1-\epsilon}{2} & \epsilon \end{bmatrix},$$

where $\epsilon = 0.05$ ensures smooth learning dynamics. Intuitively, a and b strongly transition to c, while c transitions to a and b with equal probability. We define three tasks: (1) Task A, where a always transitions to c; (2) Task B, where b always transitions to c; and (3) the retain task, where c transitions to a and b with equal probability.

Unlearning Task A modifies the transition probabilities so that a transitions uniformly to a, b, c, with Task B defined analogously. To evaluate unlearning effectiveness for Task A, we generate sequences of a, b, c uniformly at random and measure the probability of c appearing after a. A higher probability indicates stronger retention of the undesired behavior. Task B is evaluated similarly by tracking transitions from b. The retain set is assessed using the total variation (TV) distance from the original expected transition distribution of c.

To understand how the model responds to unlearning, we introduce a potential shared *inhibitor* across Tasks A and B, which encourages uniform predictions. However, to prevent the model from trivially collapsing into uniform randomness, we structure the retain task to maintain its original behavior, ensuring $c \to \{a, b\}$ with equal probability. This setup forces the model to balance unlearning constraints while preserving meaningful structure.

Training and evaluation follow a simple procedure. We train the model using sequences generated from the target bigram matrix and optimize with a standard language modeling loss. For relearning, we generate sequences where the bigram matrix is uniform at random except for the task being relearned. During evaluation, we compute the loss only on occurrences of a when relearning Task A and analogously for Task B.

As shown in Table 1, our synthetic setup reproduces the Layered Unlearning effect. Retain set performance remains largely stable under relearning, indicating that the transformer does not revert to uniform predictions, but selectively inhibits performance on Tasks A and B in an asymmetric way. This effect does not appear in a zero-layer, embedding-only transformer. Ablation experiments further confirm the importance of attention (see Appendix B.2), suggesting a key role for attention mechanisms in resisting adversarial relearning. We release code and data to support future work on this phenomenon.

Method	Relearn	A ↓	B ↓	Retain ↓
Initial	—	0.92	0.92	**0.01**
Base	—	0.34	0.34	0.03
Base-LU	—	**0.32**	**0.33**	**0.01**
Base	A	—	0.75	**0.04**
Base-LU	A	—	**0.53**	0.06
Base	B	0.80	—	**0.03**
Base-LU	B	**0.50**	—	0.05

Table 1: Results from synthetic experiments. The first section presents the initial model trained on the original data alongside models unlearned using both the standard and Layered Unlearning methods. The next two sections show results from adversarial relearning attacks. Retain accuracies are measured using total variation (TV) distance, where lower values indicate better retention of the desired distribution.

6 Main experiments

Datasets. We utilize the *Years*, *WMDP*, and *MMLU* datasets, sourced from Deeb & Roger (2025); Hendrycks et al. (2021); Li et al. (2024a). The validation set for each dataset consists of multiple-choice questions (MCQs), with corresponding corpus data generated using GPT-4o (OpenAI, 2024; Deeb & Roger, 2025). Each MCQ has four answer choices, meaning random guessing yields a baseline accuracy of 25%.

For the retain set, we use the *MMLU* dataset (Hendrycks et al., 2021). The corresponding retain corpus is sourced from *FineWeb* (Penedo et al., 2024). When unlearning portions of MMLU, we designate specific categories as the forget set while retaining the remaining categories as in Deeb & Roger (2025).

Evaluation. Given a dataset F to be unlearned, we uniformly split it into k folds, $F_1, \ldots, F_k$. For a fixed k and dataset, this partitioning remains consistent across all experiments for both unlearning and evaluation. We follow the Language Model Evaluation Harness standards for 0-shot (Gao et al., 2023).

To evaluate a model $\mathcal{M}$, we consider all $2^k - 2$ proper subsets $S \subset \{F_1, \ldots, F_k\}$ and follow the evaluation protocol in Deeb & Roger (2025). Specifically, we finetune $\mathcal{M}$ on either the MCQ data or the corresponding corpus from S and then evaluate it on the MCQ questions from $T := F \setminus S$. Model performance is tracked over epochs, and we report the highest accuracy on T across epochs as the final forget accuracy. All finetuning experiments use the Adam optimizer (Kingma & Ba, 2017).

Unlearning. Current LLM unlearning methods fall into two main categories: representation engineering and gradient ascent. We evaluate our approach using one representative algorithm from each—*Representation Misdirection Unlearning (RMU)* (Li et al., 2024a) and *SimNPO* (Fan et al., 2025), respectively. Specifically, we analyze two RMU variants: RMU-LAT and RMU-Split. In RMU-Split, each fold is projected onto a different random vector, in contrast to the shared projection used in RMU. This isolates the impact of Layered Unlearning from the confounding effect of using separate random vectors.

For a standardized comparison across different unlearning algorithms, we define the *recover rate*. Consider two resulting models, U and V, after unlearning. We aim to compare the percentage of information recovered by adversarial relearning on each model. We define the recover rate as:

$$\text{Recover Rate} = \frac{\text{U Accuracy After Relearning} - \text{U Accuracy After Unlearning}}{\text{V Accuracy After Relearning} - \text{V Accuracy After Unlearning}}.$$

This metric quantifies how much more vulnerable model U is to relearning compared to model V, with a lower recover rate indicating better unlearning robustness.

To ensure fair comparisons, we impose a minimum unlearning accuracy threshold of 0.25 (random guessing for MCQs) in cases where the model is highly unlearned. This prevents artificially inflating the recover rate when the model has already reached the theoretical lower bound of performance.

To ensure model utility, we only consider unlearned models that experience at most a 10% accuracy drop on the retain set, see Appendix Table 5. All unlearning experiments are conducted using *Zephyr-7B-β* (Tunstall et al., 2023).

6.1 Layered Unlearning is more robust to adversarial relearning

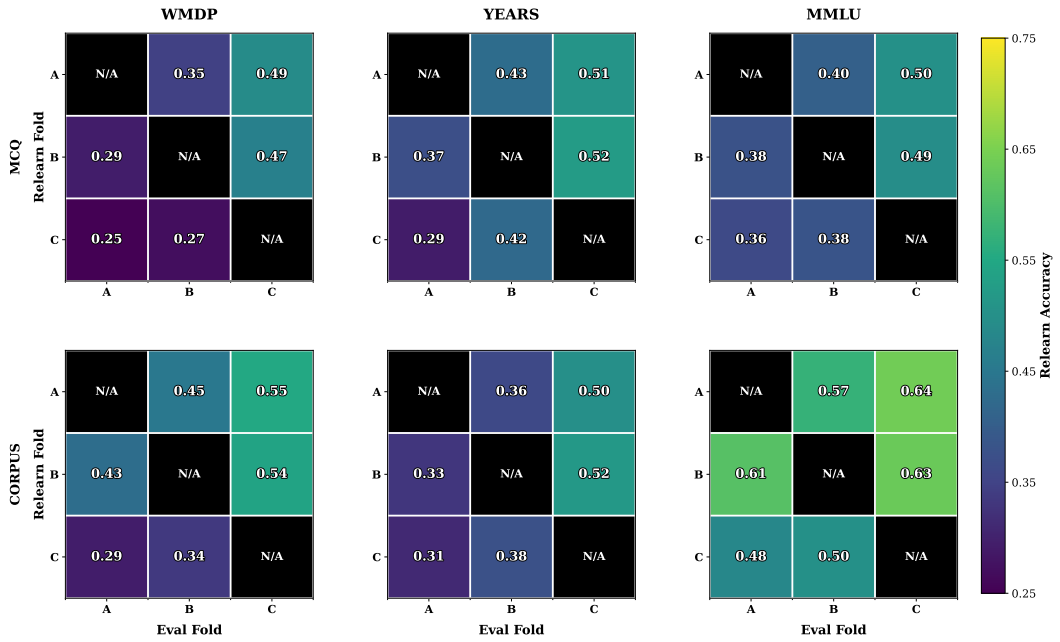

Figure 4: Relearn accuracy heatmaps across three datasets (WMDP, Years, MMLU). Each cell represents the relearning accuracy when training on one fold (y-axis) and evaluating on another (x-axis). We have relearning on MCQ in the first row and relearning on corpus in the second row. In all cases, values below the main diagonal are lower than values above the main diagonal, demonstrating the one-way barrier to relearning.

Throughout our experiments, we find that RMU is more robust to adversarial relearning when augmented with Layered Unlearning (Figure 4). The most striking finding here is that across datasets, this robustness is influenced by the order of folds. Specifically, an adversary with access to fold A can recover more information about folds B and C compared to an adversary with access only to fold C. Layered Unlearning demonstrates a path-dependent phenomenon about robustness to adversarial relearning.

Consider the case where we unlearn two sets, A and B, with $k = 2$. If unlearning were a path-independent operation, then sequentially unlearning A while retaining B, followed by unlearning B, should yield the same result as unlearning A and B simultaneously. We exploit path-dependence to strategically design the order of forgetting to enhance robustness against adversarial relearning, leveraging the fact that different sequences of unlearning can lead to different retention and generalization behaviors in the final model.

Compared to other tailored defenses to adversarial relearning, such as latent adversarial training (LAT) (Sheshadri et al., 2025), we find LU is consistently more robust for RMU (Table 2) when the relearned fold is unlearned after the evaluated fold. Otherwise, it performs comparably.

Does the choice of Layered Unlearning base algorithm matter? We also evaluate *SimNPO* as a base method for Layered Unlearning and find that the robustness previously observed with LU largely carries over (Appendix Table 13). However, the unidirectional barrier property appears weaker. As shown in Appendix Table 8, SimNPO consistently exhibits higher raw relearning accuracy—both in its base form and when combined with LU—indicating greater overall vulnerability to adversarial relearning.

Interestingly, Layered Unlearning is more effective against corpus-based attacks when applied to SimNPO than to RMU, as reflected in the lower recover rates. Nonetheless, SimNPO-LU still shows higher raw relearning accuracy than RMU, suggesting lingering susceptibility. Moreover, recover rates for SimNPO-LU suggest that its barrier effect may be more symmetric compared to the directional robustness observed with RMU.

6.2 Corpus finetuning is a stronger adversarial attack

We further investigate when Layered Unlearning becomes less robust to adversarial relearning by replacing MCQ-style prompts with corpus-based finetuning. In doing so, we uncover a new state-of-the-art attack because of LU's robustness. While some robustness to relearning remains, it is significantly weaker, as shown in Figure 4 and Table 2. This suggests that corpus-based finetuning is a stronger attack for recovering forgotten knowledge compared to targeted question-based finetuning.

These findings highlight the importance of developing unlearning techniques that can withstand a diverse range of relearning attacks. We speculate that corpus-based finetuning is particularly effective against unlearning methods that better preserve the model's general language modeling capabilities, though this remains an open question for future research.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.41	0.45	0.45	**0.49**
A	RMU-LAT	—	—	0.39	**0.44**	0.48	0.55
A	RMU-Split	—	—	0.37	**0.44**	**0.38**	0.54
A	RMU-LU	—	—	0.40	0.49	0.50	0.54
A	RMU-Split-LU	—	—	**0.35**	0.45	0.49	0.55
B	RMU	0.41	0.48	—	—	0.46	**0.48**
B	RMU-LAT	0.40	0.48	—	—	0.44	**0.48**
B	RMU-Split	0.42	0.53	—	—	**0.38**	0.54
B	RMU-LU	0.31	0.44	—	—	0.50	0.54
B	RMU-Split-LU	**0.29**	**0.43**	—	—	0.47	0.54
C	RMU	0.43	0.50	0.39	0.44	—	—
C	RMU-LAT	0.39	0.50	0.42	0.43	—	—
C	RMU-Split	0.40	0.55	0.39	0.45	—	—
C	RMU-LU	0.26	0.36	0.34	0.42	—	—
C	RMU-Split-LU	**0.25**	**0.29**	**0.27**	**0.34**	—	—

Table 2: Relearning accuracies on WMDP with 3 folds for RMU variants. Each section corresponds to the fold being relearned, with evaluation performed separately on all three folds. Finetuning on corpus leads to greater recovery compared to finetuning on MCQ.

7 Limitations and Future Work

In practice, Layered Unlearning is highly sensitive to hyperparameters. The coefficients for different loss terms must be carefully tuned at each stage, making it challenging to balance forgetting and maintaining performance on the retain set. This sensitivity suggests an inherent difficulty in disentangling representations associated with forgotten and retained data. Additionally, as the number of folds increases, the model effectively needs to "classify"

between each pair of folds, requiring it to learn $\binom{k}{2}$ classifiers. As a result, extending this algorithm to n folds, where n is the number of data points, becomes impractical.

Analyzing attacks on Layered Unlearning is difficult due to the combinatorial explosion in possible configurations needed to properly assess adversarial robustness. To comprehensively evaluate Layered Unlearning, we must test attacks on every subset of possible relearned folds. While further experiments could explore mixtures of different folds, this introduces at least $2^k - 2$ proper subsets, making exhaustive evaluation infeasible.

Sweeping over additional attack configurations—such as batch size, learning rate, dataset, and unlearning algorithm—further amplifies the computational cost. Given these challenges, we leave an extensive adversarial analysis to future work. However, we found that our initial hyperparameter settings were sufficient to break all base unlearning algorithms (RMU, RMU-LAT, SimNPO), so we kept them constant for fair comparisons. A detailed description of these hyperparameters is provided in the Appendix.

We do not directly address the challenge of harmless finetuning, where the attacker uses data unrelated to what was unlearned, but we offer an intuition to guide future work. In standard unlearning, relearning is significantly easier when the attacker has access to data that is similar to the unlearned examples. Even small amounts of related data can serve as powerful signals, making it surprisingly effective to recover forgotten information. We hypothesize that Layered Unlearning reduces this vulnerability by making recovery difficult even when related data is available. As a result, it shrinks the performance gap between finetuning with related versus unrelated data, potentially making both equally ineffective.

Finally, we also consider how the structured nature of Layered Unlearning might inform alignment. Betley et al. (2025) show that finetuning on one task—such as insecure code—can unintentionally induce harmful behaviors, revealing entanglement between seemingly unrelated capabilities and values. Our work focuses on unlearning as a way to disentangle such behaviors, potentially by implicit "orthogonalization" them, akin to Gram-Schmidt. This suggests a possible alignment strategy: first train a model to be harmless but helpless, then finetune it to be helpful while preserving harmlessness. In this setup, changing helpfulness would not affect harmfulness, but increasing harmfulness would reduce helpfulness.

8 Conclusion

We investigate adversarial relearning in LLM unlearning and propose Layered Unlearning, a k-fold sequential framework that improves robustness. Our findings show that unlearning order affects vulnerability, underscoring the importance of structured forgetting. We also demonstrate that corpus-based finetuning is a more effective attack than MCQ-based finetuning. Finally, we introduce a synthetic testbed to support future mechanistic investigations of Layered Unlearning. These results highlight the inherent difficulty of achieving persistent forgetting and the need for new strategies to strengthen unlearning.

References

Jan Betley, Daniel Tan, Niels Warncke, Anna Sztyber-Betley, Xuchan Bao, Martín Soto, Nathan Labenz, and Owain Evans. Emergent misalignment: Narrow finetuning can produce broadly misaligned llms, 2025. URL https://arxiv.org/abs/2502.17424.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pp. 463–480. IEEE, 2015.

Nicholas Carlini, Matthew Jagielski, Chiyuan Zhang, Nicolas Papernot, Andreas Terzis, and Florian Tramer. The privacy onion effect: Memorization is relative, 2022. URL https://arxiv.org/abs/2206.10469.

Zora Che, Stephen Casper, Anirudh Satheesh, Rohit Gandikota, Domenic Rosati, Stewart Slocum, Lev E McKinney, Zichu Wu, Zikui Cai, Bilal Chughtai, Furong Huang, and Dylan Hadfield-Menell. Model manipulation attacks enable more rigorous evaluations

of LLM unlearning. In *Neurips Safe Generative AI Workshop 2024*, 2024. URL https://openreview.net/forum?id=XmvgWEjkhG.

Jiaao Chen and Diyi Yang. Unlearn what you want to forget: Efficient unlearning for llms. *arXiv preprint arXiv:2310.20150*, 2023.

Badhan Chandra Das, Zhi Liu, and Qi Zhang. Security and privacy challenges of large language models: A survey. *arXiv preprint arXiv:2402.00888*, 2024. URL https://arxiv.org/abs/2402.00888. Accessed: 2025-03-24.

Aghyad Deeb and Fabien Roger. Do unlearning methods remove information from language model weights?, 2025. URL https://arxiv.org/abs/2410.08827.

Guangyao Dou, Zheyuan Liu, Qing Lyu, Kaize Ding, and Eric Wong. Avoiding copyright infringement via large language model unlearning, 2025. URL https://arxiv.org/abs/2406.10952.

Ronen Eldan and Mark Russinovich. Who's harry potter? approximate unlearning in llms, 2023.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Chongyu Fan, Jiancheng Liu, Licong Lin, Jinghan Jia, Ruiqi Zhang, Song Mei, and Sijia Liu. Simplicity prevails: Rethinking negative preference optimization for llm unlearning, 2025. URL https://arxiv.org/abs/2410.07163.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Ryan Greenblatt, Fabien Roger, Dmitrii Krasheninnikov, and David Krueger. Stress-testing capability elicitation with password-locked models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=zzOOqD6R1b.

Kai He, Rui Mao, Qika Lin, Yucheng Ruan, Xiang Lan, Mengling Feng, and Erik Cambria. A survey of large language models for healthcare: from data, technology, and applications to accountability and ethics, 2025. URL https://arxiv.org/abs/2310.05694.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Shengyuan Hu, Yiwei Fu, Zhiwei Steven Wu, and Virginia Smith. Jogging the memory of unlearned model through targeted relearning attack. *arXiv preprint arXiv:2406.13356*, 2024.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Yoichi Ishibashi and Hidetoshi Shimodaira. Knowledge sanitization of large language models. *arXiv preprint arXiv:2309.11852*, 2023.

Joel Jang, Dongkeun Yoon, Sohee Yang, Sungmin Cha, Moontae Lee, Lajanugen Logeswaran, and Minjoon Seo. Knowledge unlearning for mitigating privacy risks in language models. *arXiv preprint arXiv:2210.01504*, 2022.

Jinghan Jia, Yihua Zhang, Yimeng Zhang, Jiancheng Liu, Bharat Runwal, James Diffenderfer, Bhavya Kailkhura, and Sijia Liu. Soul: Unlocking the power of second-order optimization for llm unlearning. *arXiv preprint arXiv:2404.18239*, 2024.

Swanand Ravindra Kadhe, Farhan Ahmed, Dennis Wei, Nathalie Baracaldo, and Inkit Padhi. Split, unlearn, merge: Leveraging data attributes for more effective unlearning in llms. *arXiv preprint arXiv:2406.11780*, 2024.

Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 10697–10707, 2022. URL https://proceedings.mlr.press/v162/kandpal22a.html. Accessed: 2025-03-24.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Vinayshekhar Bannihatti Kumar, Rashmi Gangadharaiah, and Dan Roth. Privacy adhering machine un-learning in nlp. *arXiv preprint arXiv:2212.09573*, 2022.

Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. *Advances in neural information processing systems*, 36, 2024.

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D. Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, Gabriel Mukobi, Nathan Helm-Burger, Rassin Lababidi, Lennart Justen, Andrew B. Liu, Michael Chen, Isabelle Barrass, Oliver Zhang, Xiaoyuan Zhu, Rishub Tamirisa, Bhrugu Bharathi, Adam Khoja, Zhenqi Zhao, Ariel Herbert-Voss, Cort B. Breuer, Samuel Marks, Oam Patel, Andy Zou, Mantas Mazeika, Zifan Wang, Palash Oswal, Weiran Lin, Adam A. Hunt, Justin Tienken-Harder, Kevin Y. Shih, Kemper Talley, John Guan, Russell Kaplan, Ian Steneker, David Campbell, Brad Jokubaitis, Alex Levinson, Jean Wang, William Qian, Kallol Krishna Karmakar, Steven Basart, Stephen Fitz, Mindy Levine, Ponnurangam Kumaraguru, Uday Tupakula, Vijay Varadharajan, Ruoyu Wang, Yan Shoshitaishvili, Jimmy Ba, Kevin M. Esvelt, Alexandr Wang, and Dan Hendrycks. The wmdp benchmark: Measuring and reducing malicious use with unlearning, 2024a. URL https://arxiv.org/abs/2403.03218.

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, et al. The wmdp benchmark: Measuring and reducing malicious use with unlearning. *arXiv preprint arXiv:2403.03218*, 2024b.

Bo Liu, Qiang Liu, and Peter Stone. Continual learning and private unlearning. In *Conference on Lifelong Learning Agents*, pp. 243–254. PMLR, 2022.

Chris Yuhao Liu, Yaxuan Wang, Jeffrey Flanigan, and Yang Liu. Large language model unlearning via embedding-corrupted prompts. *arXiv preprint arXiv:2406.07933*, 2024a.

Sijia Liu, Yuanshun Yao, Jinghan Jia, Stephen Casper, Nathalie Baracaldo, Peter Hase, Xiaojun Xu, Yuguang Yao, Hang Li, Kush R Varshney, et al. Rethinking machine unlearning for large language models. *arXiv preprint arXiv:2402.08787*, 2024b.

Ximing Lu, Sean Welleck, Jack Hessel, Liwei Jiang, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning. *Advances in neural information processing systems*, 35:27591–27609, 2022.

Aengus Lynch, Phillip Guo, Aidan Ewart, Stephen Casper, and Dylan Hadfield-Menell. Eight methods to evaluate robust unlearning in llms. *arXiv preprint arXiv:2402.16835*, 2024.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. Tofu: A task of fictitious unlearning for llms, 2024.

Seth Neel and Peter Chang. Privacy issues in large language models: A survey. *arXiv preprint arXiv:2312.06717*, 2023. URL https://arxiv.org/abs/2312.06717. Accessed: 2025-03-24.

Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M. Mulvey, H. Vincent Poor, Qingsong Wen, and Stefan Zohren. A survey of large language models for financial applications: Progress, prospects and challenges, 2024. URL https://arxiv.org/abs/2406.11903.

OpenAI. Hello gpt-4o, 2024. URL https://openai.com/index/hello-gpt-4o/. Accessed: 2025-03-04.

Vaidehi Patil, Peter Hase, and Mohit Bansal. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. *ICLR*, 2024.

Martin Pawelczyk, Seth Neel, and Himabindu Lakkaraju. In-context unlearning: Language models as few shot unlearners. *arXiv preprint arXiv:2310.07579*, 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Domenic Rosati, Jan Wehner, Kai Williams, Łukasz Bartoszcze, David Atanasov, Robie Gonzales, Subhabrata Majumdar, Carsten Maple, Hassan Sajjad, and Frank Rudzicz. Representation noising: A defence mechanism against harmful finetuning, 2024. URL https://arxiv.org/abs/2405.14577.

Avi Schwarzschild, Zhili Feng, Pratyush Maini, Zachary C Lipton, and J Zico Kolter. Rethinking llm memorization through the lens of adversarial compression. *arXiv preprint arXiv:2404.15146*, 2024.

Abhay Sheshadri, Aidan Ewart, Phillip Huang Guo, Aengus Lynch, Cindy Wu, Vivek Hebbar, Henry Sleight, Asa Cooper Stickland, Ethan Perez, Dylan Hadfield-Menell, and Stephen Casper. Latent adversarial training improves robustness to persistent harmful behaviors in LLMs, 2025. URL https://openreview.net/forum?id=wI5uHZLeCZ.

Vinith Suriyakumar and Ashia C Wilson. Algorithms that approximate data removal: New results and limitations. *Advances in Neural Information Processing Systems*, 35:18892–18903, 2022.

Vinith M Suriyakumar, Rohan Alur, Ayush Sekhari, Manish Raghavan, and Ashia C Wilson. Unstable unlearning: The hidden risk of concept resurgence in diffusion models. *arXiv preprint arXiv:2410.08074*, 2024.

Rishub Tamirisa, Bhrugu Bharathi, Long Phan, Andy Zhou, Alice Gatti, Tarun Suresh, Maxwell Lin, Justin Wang, Rowan Wang, Ron Arel, Andy Zou, Dawn Song, Bo Li, Dan Hendrycks, and Mantas Mazeika. Tamper-resistant safeguards for open-weight LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FIjRodbW6.

Pratiksha Thaker, Yash Maurya, and Virginia Smith. Guardrail baselines for unlearning in llms. *arXiv preprint arXiv:2403.03329*, 2024.

Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 4007–4022, 2022.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of lm alignment, 2023. URL https://arxiv.org/abs/2310.16944.

Yu Wang, Ruihan Wu, Zexue He, Xiusi Chen, and Julian McAuley. Large scale knowledge washing. *arXiv preprint arXiv:2405.16720*, 2024.

Yuanshun Yao, Xiaojun Xu, and Yang Liu. Large language model unlearning. *arXiv preprint arXiv:2310.10683*, 2023.

Eric Zhang, Kai Wang, Xingqian Xu, Zhangyang Wang, and Humphrey Shi. Forget-me-not: Learning to forget in text-to-image diffusion models. *arXiv preprint arXiv:2303.17591*, 2023.

Eric Zhang, Leshem Chosen, and Jacob Andreas. Unforgettable generalization in language models, 2024a. URL https://arxiv.org/abs/2409.02228.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. Negative preference optimization: From catastrophic collapse to effective unlearning. *arXiv preprint arXiv:2404.05868*, 2024b.

Kairan Zhao, Meghdad Kurmanji, George-Octavian Bărbulescu, Eleni Triantafillou, and Peter Triantafillou. What makes unlearning hard and what to do about it, 2024. URL https://arxiv.org/abs/2406.01257.

Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, Rowan Wang, Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers, 2024. URL https://arxiv.org/abs/2406.04313.

A Algorithm

We provide a detailed description of Layered Unlearning along with graphics to better communicate the main idea.

Algorithm 1 Layer Unlearning Algorithm

Require: LLM parameters θ, datasets $\{F_1, \ldots, F_k, R\}$, forgetting coefficients α_{ij}, retaining coefficients β_i, learning rate η, loss functions $\mathcal{L}_{\text{forget}}$, $\mathcal{L}_{\text{retain}}$

Ensure: Updated model θ^*

for $i = 1$ to k **do** ▷ Iterate through sequential forget stages
  Sample mini-batches $\mathcal{F}_j \sim F_j$ for $j \leq i$, and $\mathcal{R} \sim R$
  while not converged **do** ▷ Inner optimization loop
    Compute forget loss:

$$\mathcal{L}_{\text{forget}}(\theta) = \sum_{j=1}^{i} \alpha_{ij} \mathcal{L}_{\text{forget}}(\theta, \mathcal{F}_j)$$

    Compute retain loss:

$$\mathcal{L}_{\text{retain}}(\theta) = \beta_i \mathcal{L}_{\text{retain}}(\theta, \mathcal{R})$$

    Compute total loss:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{forget}}(\theta) + \mathcal{L}_{\text{retain}}(\theta)$$

    Update model parameters:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

    if Accuracy$(F_i) < 0.35$ **then** ▷ Check forgetting threshold
      break ▷ Stop forgetting if target accuracy is reached
    end if
  end while
end for
return θ^*

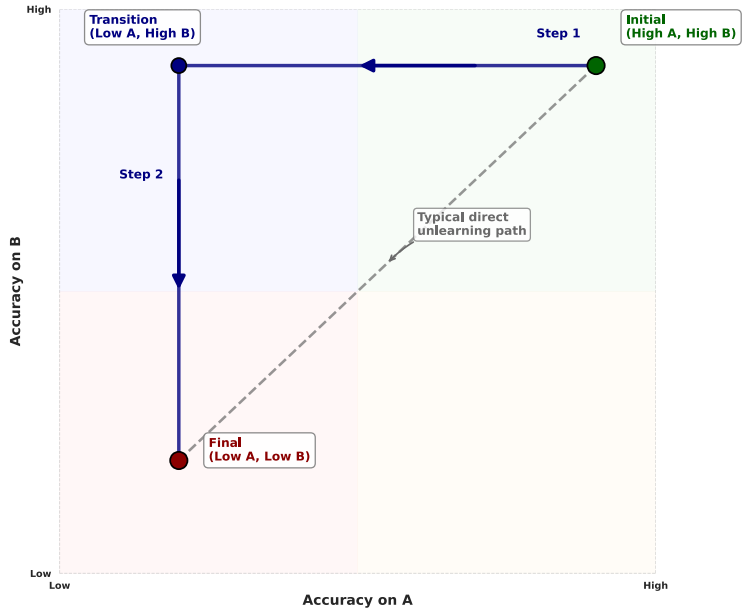

Figure 5: The performance trajectory of Layered Unlearning on two folds A, B. Normally, unlearning methods lose performance on A, B jointly and directly head towards the red point. However, we propose performing Layered Unlearning to retain performance on B while forgetting A and then forgeting both folds.

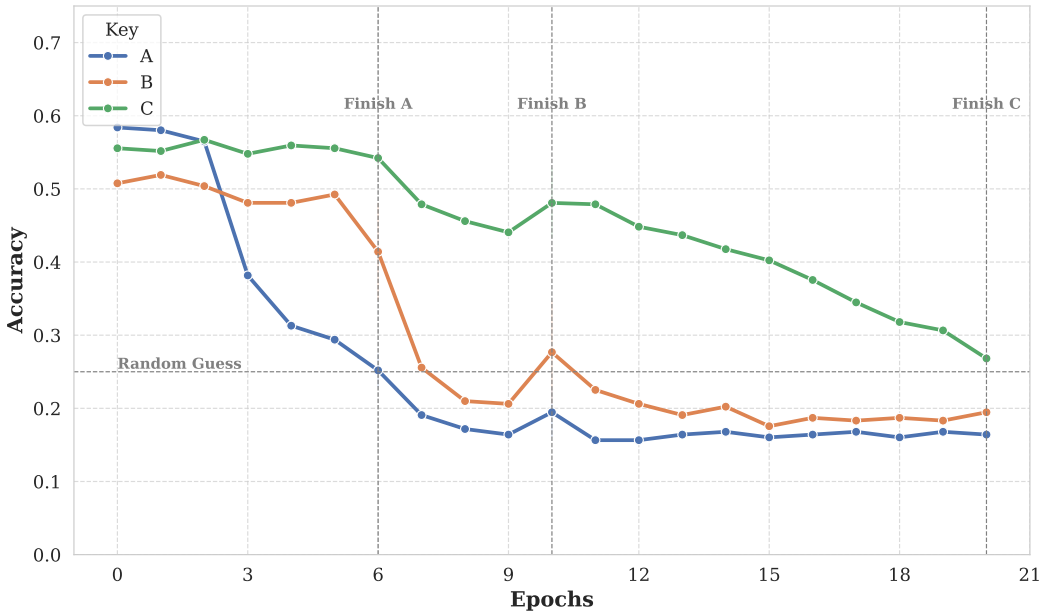

Figure 6: We show the accuracy progression of forgetting three sets A, B, C in that order using RMU-Split on WMDP. The vertical dotted gray lines show when we move to forgetting the next fold. Note that the A accuracy drops in the first iteration of forgetting and remains low. The accuracy of B remains high until the second iteration of forgetting, and then drops and remains low. Finally, the accuracy of C remains high until the third half of forgetting, when it drops.

B Synthetic ablation experiments

B.1 Binary classification

We investigate different clustering schemes for grouping Gaussians into tasks A, B, and C. In the K-Means setup, we first cluster the Gaussian means using K-Means, then solve a linear assignment problem to evenly assign clusters to tasks based on proximity. Appendix Figures 7 and 8 show that adversarial relearning becomes more effective as the number of clusters increases under random clustering. In contrast, Layered Unlearning consistently resists relearning, though its robustness is reduced under random clustering.

Our intuition is that when tasks are more distinct, adversarial relearning is less effective. K-Means clustering tends to separate tasks more cleanly, limiting overlap. Random clustering, especially with more Gaussians, increases overlap between tasks, making it harder to defend against adversarial relearning.

B.2 Bigram modeling

We analyze the effect of substituting components from the Base-LU model into the Base model on adversarial relearning, using the notation of Elhage et al. (2021). Substitutions are grouped as follows:

- QK: Replace W_Q and W_K.
- OV: Replace W_O and W_V.
- UE: Replace W_U and W_E.

These groupings reflect functional units in the model. As shown in Appendix Table 3, substituting QK or OV consistently yields the greatest robustness to adversarial relearning,

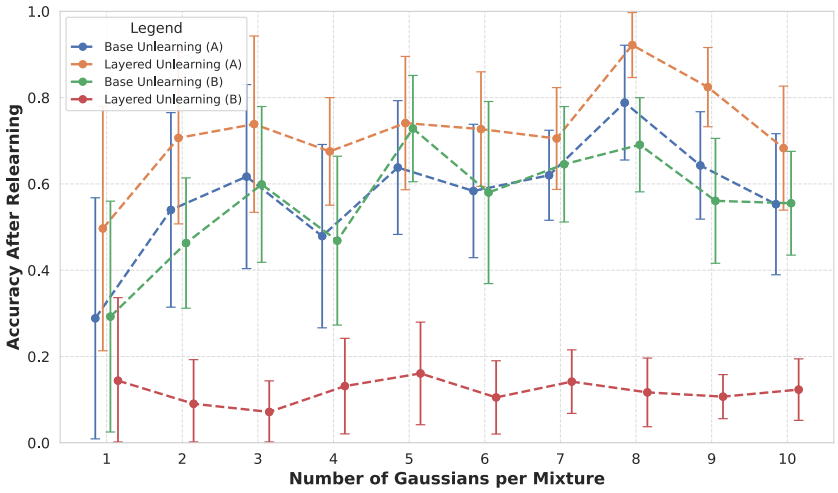

Figure 7: Relearning accuracies on A and B using datasets generated via K-Means clustering. Error bars denote 2-std confidence intervals across 10 random seeds.

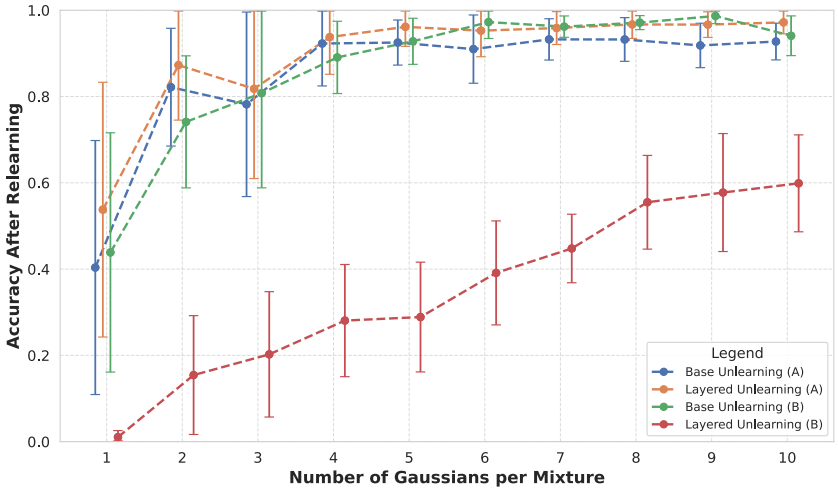

Figure 8: Relearning accuracies on A and B using datasets generated via random clustering. Error bars denote 2-std confidence intervals across 10 random seeds.

highlighting the key role of attention. This pattern is also visible in Appendix Figure 9. Notably, retain and task accuracies remain stable across all substitution settings (Appendix Table 4), indicating no degradation in core performance. Investigating how attention confers this robustness is left for future work.

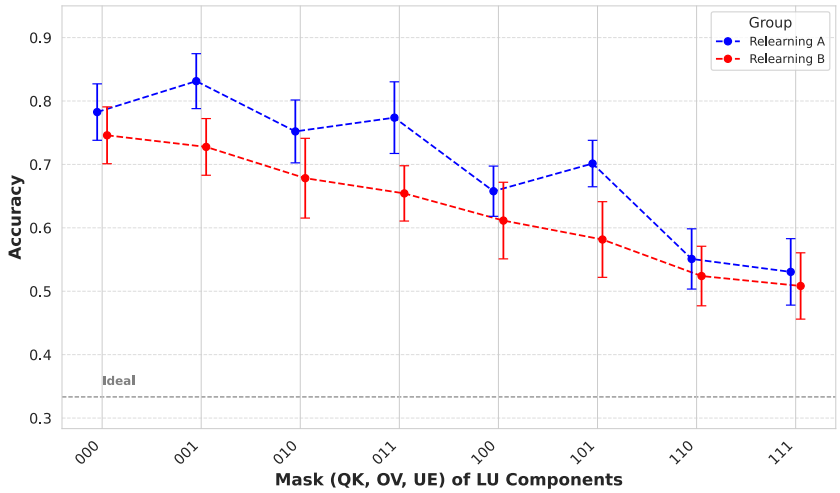

Figure 9: Relearning accuracies on A and B as a function of which components (QK, OV, UE) of the Layered Unlearning model are substituted in. The x-axis encodes binary masks (e.g., 000 is base unlearning; 111 is full Layered Unlearning). Error bars denote 2-std confidence intervals over 10 random seeds. One seed was excluded due to universal resistance to relearning, which only increased error bar size without affecting trends. "Ideal" denotes perfect unlearning.

Relearn	QK	OV	UE	A ↓	B ↓	Retain ↓
A	0	0	0	—	0.75	0.04
A	0	0	1	—	0.77	0.04
A	0	1	0	—	0.64	0.04
A	0	1	1	—	0.70	0.04
A	1	0	0	—	0.63	0.06
A	1	0	1	—	0.65	0.04
A	1	1	0	—	0.55	**0.03**
A	1	1	1	—	**0.53**	0.06
B	0	0	0	0.80	—	0.03
B	0	0	1	0.83	—	0.04
B	0	1	0	0.68	—	**0.02**
B	0	1	1	0.67	—	0.03
B	1	0	0	0.72	—	0.05
B	1	0	1	0.67	—	0.03
B	1	1	0	**0.50**	—	0.02
B	1	1	1	0.51	—	0.05

Table 3: Table of relearning accuracies when substituting different model components. A value of 0 indicates components from the Base model, while 1 indicates components from the Base-LU model. The "Retain" column reports total variation (TV) distance on the retain set.

C Retain accuracies

For evaluation, we either use full MMLU evaluation or we use specific categories for MMLU created in Deeb & Roger (2025). The retain categories for MMLU consist of questions relating to health, history, law, philosophy, and the social sciences. The forget categories for MMLU consist of questions relating to geography, culture, STEM, chemistry, and business.

QK	OV	UE	A ↓	B ↓	Retain ↓
0	0	0	0.34	0.34	0.03
0	0	1	0.37	0.38	0.02
0	1	0	**0.30**	**0.31**	0.02
0	1	1	0.31	0.34	**0.01**
1	0	0	0.34	0.33	0.03
1	0	1	0.37	0.38	0.02
1	1	0	**0.30**	**0.31**	0.02
1	1	1	0.32	0.33	**0.01**

Table 4: Unlearned accuracies after substituting model components. A value of 0 indicates components from the Base model, and 1 indicates components from the Base-LU model. The "Retain" column reports the total variation (TV) distance on the retain set.

Size	Dataset	Method	Retain ↑
Small	—	None	0.59
Full	—	None	0.58
Full	WMDP 2	RMU	0.57
Full	WMDP 2	RMU-Split	0.58
Full	WMDP 2	RMU-LU	0.57
Full	WMDP 2	RMU-Split-LU	0.57
Full	WMDP 3	RMU	0.57
Full	WMDP 3	RMU-LAT	0.56
Full	WMDP 3	RMU-Split	0.57
Full	WMDP 3	RMU-LU	0.57
Full	WMDP 3	RMU-Split-LU	0.55
Full	WMDP 3	SimNPO	0.57
Full	WMDP 3	SimNPO-LU	0.55
Full	WMDP 4	RMU	0.57
Full	WMDP 4	RMU-Split	0.57
Full	WMDP 4	RMU-LU	0.57
Full	WMDP 4	RMU-Split-LU	0.57
Small	MMLU 3	RMU	0.55
Small	MMLU 3	RMU-Split	0.56
Small	MMLU 3	RMU-LU	0.54
Small	MMLU 3	RMU-Split-LU	0.55
Full	Years 3	RMU	0.58
Full	Years 3	RMU-Split	0.58
Full	Years 3	RMU-LU	0.58
Full	Years 3	RMU-Split-LU	0.57

Table 5: Retain accuracy by method. We refer to the restricted subset of MMLU questions as the *small set* and the full MMLU dataset as the *full set*. The dataset column indicates which dataset was unlearned for each method. The first section shows results from the original model before any unlearning was applied.

D Unlearning accuracies

We provide the accuracies after applying the unlearning methods on each fold for each dataset.

Dataset	Method	A ↓	B ↓	C ↓	D ↓
WMDP 2	RMU	0.26	0.29	—	—
WMDP 2	RMU-LU	**0.21**	0.28	—	—
WMDP 2	RMU-Split	0.25	**0.26**	—	—
WMDP 2	RMU-Split-LU	0.24	0.28	—	—
WMDP 3	RMU	0.27	0.24	0.33	—
WMDP 3	RMU-LAT	0.21	0.19	0.25	—
WMDP 3	RMU-LU	**0.15**	0.24	0.33	—
WMDP 3	RMU-Split	0.22	**0.18**	**0.22**	—
WMDP 3	RMU-Split-LU	0.16	0.19	0.27	—
WMDP 3	SimNPO	0.32	0.24	0.31	—
WMDP 3	SimNPO-LU	0.29	0.32	0.33	—
WMDP 4	RMU	0.25	0.27	0.29	0.30
WMDP 4	RMU-LU	**0.15**	**0.21**	0.25	0.34
WMDP 4	RMU-Split	0.28	0.26	**0.23**	**0.22**
WMDP 4	RMU-Split-LU	0.18	0.23	0.27	0.34
MMLU 3	RMU	0.28	0.30	**0.30**	—
MMLU 3	RMU-LU	0.23	0.29	0.34	—
MMLU 3	RMU-Split	0.26	0.34	0.30	—
MMLU 3	RMU-Split-LU	**0.21**	**0.29**	0.32	—
Years 3	RMU	0.30	0.22	0.30	—
Years 3	RMU-LU	**0.29**	0.28	0.31	—
Years 3	RMU-Split	0.33	0.29	**0.22**	—
Years 3	RMU-Split-LU	0.29	**0.20**	0.27	—

Table 6: Unlearning accuracy by method across datasets and evaluation folds.

E Relearning accuracies

All attacks are performed with learning rate 10^{-6} and batch size 4. We finetune on MCQ for 8 epochs and finetune on corpus for 5 epochs. This difference is because we wish to finetune until the accuracy on the relearn set stops increasing for a while, and by definition finetuning on MCQ can reach 1.0 accuracy, so we finetune for longer. We then take the maximum validation accuracy across all epochs.

E.1 WMDP 2 folds

Table 7: Relearning accuracy across methods for WMDP 2 folds.

Relearn	Method	A ↓		B ↓	
		MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.45	**0.48**
A	RMU-LU	—	—	0.43	0.53
A	RMU-Split	—	—	**0.42**	0.51
A	RMU-Split-LU	—	—	0.42	0.54
B	RMU	0.45	0.51	—	—
B	RMU-LU	**0.30**	**0.41**	—	—
B	RMU-Split	0.40	0.53	—	—
B	RMU-Split-LU	0.32	0.48	—	—

E.2 WMDP 3 folds

Table 8: Relearning accuracy across methods for WMDP 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.41	0.45	0.45	0.49
A	RMU-LAT	—	—	0.39	0.44	0.48	0.55
A	RMU-LU	—	—	0.40	0.49	0.50	0.54
A	RMU-Split	—	—	0.37	0.44	**0.38**	0.54
A	RMU-Split-LU	—	—	**0.35**	0.45	0.49	0.55
A	SimNPO	—	—	0.47	0.45	0.50	0.54
A	SimNPO-LU	—	—	0.41	**0.35**	0.41	**0.40**
B	RMU	0.41	0.48	—	—	0.46	0.48
B	RMU-LAT	0.40	0.48	—	—	0.44	0.48
B	RMU-LU	0.31	0.44	—	—	0.50	0.54
B	RMU-Split	0.42	0.53	—	—	**0.38**	0.54
B	RMU-Split-LU	**0.29**	**0.43**	—	—	0.47	0.54
B	SimNPO	0.54	0.52	—	—	0.54	0.53
B	SimNPO-LU	0.42	0.45	—	—	0.40	**0.41**
C	RMU	0.43	0.50	0.39	0.44	—	—
C	RMU-LAT	0.39	0.50	0.42	0.43	—	—
C	RMU-LU	0.26	0.36	0.34	0.42	—	—
C	RMU-Split	0.40	0.55	0.39	0.45	—	—
C	RMU-Split-LU	**0.25**	**0.29**	**0.27**	**0.34**	—	—
C	SimNPO	0.52	0.55	0.48	0.44	—	—
C	SimNPO-LU	0.42	0.47	0.45	0.42	—	—
A, B	RMU	—	—	—	—	0.48	0.51
A, B	RMU-LAT	—	—	—	—	0.51	0.56
A, B	RMU-LU	—	—	—	—	0.49	0.56
A, B	RMU-Split	—	—	—	—	**0.41**	0.55
A, B	RMU-Split-LU	—	—	—	—	0.50	0.54
A, B	SimNPO	—	—	—	—	0.57	0.53
A, B	SimNPO-LU	—	—	—	—	0.43	**0.40**
A, C	RMU	—	—	0.43	0.47	—	—
A, C	RMU-LAT	—	—	0.40	0.47	—	—
A, C	RMU-LU	—	—	0.38	0.50	—	—
A, C	RMU-Split	—	—	0.42	0.45	—	—
A, C	RMU-Split-LU	—	—	**0.34**	0.45	—	—
A, C	SimNPO	—	—	0.48	0.44	—	—
A, C	SimNPO-LU	—	—	0.44	**0.40**	—	—
B, C	RMU	0.43	0.54	—	—	—	—
B, C	RMU-LAT	0.39	0.55	—	—	—	—
B, C	RMU-LU	**0.34**	0.47	—	—	—	—
B, C	RMU-Split	0.39	0.56	—	—	—	—
B, C	RMU-Split-LU	0.35	**0.42**	—	—	—	—
B, C	SimNPO	0.52	0.56	—	—	—	—
B, C	SimNPO-LU	0.44	0.47	—	—	—	—

E.3 WMDP 4 folds

Table 9: Relearning accuracy across methods for WMDP 4 folds.

Relearn	Method	A ↓		B ↓		C ↓		D ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.41	**0.44**	0.37	**0.44**	0.41	**0.47**
A	RMU-LU	—	—	0.38	0.46	**0.35**	0.46	**0.40**	0.57
A	RMU-Split	—	—	0.42	0.46	0.38	0.44	0.41	0.51
A	RMU-Split-LU	—	—	**0.38**	0.50	0.39	0.47	0.44	0.56
B	RMU	0.46	**0.47**	—	—	0.44	**0.41**	0.49	**0.44**
B	RMU-LU	**0.29**	0.51	—	—	**0.36**	0.46	0.46	0.56
B	RMU-Split	0.42	0.51	—	—	0.37	0.45	**0.40**	0.51
B	RMU-Split-LU	0.33	**0.47**	—	—	0.37	0.47	0.51	0.56
C	RMU	0.44	0.49	0.43	0.47	—	—	0.46	**0.45**
C	RMU-LU	0.33	**0.45**	0.37	0.47	—	—	0.46	0.57
C	RMU-Split	0.43	0.51	**0.35**	**0.46**	—	—	**0.41**	0.49
C	RMU-Split-LU	**0.29**	0.47	**0.35**	0.47	—	—	0.45	0.55
D	RMU	0.43	0.47	0.42	0.47	0.40	0.42	—	—
D	RMU-LU	**0.25**	**0.33**	**0.27**	**0.42**	0.36	0.47	—	—
D	RMU-Split	0.37	0.52	0.38	0.45	0.34	0.45	—	—
D	RMU-Split-LU	**0.25**	0.40	0.32	0.43	**0.32**	**0.40**	—	—
A, B	RMU	—	—	—	—	0.41	0.47	0.48	**0.53**
A, B	RMU-LU	—	—	—	—	0.43	0.49	0.46	0.56
A, B	RMU-Split	—	—	—	—	**0.38**	**0.45**	**0.43**	**0.53**
A, B	RMU-Split-LU	—	—	—	—	0.39	0.48	0.57	0.56
A, C	RMU	—	—	0.41	0.49	—	—	0.50	**0.49**
A, C	RMU-LU	—	—	0.43	0.48	—	—	0.46	0.57
A, C	RMU-Split	—	—	**0.38**	**0.48**	—	—	**0.39**	0.53
A, C	RMU-Split-LU	—	—	0.41	0.52	—	—	0.48	0.56
A, D	RMU	—	—	0.43	0.49	0.40	**0.45**	—	—
A, D	RMU-LU	—	—	**0.35**	**0.48**	**0.38**	0.46	—	—
A, D	RMU-Split	—	—	0.46	0.49	0.41	0.47	—	—
A, D	RMU-Split-LU	—	—	0.44	0.51	0.40	0.48	—	—
B, C	RMU	0.47	0.53	—	—	—	—	0.48	**0.51**
B, C	RMU-LU	**0.30**	0.53	—	—	—	—	0.46	0.56
B, C	RMU-Split	0.43	0.55	—	—	—	—	**0.40**	0.53
B, C	RMU-Split-LU	0.35	**0.49**	—	—	—	—	0.56	0.56
B, D	RMU	0.45	0.56	—	—	0.44	**0.46**	—	—
B, D	RMU-LU	0.35	0.51	—	—	0.42	0.47	—	—
B, D	RMU-Split	0.45	0.55	—	—	**0.39**	**0.46**	—	—
B, D	RMU-Split-LU	**0.34**	**0.49**	—	—	0.39	0.49	—	—
C, D	RMU	0.43	0.54	0.42	0.48	—	—	—	—
C, D	RMU-LU	**0.27**	**0.47**	**0.35**	**0.45**	—	—	—	—
C, D	RMU-Split	0.36	0.55	0.38	0.48	—	—	—	—
C, D	RMU-Split-LU	0.29	0.48	0.41	0.51	—	—	—	—
A, B, C	RMU	—	—	—	—	—	—	0.46	**0.52**
A, B, C	RMU-LU	—	—	—	—	—	—	0.49	0.56
A, B, C	RMU-Split	—	—	—	—	—	—	**0.42**	0.53

Continued on next page

Continued from previous page

Relearn	Method	A ↓		B ↓		C ↓		D ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A, B, C	RMU-Split-LU	—	—	—	—	—	—	0.59	0.56
A, B, D	RMU	—	—	—	—	0.44	0.48	—	—
A, B, D	RMU-LU	—	—	—	—	**0.42**	0.51	—	—
A, B, D	RMU-Split	—	—	—	—	**0.42**	**0.47**	—	—
A, B, D	RMU-Split-LU	—	—	—	—	0.45	0.49	—	—
A, C, D	RMU	—	—	0.48	0.49	—	—	—	—
A, C, D	RMU-LU	—	—	**0.39**	**0.48**	—	—	—	—
A, C, D	RMU-Split	—	—	0.49	0.49	—	—	—	—
A, C, D	RMU-Split-LU	—	—	0.43	0.51	—	—	—	—
B, C, D	RMU	0.47	0.55	—	—	—	—	—	—
B, C, D	RMU-LU	**0.31**	0.51	—	—	—	—	—	—
B, C, D	RMU-Split	0.44	0.57	—	—	—	—	—	—
B, C, D	RMU-Split-LU	0.37	**0.48**	—	—	—	—	—	—

E.4 MMLU 3 folds

Table 10: Relearning accuracy across methods for MMLU 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.49	0.66	0.51	0.64
A	RMU-LU	—	—	**0.40**	0.61	0.56	0.64
A	RMU-Split	—	—	0.52	0.63	**0.48**	**0.62**
A	RMU-Split-LU	—	—	**0.40**	**0.57**	0.50	0.64
B	RMU	0.52	0.65	—	—	0.51	**0.62**
B	RMU-LU	**0.30**	**0.54**	—	—	0.59	0.64
B	RMU-Split	0.50	0.63	—	—	**0.49**	0.62
B	RMU-Split-LU	0.38	0.61	—	—	**0.49**	0.63
C	RMU	0.53	0.64	0.52	0.64	—	—
C	RMU-LU	**0.33**	**0.39**	0.39	**0.46**	—	—
C	RMU-Split	0.55	0.65	0.57	0.64	—	—
C	RMU-Split-LU	0.36	0.48	**0.38**	0.50	—	—
A, B	RMU	—	—	—	—	0.54	0.63
A, B	RMU-LU	—	—	—	—	0.60	0.64
A, B	RMU-Split	—	—	—	—	0.61	0.64
A, B	RMU-Split-LU	—	—	—	—	**0.53**	**0.63**
A, C	RMU	—	—	0.55	0.64	—	—
A, C	RMU-LU	—	—	**0.44**	0.61	—	—
A, C	RMU-Split	—	—	0.61	0.66	—	—
A, C	RMU-Split-LU	—	—	0.48	**0.59**	—	—
B, C	RMU	0.52	0.65	—	—	—	—
B, C	RMU-LU	**0.32**	**0.55**	—	—	—	—
B, C	RMU-Split	0.53	0.67	—	—	—	—
B, C	RMU-Split-LU	0.38	0.60	—	—	—	—

E.5 Years 3 folds

Table 11: Relearning accuracy across methods for Years 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU	—	—	0.55	0.57	0.55	0.58
A	RMU-LU	—	—	0.47	0.45	**0.50**	0.53
A	RMU-Split	—	—	0.55	0.48	0.54	**0.48**
A	RMU-Split-LU	—	—	**0.43**	**0.36**	0.51	0.50
B	RMU	0.56	0.59	—	—	0.55	0.58
B	RMU-LU	0.40	0.33	—	—	0.51	**0.51**
B	RMU-Split	0.59	0.48	—	—	**0.49**	0.52
B	RMU-Split-LU	**0.37**	**0.33**	—	—	0.52	0.52
C	RMU	0.54	0.58	0.58	0.58	—	—
C	RMU-LU	**0.29**	0.32	0.43	0.42	—	—
C	RMU-Split	0.54	0.46	0.54	0.47	—	—
C	RMU-Split-LU	0.29	**0.31**	**0.42**	**0.38**	—	—
A, B	RMU	—	—	—	—	0.61	0.58
A, B	RMU-LU	—	—	—	—	**0.51**	**0.52**
A, B	RMU-Split	—	—	—	—	0.59	0.54
A, B	RMU-Split-LU	—	—	—	—	0.56	0.53
A, C	RMU	—	—	0.61	0.57	—	—
A, C	RMU-LU	—	—	0.52	0.45	—	—
A, C	RMU-Split	—	—	0.60	0.51	—	—
A, C	RMU-Split-LU	—	—	**0.51**	**0.42**	—	—
B, C	RMU	0.62	0.59	—	—	—	—
B, C	RMU-LU	0.43	**0.32**	—	—	—	—
B, C	RMU-Split	0.59	0.50	—	—	—	—
B, C	RMU-Split-LU	**0.40**	0.34	—	—	—	—

F Relearning recover rates

When computing recover rates, we compare how much performance is regained between two unlearning algorithms, effectively normalizing for the choice of base method. We specify each algorithm and its base algorithm as follows:

- RMU-LAT: RMU.
- RMU-LU: RMU.
- RMU-Split-LU: RMU-Split.
- SimNPO-LU: SimNPO.

F.1 WMDP 2 folds

Table 12: Recover rate across methods for WMDP 2 folds.

Relearn	Method	A ↓		B ↓	
		MCQ	Corpus	MCQ	Corpus
A	RMU-LU	—	—	0.95	1.35
A	RMU-Split-LU	—	—	**0.92**	**1.07**
B	RMU-LU	**0.26**	**0.66**	—	—
B	RMU-Split-LU	0.43	0.82	—	—

F.2 WMDP 3 folds

Table 13: Recover rate across methods for WMDP 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU-LAT	—	—	0.88	0.96	1.76	1.75
A	RMU-LU	—	—	0.91	1.21	1.33	1.25
A	RMU-Split-LU	—	—	0.84	1.04	1.77	0.96
A	SimNPO-LU	—	—	**0.40**	**0.17**	**0.40**	**0.30**
B	RMU-LAT	1.07	1.10	—	—	1.40	1.51
B	RMU-LU	0.42	0.88	—	—	1.29	1.44
B	RMU-Split-LU	**0.26**	**0.64**	—	—	1.54	0.95
B	SimNPO-LU	0.58	0.79	—	—	**0.31**	**0.38**
C	RMU-LAT	0.85	1.07	1.19	0.92	—	—
C	RMU-LU	0.06	0.48	0.62	0.88	—	—
C	RMU-Split-LU	**0.00**	**0.15**	**0.15**	**0.43**	—	—
C	SimNPO-LU	0.63	0.76	0.58	0.51	—	—
A, B	RMU-LAT	—	—	—	—	1.63	1.68
A, B	RMU-LU	—	—	—	—	1.05	1.26
A, B	RMU-Split-LU	—	—	—	—	1.40	0.91
A, B	SimNPO-LU	—	—	—	—	**0.39**	**0.33**
A, C	RMU-LAT	—	—	0.83	1.02	—	—
A, C	RMU-LU	—	—	0.71	1.14	—	—
A, C	RMU-Split-LU	—	—	0.51	1.00	—	—
A, C	SimNPO-LU	—	—	**0.50**	**0.42**	—	—
B, C	RMU-LAT	0.87	1.10	—	—	—	—
B, C	RMU-LU	**0.58**	0.78	—	—	—	—
B, C	RMU-Split-LU	0.73	**0.56**	—	—	—	—
B, C	SimNPO-LU	0.75	0.74	—	—	—	—

F.3 WMDP 4 folds

Table 14: Recover rate across methods for WMDP 4 folds.

Relearn	Method	A ↓		B ↓		C ↓		D ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU-LU	—	—	0.93	1.24	1.19	1.37	**0.62**	1.35
A	RMU-Split-LU	—	—	**0.78**	**1.22**	**0.96**	**1.08**	0.68	**0.84**
B	RMU-LU	**0.21**	1.18	—	—	**0.71**	1.71	**0.68**	1.63
B	RMU-Split-LU	0.56	**0.97**	—	—	0.83	**1.05**	1.13	**0.86**
C	RMU-LU	0.43	**0.86**	**0.77**	1.10	—	—	0.75	1.59
C	RMU-Split-LU	**0.26**	0.97	1.05	**1.10**	—	—	**0.74**	**0.89**
D	RMU-LU	**0.00**	**0.37**	**0.10**	**0.85**	1.00	1.63	—	—
D	RMU-Split-LU	**0.00**	0.61	0.56	0.92	**0.56**	**0.68**	—	—
A, B	RMU-LU	—	—	—	—	1.40	1.31	**0.71**	1.00
A, B	RMU-Split-LU	—	—	—	—	**0.96**	**1.05**	1.25	**0.81**
A, C	RMU-LU	—	—	**1.30**	**1.07**	—	—	**0.64**	1.22
A, C	RMU-Split-LU	—	—	1.33	1.18	—	—	1.04	**0.80**
A, D	RMU-LU	—	—	**0.65**	**1.05**	1.18	1.27	—	—
A, D	RMU-Split-LU	—	—	0.90	1.09	**0.84**	**0.98**	—	—
B, C	RMU-LU	**0.26**	1.00	—	—	—	—	**0.69**	1.07
B, C	RMU-Split-LU	0.68	**0.88**	—	—	—	—	1.47	**0.80**
B, D	RMU-LU	**0.48**	**0.85**	—	—	1.13	1.29	—	—
B, D	RMU-Split-LU	0.52	0.90	—	—	**0.93**	**1.07**	—	—
C, D	RMU-LU	**0.11**	**0.77**	**0.66**	**0.95**	—	—	—	—
C, D	RMU-Split-LU	0.55	0.85	1.24	1.11	—	—	—	—
A, B, C	RMU-LU	—	—	—	—	—	—	**0.94**	1.02
A, B, C	RMU-Split-LU	—	—	—	—	—	—	1.48	**0.80**
A, B, D	RMU-LU	—	—	—	—	**1.10**	1.31	—	—
A, B, D	RMU-Split-LU	—	—	—	—	1.12	**1.00**	—	—
A, C, D	RMU-LU	—	—	**0.67**	**1.05**	—	—	—	—
A, C, D	RMU-Split-LU	—	—	0.77	1.09	—	—	—	—
B, C, D	RMU-LU	**0.30**	0.87	—	—	—	—	—	—
B, C, D	RMU-Split-LU	0.77	**0.80**	—	—	—	—	—	—

F.4 MMLU 3 folds

Table 15: Recover rate across methods for MMLU 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU-LU	—	—	**0.61**	**0.89**	1.04	**0.87**
A	RMU-Split-LU	—	—	0.63	0.96	**1.02**	1.01
B	RMU-LU	**0.20**	**0.79**	—	—	1.20	**0.93**
B	RMU-Split-LU	0.53	0.96	—	—	**0.92**	0.98
C	RMU-LU	**0.33**	**0.38**	0.44	**0.51**	—	—
C	RMU-Split-LU	0.38	0.59	**0.40**	0.71	—	—
A, B	RMU-LU	—	—	—	—	1.05	**0.89**
A, B	RMU-Split-LU	—	—	—	—	**0.70**	0.93
A, C	RMU-LU	—	—	**0.59**	**0.93**	—	—
A, C	RMU-Split-LU	—	—	0.72	0.94	—	—
B, C	RMU-LU	**0.30**	**0.82**	—	—	—	—
B, C	RMU-Split-LU	0.47	0.84	—	—	—	—

F.5 Years 3 folds

Table 16: Recover rate across methods for Years 3 folds.

Relearn	Method	A ↓		B ↓		C ↓	
		MCQ	Corpus	MCQ	Corpus	MCQ	Corpus
A	RMU-LU	—	—	**0.65**	**0.53**	**0.77**	**0.79**
A	RMU-Split-LU	—	—	0.67	0.56	0.84	1.01
B	RMU-LU	0.42	**0.17**	—	—	**0.81**	**0.70**
B	RMU-Split-LU	**0.32**	0.27	—	—	1.04	0.90
C	RMU-LU	**0.00**	**0.11**	**0.46**	**0.44**	—	—
C	RMU-Split-LU	0.02	0.17	0.67	0.69	—	—
A, B	RMU-LU	—	—	—	—	**0.66**	**0.78**
A, B	RMU-Split-LU	—	—	—	—	0.86	0.89
A, C	RMU-LU	—	—	**0.67**	**0.55**	—	—
A, C	RMU-Split-LU	—	—	0.84	0.76	—	—
B, C	RMU-LU	0.46	**0.13**	—	—	—	—
B, C	RMU-Split-LU	**0.41**	0.29	—	—	—	—

G Hyperparameters

We set the forgetting threshold to 0.35, motivated by the following statistical reasoning.

For multiple-choice questions (MCQ), assuming random guessing, the expected accuracy is 0.25. Each dataset contains a total of 735 questions, and the smallest fold we consider consists of $\frac{735}{4}$. By applying the central limit theorem, the expected final accuracy follows approximately a normal distribution:

$$\mathcal{N}\left(\frac{1}{4}, \sqrt{\frac{1}{4}\cdot\frac{3}{4}\cdot\frac{4}{735}}\right) \approx \mathcal{N}\left(0.25, 0.032\right).$$

A three-standard-deviation event corresponds to:

$$3 \times 0.032 + 0.25 \leq 0.35.$$

Since we do not reject the null hypothesis of random guessing if accuracy remains within three standard deviations of 0.25, we set the forgetting threshold to approximately 0.35. Note that this bound does get tighter if we consider bigger folds, but in practice, we find that this threshold does not significantly impact results, so we standardize it across all instances of Layered Unlearning.

For all models, we use Zephyr-7B-β (Tunstall et al., 2023).

G.1 RMU hyperparameters

WMDP: official model checkpoint from (Li et al., 2024a). MMLU:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficient: 2.00.
- Retain coefficient: 16.00.
- Learning rate: 5×10^{-5}.
- Batch size: 4.

Years:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 15.
- Forget coefficient: 0.25.
- Retain coefficient: 1.00.
- Learning rate: 5×10^{-5}.
- Batch size: 4.

G.2 RMU-Split hyperparameters

WMDP 2 folds:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficients: 1.00, 1.00.
- Retain coefficient 32.

- Learning rate: 1×10^{-5}.
- Batch size: 4.

WMDP 3 folds:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficients: 1.00, 1.00, 1.00.
- Retain coefficient 16.
- Learning rate: 1×10^{-5}.
- Batch size: 4.

WMDP 4 folds:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficients: 1.00, 1.00, 1.00, 1.00.
- Retain coefficient 32.
- Learning rate: 1×10^{-5}.
- Batch size: 4.

MMLU 3 folds:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficients: 2.00, 2.00, 2.00.
- Retain coefficient 24.
- MMLU retain coefficient: 12.0.
- Learning rate: 1×10^{-5}.
- Batch size: 8.

Years 3 folds:

- Activation layer: 7.
- Layers finetuned: 5, 6, 7.
- Magnitude: 10.
- Forget coefficients: 1.00, 1.00, 1.00.
- Retain coefficient 32.
- Learning rate: 1×10^{-5}.
- Batch size: 4.

G.3 RMU-LAT hyperparameters

WMDP: official model checkpoint from (Sheshadri et al., 2025).

G.4 RMU-LU hyperparameters

WMDP 2 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00.
  - Retain coefficients: 0.00, 13.52.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 1.00.
  - Retain coefficients: 0.00, 0.00.
  - Retain set coefficient: 24.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.

WMDP 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00, 0.00.
  - Retain coefficients: 0.00, 6.76, 6.76.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.05, 0.50, 0.00.
  - Retain coefficients: 0.00, 0.00, 8.00.
  - Retain set coefficient: 16.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.00, 0.03, 0.33.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 24.
  - Learning rate: 1×10^{-5}.

  - Batch size: 8.

WMDP 4 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 10.67, 10.67, 10.67.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 1.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 4.00, 4.00.
  - Retain set coefficient: 16.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.01, 0.07, 0.67, 0.00.
  - Retain coefficients: 0.00, 0.00, 0.00, 8.00.
  - Retain set coefficient: 16.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.
- Stage 4:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.01, 0.03, 0.15, 0.75.
  - Retain coefficients: 0.00, 0.00, 0.00, 0.00.
  - Retain set coefficient: 32.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.

MMLU 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 2.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 8.00, 8.00.
  - Retain set coefficient: 2.
  - Learning rate: 1×10^{-5}.

  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 2.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 4.00.
  - Retain set coefficient: 32.
  - MMLU retain coefficient: 8.0.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.01, 0.13, 2.67.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 36.
  - MMLU retain coefficient: 18.0.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.

Years 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 4.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 16.00, 16.00.
  - Retain set coefficient: 2.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 2.25, 22.50, 0.00.
  - Retain coefficients: 0.00, 0.00, 16.00.
  - Retain set coefficient: 16.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.15, 1.50, 15.00.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 32.
  - Learning rate: 1×10^{-5}.
  - Batch size: 8.

G.5 RMU-Split-LU hyperparameters

WMDP 2 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00.
  - Retain coefficients: 0.00, 13.52.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 0.20.
  - Retain coefficients: 0.00, 0.00.
  - Retain set coefficient: 14.51609.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.

WMDP 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00, 0.00.
  - Retain coefficients: 0.00, 6.76, 6.76.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 1.00, 4.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 13.52.
  - Retain set coefficient: 32.
  - Learning rate: 3×10^{-6}.
  - Batch size: 4.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.33, 1.33, 5.33.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 45.51609.
  - Learning rate: 3×10^{-6}.

  - Batch size: 12.

WMDP 4 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.39, 0.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 10.67, 10.67, 10.67.
  - Retain set coefficient: 1.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 0.20, 0.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 16.00, 16.00.
  - Retain set coefficient: 8.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.03, 0.07, 0.13, 0.00.
  - Retain coefficients: 0.00, 0.00, 0.00, 32.00.
  - Retain set coefficient: 16.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 4:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.03, 0.06, 0.12, 0.25.
  - Retain coefficients: 0.00, 0.00, 0.00, 0.00.
  - Retain set coefficient: 32.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.

MMLU 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 2.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 8.00, 8.00.
  - Retain set coefficient: 2.
  - Learning rate: 1×10^{-5}.

  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 0.10, 2.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 8.00.
  - Retain set coefficient: 32.
  - MMLU retain coefficient: 8.0.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 10.
  - Forget coefficients: 0.00, 0.07, 1.33.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 40.
  - MMLU retain coefficient: 10.0.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.

Years 3 folds:

- Stage 1:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 6.5.
  - Forget coefficients: 4.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 16.00, 16.00.
  - Retain set coefficient: 2.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 2:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 12.
  - Forget coefficients: 1.20, 4.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 32.00.
  - Retain set coefficient: 2.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.
- Stage 3:
  - Activation layer: 7.
  - Layers finetuned: 5, 6, 7.
  - Magnitude: 12.
  - Forget coefficients: 0.17, 0.67, 2.67.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 36.
  - Learning rate: 1×10^{-5}.
  - Batch size: 4.

G.6 SimNPO hyperparameters

WMDP:

- Beta: 0.1.
- Forget coefficients: 8.00.
- Retain coefficients: 1.00.
- Learning rate: 4×10^{-6}.
- Batch size: 4.

We reran SimNPO with our own implementation as the checkpoint online did not sufficiently unlearn the data on our folds.

G.7 SimNPO-LU hyperparameters

WMDP 3 folds:

- Stage 1:
  - Beta: 0.1.
  - Forget coefficients: 3.00, 0.00, 0.00.
  - Retain coefficients: 0.00, 1.00, 1.00.
  - Retain set coefficient: 4.
  - Learning rate: 4×10^{-6}.
  - Batch size: 4.
- Stage 2:
  - Beta: 0.1.
  - Forget coefficients: 0.60, 3.00, 0.00.
  - Retain coefficients: 0.00, 0.00, 1.00.
  - Retain set coefficient: 6.
  - Learning rate: 4×10^{-6}.
  - Batch size: 4.
- Stage 3:
  - Beta: 0.1.
  - Forget coefficients: 5.00, 5.00, 5.00.
  - Retain coefficients: 0.00, 0.00, 0.00.
  - Retain set coefficient: 6.
  - Learning rate: 4×10^{-6}.
  - Batch size: 4.