REMOVING SEQUENTIALLY TRAINED MODULES: EFFECTS ON RETAINED SOURCES AND SUBSEQUENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We show that teaching a language model new information through separate, trainable components does not ensure that removing a component leaves what other components learned unaffected, and that removal can make subsequent learning easier. This matters for models that are continually learning from new datasets or environments while needing to dynamically disable some components and leave others activated. We test this by giving each dataset its own separately trainable component, called a module, while freezing the original model weights and varying which modules are active while subsequent modules learn. On synthetic tasks across five seeds we find that removing the first module trained reduces accuracy on the remaining datasets by 93.7 percentage points; when randomly disabling each earlier-trained module with probability .75 during training we avoid this observed degradation entirely at the cost of 58.3% more fitting steps. A separate three-seed experiment showed that removing four sources and their recorded dependent modules reduced the total steps needed to learn a new source to 43.6% of the no-removal baseline, although this removed all eight earlier modules.

1 INTRODUCTION

Teaching a model through separate trainable components, which we call modules, makes it trivial to both identify and remove the parameters updated by a specific dataset during module training. We show that removing a module can lead to damaging what the model learned from other data sources not associated with the module, due to later-trained modules relying on the outputs of earlier modules. For models that must both continually learn from new datasets as well as selectively remove modules over time, this dependence matters for balancing learning from new data with preserving performance on existing tasks.

We study this behavior by using residual modules, which we call lanes, where each source is assigned its own lane, and changing what modules, if any, are active while new ones are learning. Prior work already supports adding, combining, and removing separate components, including DEMix and AdapterSwap, while GRAM trains the model to be more robust against having different modules active at the same time (Gururangan et al., 2022; Fleshman et al., 2024; Roland et al., 2026). Our approach builds on these methods by freezing the backbone and earlier-trained lane parameters, then changing whether or not they contribute to the forward pass during later training. This lets us test whether dependence on those outputs is enough to cause deletion damage (Figure 1). We also record dependencies and test cascade deletion, in which a source deletion request removes the source's module and its recorded direct and indirect dependents.

On synthetic tasks across a five-seed sweep, removing the first trained module reduced accuracy on surviving sources by $93.70\,\mathrm{pp}$. Module-level dropout at $p = .75$ avoided this accuracy loss at the cost of 58.3% more fitting steps. We repeated these training conditions at 360M and 1.7B parameters using the same five datasets and training code, and found that both effects persisted at these sizes. In a separate three-seed sweep, removing sources and their recorded dependents reduced the steps needed to fit a new lane to 43.6% of the no-deletion baseline, although this cascade deletion did remove all eight earlier-trained lanes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

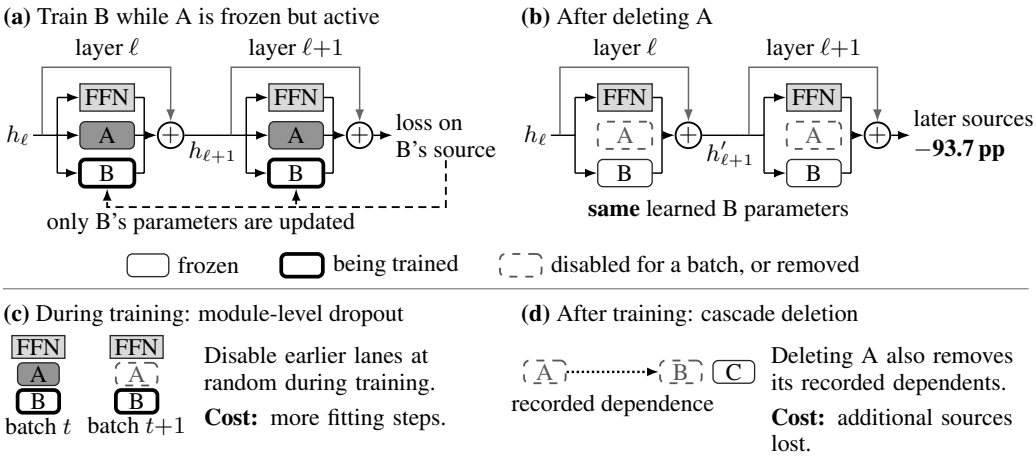

Figure 1: Frozen parameters do not prevent dependence. (a) We train B with A frozen but active; their outputs contribute to the computation entering deeper layers. (b) Removing A leaves B's parameters unchanged. In the five-seed 135M experiment without dropout, removing the first lane reduced retained-source accuracy by 93.7 pp. (c) Module-level dropout reduces deletion damage at the cost of more fitting steps. (d) Cascade deletion removes recorded dependents at the cost of additional sources. Attention and normalization are omitted.

2 Experimental Setup

We give each source its own low-rank residual lane at each decoder block. Each lane uses two trainable matrices to process the same input as the feed-forward sublayer, and we add its output to the feed-forward output. Active lanes receive the same input within each layer, but their outputs change what the deeper layers receive as input. When training on source j, we only update its lane, keeping the model backbone and all earlier-trained lanes frozen.

We tested three different training conditions. Prior-active training trains new lanes with all earlier lanes still active and contributing to the model's forward pass. Independent training trains each lane without any other lanes active. Module-level dropout independently disables earlier lanes with probability p per training batch.

At evaluation time, we use the dataset each question came from to decide which lanes are active for that question. For questions from source j, we activate lane j and all lanes trained prior to it for the prior-active and dropout-enabled training conditions; for independent training we only activate lane j. We also ran a separate experiment activating all independently trained lanes together as well as training a router to select active lanes per question.

2.1 Deletion and recorded dependencies

We test deletion by removing the requested lanes from the active set without changing the model backbone or other lane parameters. We first remove the first lane trained, then separately test removing the last lane and removing the first two lanes together. For each deletion, we compare accuracy on the same question set before and after removal, and we call this change collateral:

$$\Delta_R(D) = \mathrm{Acc}(R; \mathcal{P} \setminus D) - \mathrm{Acc}(R; \mathcal{P}), \tag{1}$$

where $\mathcal{P}$ is the set of trained lanes, D is the set of removed lanes, and R contains questions from the datasets that we want to keep. Negative collateral means that deletion reduced accuracy on R.

We also test deletion by tracking dependent lanes. We track dependencies in two ways: an exposure graph and a functional graph. The exposure graph records an edge $i \to j$ when lane i contributes to the model's forward pass while lane j is training. The functional graph records an edge when disabling lane i reduces the model's accuracy on source j by at least τ. We use $\tau = .05$ everywhere unless stated otherwise.

Another way we tested deletion was through cascade deletion, where we remove lane j as well as all directly or indirectly dependent lanes. We do this by choosing a dependency graph, then following all outgoing edges from lane j and removing all lanes reachable from those outgoing edges. We count the additional sources removed as the cascade's blast radius.

If none of the sources survive deletion, retained accuracy is marked as undefined. Since the functional graph depends on the threshold and evaluation context, a cascade based on it may miss dependencies that the ablations did not detect.

2.2 Data and training

We use synthetic biographies so that we can control the facts each source contains and which facts are shared among sources. For each source, we create ten biographies with invented names and four factual fields per biography. We ask about each field in five different ways - four questions and one fact-completion prompt - which gives us a total of 200 question-answer pairs covering 40 facts per source. We also include the ten biography texts during training, for a total of 210 training examples per source. In our primary experiment, we train on three sources with no overlapping facts and hold out one additional source from training as a control. We also test sources with overlapping information, where the sources describe the same entity with conflicting facts, duplicate another source's texts, or share a portion of their facts. In these and the eight-source experiments, every prompt begins with a tag that names its source.

To evaluate accuracy, we have the model rank 256 candidate answers for the field that is being asked about, and check whether it ranks the correct answer first. We rank candidates by their average token loss over the answer, without including the prompt tokens. This is done instead of freeform generation to control for differences in the generated format. We evaluate on questions that were present in training in the primary experiment since we are testing whether the model can still answer them after lanes have been removed.

Before training, we check to make sure the model cannot already answer the questions. We place an upper bound on accuracy of 2% and a Wilson upper confidence bound of 5% for every source. If any source fails this check, that entire set of sources is excluded from training in any of the three training conditions specified previously. We check accuracy during training in the primary experiment every 200 steps and stop fitting each lane once it reaches 90% accuracy, with an upper bound of 2400 steps per lane. If a lane does not reach the target accuracy within this step budget, that run is marked as a failure to fit within budget.

We use SmolLM2-135M (Ben Allal et al., 2025), with lane rank and scale both set to 32. We train with AdamW at learning rate 10^{-3}, using one example per batch, a maximum sequence length of 192 tokens, and loss over the full example. We test module-level dropout at $p \in \{0, .25, .5, .75\}$. Dropout within each lane is $.05$, separate from disabling the lane fully.

2.3 Learning after deletion

We also test whether removing earlier-trained lanes changes the training needed to learn a new source. Starting with eight trained sources, we train a ninth lane with earlier-trained lanes present, after direct deletion, or after cascade deletion. We compare fitting steps against the no-deletion condition using the same new-source data and fitting target, and report how many earlier-trained lanes remain. We also estimate training computation as removing earlier-trained lanes can make each step cheaper as well as reduce the total number of fitting steps needed.

2.4 Repeated runs and uncertainty

We run the primary experiment across a five-seed sweep, using the same datasets for every training condition within each seed. We fix the candidate seed order before running the experiment, and use the first five sets of sources that pass the pre-training accuracy check described above. Six sets failed this check before we obtained five sets of sources that passed. For each selected seed, we use each of the four module-level dropout probabilities as well as independent training, giving a total of 25 completed runs with no fitting or technical failures. We give the seed selection details in Appendix A.

We report the mean and sample standard deviation across seeds. To compare training conditions, we calculate the difference between them for each seed and then use these five differences to compute the mean difference and a 95% t confidence interval for the primary experiment.

We repeated all five training condition experiments at 360M and 1.7B parameters using the exact same datasets, training code, and settings, with gradient checkpointing enabled for the larger models. Combined with the primary experiment, this gives us a total of 75 runs across three model sizes. The earlier experiments with shared information and learning after deletion are reported separately and use three seeds, unless stated otherwise.

3 Results

3.1 Frozen parameters do not prevent dependence

We first tested whether keeping earlier lane parameters frozen was enough to preserve model performance on the sources that remained after deletion. We compared prior-active against independent training, using the same datasets and lane implementation code across a five-seed sweep. Before deletion, these training conditions yielded $96.33 \pm 1.22\%$ and $97.47 \pm 1.91\%$ accuracy, respectively. Removing the first lane trained reduced accuracy by 93.70 ± 3.17 pp under the prior-active training condition, compared with no change in accuracy under independent training (Table 1).

In both training conditions, earlier parameters were frozen, although those parameters' outputs were available during later training only under prior-active training. The resulting accuracy drop therefore signifies that later-trained lanes can become dependent on earlier outputs without changing the parameters that produce them. Removing the first two lanes trained yielded an accuracy drop of 96.30 ± 3.65 pp under the prior-active training condition. Removing the last lane trained gave zero collateral, as expected: questions from earlier sources would not have activated that lane.

Another, separate five-seed sweep with sequentially merged LoRA adapters also found large deletion damage, with an accuracy drop of 95.90 pp after removing the first adapter with a 95% confidence interval of $[93.77, 98.03]$. In that experiment, the earlier adapters' updates were merged into the model backbone before the later adapters were trained. Removing an earlier update consequently changed the model backbone that the later adapters had trained with. In our primary lane experiment, the model backbone itself was unchanged, but deletion still caused a large accuracy drop. This does establish dependence on earlier outputs without changes to earlier weights, rather than showing that all modular training must behave in this way.

3.2 Module-level dropout reduces damage at an additional fitting cost

We next tested whether disabling earlier-trained lanes during training would reduce deletion damage, keeping the datasets and fitting targets fixed. As shown in Table 1, increasing p reduced mean damage from removing either the first trained lane or the first two lanes together. At $p = .5$, removing two lanes still caused more damage than removing the first one. At $p = .75$, neither

Table 1: Primary comparison on disjoint sources, five seeds; mean $\pm$ sample SD. Collateral is the change in accuracy on the same retained questions, so negative values indicate damage. First and joint deletion remove the first lane and the first two lanes, respectively. Independent evaluation selects only the source's own lane; it does not activate all independent lanes together. Fit steps are totals over the three trained lanes.

Condition	Active accuracy (%)	First-delete collateral (pp)	Joint-delete collateral (pp)	Fit steps
$p = 0$	96.33 ± 1.22	-93.70 ± 3.17	-96.30 ± 3.65	2400 ± 200
$p = .25$	96.13 ± 2.42	-28.65 ± 8.04	-61.30 ± 8.80	3080 ± 178.89
$p = .5$	95.67 ± 1.43	-2.90 ± 2.92	-9.00 ± 8.46	3320 ± 178.89
$p = .75$	94.70 ± 1.67	$+3.65 \pm 2.32$	$+5.00 \pm 3.04$	3800 ± 141.42
Independent	97.47 ± 1.91	0.00 ± 0.00	0.00 ± 0.00	3080 ± 109.54

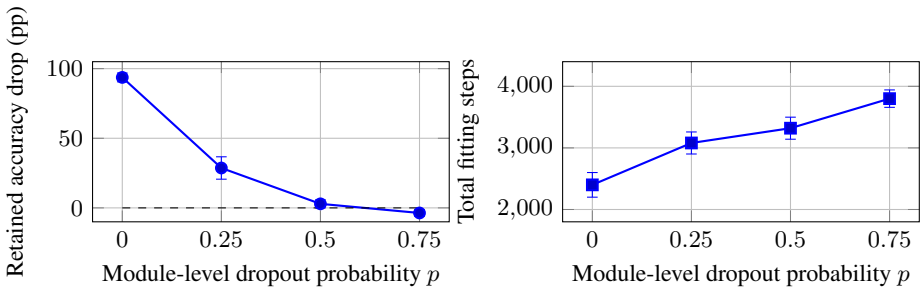

Figure 2: Deletion damage and fitting cost across module-level dropout probabilities. The left panel shows the retained accuracy drop after removing the first lane; negative values indicate an accuracy increase. The right panel shows total fitting steps. Points are means over five seeds, with sample-standard-deviation error bars. These are the current primary runs, not the historical three-seed comparison.

request reduced mean accuracy and removing the first lane trained actually increased accuracy across all five seeds. This specifically shows protection against deletion requests, not that deletion generally improves performance.

This protection came at the cost of more training. The mean steps to hit the fitting target went from 2400 without module-level dropout to 3800 at $p = .75$, a 58.3% increase (Figure 2). The mean increase within the same seed was 1400 steps, with a 95% confidence interval of $[1224, 1576]$. We did not find clear evidence of lower accuracy pre-deletion in this 135M experiment, as the paired difference was -1.63 pp with a 95% confidence interval of $[-4.99, 1.72]$. Our measurements indicate fitting required more steps, while the pre-deletion accuracy difference remains uncertain.

3.3 Recording dependencies makes additional removal explicit

We next tested whether recording lane dependencies could help account for deletion damage. In a separate three-seed sweep experiment with duplicated sources, lanes trained with $p = .5$ outperformed independent LoRA by 2.37 pp, with a 95% confidence interval of $[.20, 4.55]$. Removing an earlier lane also reduced accuracy on retained sources by 30.22 ± 1.50 pp, however. Since these conditions used different module implementations, the accuracy advantage cannot be attributed to the earlier-trained lanes being active alone. Cascade deletion only reduced accuracy by $.17$ pp on unrelated sources, but removed the dependent source fully, accounting for that source's loss by removing it instead of preserving its performance at all.

The dependency graph chosen also impacted which sources were removed. Across the eight-source three-seed sweep experiments, the functional graph kept 65.5% of its exposure edges without dropout and 7.1% when $p = .75$, using $\tau = .05$ (Table 6, Appendix C). Exposure-based cascade deletions removed sources that the functional graph would have kept. At $p = .5$, increasing τ from .01 to .10 reduced the average additional sources removed from 2.00 to .38.

In the experiments with different source relationships and deletion policies, cascade deletion did not consistently improve accuracy over direct removal. In some cases it removed every source, with none left to evaluate (Appendix C). Recording dependencies makes removal explicit and predictable but does not necessarily improve performance on the surviving sources. We compared direct and cascade deletion on the same surviving sources, so removing additional sources could not improve the comparison simply by excluding their questions.

3.4 Independently trained lanes do not work together under ungated addition

We next tested whether independently trained lanes could work together without selecting specific ones to be active. Independent training had avoided deletion damage when each source was evaluated with its corresponding lane. When those lanes were activated together, however, accuracy went from $97.47 \pm 1.91\%$ to $2.53 \pm 1.02\%$ across the five-seed sweep. The lane parameters were

Table 2: Routing and recovery with four trained sources across three seeds. Values are mean ± sample standard deviation across seeds. For recovery, we first average the three deletion possibilities within each seed. Useful misroutes are questions the model answered correctly even though the router selected the wrong source. We add these counts across all three seeds. Correct answers can survive removal when the surviving sources share the facts.

Relation	Route accuracy (%)	Useful misroutes	Deleted-source recovery (%)
Shared entity, conflicting facts	$95.83 \pm .72$	$0/20$	0.00 ± 0.00
Duplicated sources	93.12 ± 1.25	$31/33$	89.17 ± 2.50
Partial fact overlap	$95.83 \pm .72$	$11/20$	41.39 ± 2.93

unchanged across this evaluation. Each lane had learned its corresponding source, but adding their outputs together did not preserve their source-specific performance.

A separate three-seed sweep diagnostic with $p = 1$ showed the same problem, where the lane reached 100% accuracy alone, but the same parameters gave only 10–16.5% with the earlier lane active after 2400 steps, and 17–24% after 4800 steps. Independent acquisition via $p = 1$ did succeed, but simply adding the output of the lanes together remained below the fitting target even with double the training budget (Appendix B).

We then tested if training a router to select the lane based on the prompt would improve performance. In a single-seed experiment, this recovered 97.19% accuracy after only 75 router training steps. In this case, routing was easy because the entities in the questions identified which source they came from. We further tested routing across a three-seed sweep where the sources had overlapping entities and each prompt's tag named its source. Here, the routers selected the correct source on 93–96% of the questions held out from router training, and reached 100% training accuracy within the 1000-step cap (Table 2; Appendix D). None of the routers reached our 98% held-out accuracy target within the step cap, although they still selected the intended source for most questions. Whether those mistakes changed the answer depended on what the selected source contained.

3.5 Removing a source does not necessarily remove its answers

We also tested whether the remaining lanes could answer questions from a deleted source. We removed the requested lane, prevented the router from selecting it, and then evaluated its questions using the other lanes. Across a three-seed sweep, sources which had duplicates of the deleted source's texts answered the deleted source's questions with $89.17 \pm 2.50\%$ accuracy. When there was some fact overlap amongst the sources, it reached $41.39 \pm 2.93\%$ accuracy, while sources describing the same entities with conflicting facts gave 0% (Table 2; Appendix D).

This difference also explains why an incorrect routing decision does not always lead to an incorrect answer. Before deletion, 31 out of the 33 duplicated source questions that were routed to the wrong source were still answered correctly. None of the 20 misrouted questions in the conflicting fact experiment were answered correctly.

HANKER also distinguishes knowledge unique to the forget set from knowledge shared with retained data when evaluating unlearning (Jiang et al., 2025). These results go to show why defining what a deletion request does is important. If the request is to remove a source's contribution, while allowing other sources to share facts, correct answers may remain after the corresponding lane is removed. If instead the request requires all facts from the deleted source to be removed even if they appear elsewhere in the model, then the other sources containing the same fact must also be considered. The model being able to answer after deletion of a lane does not necessarily mean that the removed lane is still being used.

3.6 Removal can make subsequent learning easier

We next tested whether removing earlier lanes changed the amount of training needed to learn a new source. In a three-seed sweep, we started with eight trained sources and trained an additional

Table 3: Learning a ninth source after deletion, three seeds per condition. Selected comparisons discussed in the text; each percentage is the mean of within-seed step ratios against that training policy's own no-deletion baseline, with a 95% t interval. Earlier lanes remaining exclude the new lane. The cascade follows a four-source request. Different surviving source sets prevent interpreting the two policies as an equal-retention comparison.

Removal	Earlier lanes remaining	Mean steps spent	Steps relative to baseline (%; 95% CI)
Higher dropout for sources designated for possible deletion			
None	8	2133	100 (reference)
Two sources	6	1333	62.7 $[43.4, 82.0]$
Four sources and dependents	0	933	43.6 $[35.8, 51.5]$
Dropout $p = .75$ for all earlier lanes			
None	8	3933†	100 (reference)
Four sources and dependents	2	1867	48.8 $[22.2, 75.4]^\dagger$

†One no-deletion run did not fit within 4800 steps. Its cap enters the mean and paired ratio. These are capped-budget comparisons; the interval does not account for the unknown completion time. All post-deletion runs shown reached the fitting target.

ninth lane after either keeping the earlier-trained ones or removing them. We disabled earlier-trained lanes designated for possible deletion with $p = .75$, and those designated to remain with $p = .25$, including while the ninth lane trained (Appendix C). Under this policy, fitting the new source took an average of 2133 steps with all earlier lanes present, and 1333 steps with two sources removed (Table 3). Across the three-seed sweep, training after deletion took an average of 62.7% of the number of steps that training without deletion did, with a 95% confidence interval of $[43.4\%, 82.0\%]$. We used the same earlier lanes and new-source data for these comparisons, but initialized the new lane separately and used different training seeds for each condition. The runs differed in more ways than just which earlier-trained lanes were removed (Appendix C).

We also tested deleting four sources at once, by using cascade deletion to remove their recorded dependents as well. Training the new lane took an average of 933 steps, averaging 43.6% of the steps needed without deletion across the three seeds, with a 95% confidence interval of $[35.8\%, 51.5\%]$. However, this cascade deletion removed all eight earlier-trained lanes, so the reduction in fitting steps came at the cost of all previously added modules.

When all earlier lanes were trained with module-level dropout with $p = .75$ instead, the cascade deletion left two lanes active. Training the new lane took 48.8% as many steps as without deletion, with a 95% confidence interval of $[22.2\%, 75.4\%]$. Each condition is compared with its own no-deletion baseline since the training conditions leave different sources active after deletion. We note that with $p = .75$ and no lanes removed, the new lane never trained with all earlier trained lanes active, even though we checked if it hit the fitting target with all eight active. Removing lanes made training match the check more often, which may explain the reduction in fitting steps.

One no-deletion run with $p = .75$ did not reach the fitting target within 4800 steps, so we used that cap instead of a known completion time when calculating its cost.

We also estimated training computation, since removing lanes can make each step cheaper as well as reduce the steps needed. After cascade deletion, computation was 42.3% of the no-deletion baseline when dropout varied by source, and 48.2% when all earlier-trained lanes used $p = .75$. These estimates exclude optimizer and evaluation overhead, so they do not measure wall-clock time.

In another single-seed diagnostic, keeping earlier lanes did help without module-level dropout. Training the new lane took 400 steps with earlier lanes active, compared with 1000 after removing them all. Therefore, removing earlier lanes does not guarantee easier or faster learning. Our three-seed results show that deletion can reduce subsequent fitting cost in the tested conditions at the cost of giving up previously learned sources.

Table 4: Five-seed comparison across model sizes, mean ± sample SD. The first four rows show retained accuracy changes after removing the first lane, in percentage points. The last two show accuracy in percent for the same three independently trained lanes under different activation conditions. Selecting each source's own lane gives zero deletion collateral at every size.

Condition	135M	360M	1.7B
Disjoint, $p = 0$	-93.70 ± 3.17	$-98.45 \pm .54$	-96.75 ± 1.70
Disjoint, $p = .25$	-28.65 ± 8.04	-29.40 ± 9.25	-21.75 ± 4.43
Disjoint, $p = .5$	-2.90 ± 2.92	-2.35 ± 2.38	-4.00 ± 2.38
Disjoint, $p = .75$	$+3.65 \pm 2.32$	$+2.05 \pm 2.51$	$+.20 \pm .65$
Independent, own lane	97.47 ± 1.91	$96.57 \pm .67$	94.60 ± 2.67
Independent, all lanes	2.53 ± 1.02	$2.50 \pm .81$	4.27 ± 1.10

3.7 Replication across model sizes

We also tested whether dependence on earlier-trained lanes and module-level dropout's protection persisted with the larger models. Using the same five datasets and training conditions at 135M, 360M, and 1.7B parameters, we found that all 75 runs reached the fitting target. Without module-level dropout, removing the first lane trained reduced accuracy on the retained sources by 93.70 ± 3.17, $98.45 \pm .54$, and 96.75 ± 1.70 pp, respectively (Table 4).

At $p = .75$, removing the first lane trained increased mean retained accuracy at every model size, although the 95% confidence intervals at 360M and 1.7B parameters did include small losses. Reaching the fitting target required 58.3%, 58.5%, and 47.4% more steps than without module-level dropout, respectively. At 360M parameters, pre-deletion accuracy was 2.90 pp lower, with a paired 95% confidence interval of $[-5.44, -.36]$. The corresponding intervals at 135M and 1.7B parameters included zero as well.

Under the independent training condition, we avoided deletion damage at every model size when we selected each source's own lane. Activating those lanes together reduced accuracy to 2.53%, 2.50%, and 4.27%, respectively. Here, increasing model size did not remove the observed dependence or make independently trained lanes additively work together.

It is important to note that these comparisons all use one model family and five matched sets of synthetic sources. These experiments primarily show that the tested effects persist across model sizes, but do not empirically establish a scaling law. Learned routing, answers surviving through duplicate facts, and learning after deletion were not tested across sizes in this experiment.

4 Related Work

Modular learning and removal. Prior work has used separate trainable components to add knowledge while preserving earlier parameters. Adapters and LoRA allow models to learn updates without updating the underlying backbone, while Progressive Neural Networks let new components use the outputs of the earlier, frozen ones (Houlsby et al., 2019; Hu et al., 2022; Rusu et al., 2016). AdapterFusion learns to combine independently trained adapters, and DEMix supports adding, combining, and removing domain-specific experts (Pfeiffer et al., 2021; Gururangan et al., 2022). AdapterSwap already partitions training data across adapters to support continual learning, data removal, and access control (Fleshman et al., 2024). We study whether removing an earlier-trained component damages later components trained with it active, despite their separate parameters.

Training with missing components. Dropout was introduced to reduce co-adaptation by randomly disabling model units during training (Srivastava et al., 2014). Related methods disable larger components of the model, such as LayerDrop, which trains Transformers with layers missing, and AdapterDrop, which removes adapters from lower model layers during training and inference (Fan et al., 2020; Rücklé et al., 2021). GRAM applies a related approach during modular pretraining, where it routes capability gradients to auxiliary modules and trains the model core with different

modules present (Roland et al., 2026). We use this established approach with a frozen model backbone and earlier-trained lanes, measuring the deletion damage, fitting cost, and the dependencies that remain after deletion.

Unlearning and shared knowledge. Machine unlearning methods differ in how they remove the influence of training data on the model. SISA isolates training into shards to minimize the retraining required after deletion, while O^3 trains separate LoRA adapters for sequential unlearning requests, and uses the input to decide each adapter's strength in contributing to the model's answer (Bourtoule et al., 2021; Gao et al., 2025). Benchmarks such as TOFU and MUSE evaluate forgetting alongside performance on retained information (Maini et al., 2024; Shi et al., 2024). HANKER shows why distinguishing facts shared between forgotten and retained data from facts unique to the forgotten data matters when evaluating unlearning performance (Jiang et al., 2025). We test whether surviving lanes can still answer a removed source's questions, and count additional sources removed through cascade deletion as a part of the total reported deletion cost.

Learning after removal. Continual learning also depends on a model's ability to learn new information after earlier training. Dohare et al. study the loss of this ability during continued training and show that replacing features can help preserve learning ability (Dohare et al., 2024). Our experiment asks a related question: does removing earlier-trained frozen lanes change the amount of training a new lane needs? Since the new lane starts with fresh parameters, we do not test whether deletion restores the learning ability of the previously trained parameters. What we measure instead is how the active earlier-trained lanes affect new-source fitting cost, and report which earlier sources are lost when removal makes that fitting faster.

5 Discussion and Limitations

Our experiments show that checking whether a source's parameters were removed is not sufficient to evaluate deletion. We must also check performance on surviving sources, whether they were allowed to preserve the same facts as the removed source, and how the model's learning capability changes afterwards. We report these outcomes together because reducing deletion damage or making later learning faster can require more training or removing additional sources.

We use synthetic datasets and rank candidate answers to questions seen during training. This lets us control fact sharing and measure deletion damage, but does not indicate model performance on held-out questions or freeform text generation. The model-size comparison uses one model family and excludes routing, shared-fact experiments, and post-deletion learning. Our results apply to the tested lanes and training conditions, not all modular architectures. Lane removal also does not establish that its source no longer influences the parameters learned by later lanes.

We measure fitting steps and estimated computation, not wall-clock time. One no-deletion run did not reach the fitting target within its cap, so its cost comparison uses that cap, not a known completion time. Earlier code also left some training randomness uncontrolled by the recorded seed, so without the original random state, we could not recreate the original checkpoints or reproduce the entire earlier experiment. We report these experiments separately from the primary and matched-scale results (Appendix E).

6 Conclusion

We found that keeping source-specific parameters separate and frozen does not prevent later-trained modules from depending on their outputs. In our experiments, removing an earlier module could damage model performance on sources we intended to retain. Module-level dropout reduced this damage at the cost of more fitting steps, while cascade deletion removed recorded dependents at the cost of additional sources. We also found that removal could make learning a new source easier, although this improvement came at the cost of losing previously learned modules. Separating parameters does make source removal straightforward, but its consequences depend on what later learned modules use from those parameters and what surviving sources are expected to preserve.

AI USE STATEMENT

We used generative AI tools for literature discovery, experimental design and critical review, implementation, debugging, experiment orchestration, synthetic data generation, data cleaning, figure preparation, and manuscript drafting. We reviewed AI-assisted work, checked citations against primary sources, and checked reported results against the experimental artifacts. We take responsibility for the final text, claims, analyses, code, and other artifacts.

ETHICS STATEMENT

The primary experiments use synthetic biographies with invented names and assigned facts. Our results concern the removal of source-specific modules and its measured effects, and do not establish complete erasure of a source's influence or compliance with legal deletion requirements. In particular, retained sources may preserve shared facts, and cascade deletion may remove sources beyond the original request. These limitations should be considered when interpreting modular removal as a privacy or access-control mechanism.

REPRODUCIBILITY STATEMENT

We provide the code, datasets, training settings, and saved results for the experiments listed in Appendix E; trained checkpoints will be released after the review period due to size constraints. We describe our seed selection process, accuracy checks before and during training, and how we compared results across seeds in Appendix A. The $p = 1$ and eight-source experiments are described in Appendices B and C.

Appendix E describes our checks on the earlier training and evaluation code and what we could and could not reproduce.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. SmolLM2: When smol goes big—data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025. URL https://arxiv.org/abs/2502.02737.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy*, pp. 141–159, 2021. URL https://arxiv.org/abs/1912.03817.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632: 768–774, 2024. doi: 10.1038/s41586-024-07711-7. URL https://doi.org/10.1038/s41586-024-07711-7.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.11556.

William Fleshman, Aleem Khan, Marc Marone, and Benjamin Van Durme. AdapterSwap: Continuous training of LLMs with data removal and access-control guarantees. *arXiv preprint arXiv:2404.08417*, 2024. URL https://arxiv.org/abs/2404.08417.

Chongyang Gao, Lixu Wang, Kaize Ding, Chenkai Weng, Xiao Wang, and Qi Zhu. On large language model continual unlearning. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.10223.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. DEMix layers: Disentangling domains for modular language modeling. In *Proceedings of the 2022*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 5557–5576. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.naacl-main.407. URL https://aclanthology.org/2022.naacl-main.407/.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pp. 2790–2799, 2019. URL https://arxiv.org/abs/1902.00751.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Weipeng Jiang, Juan Zhai, Shiqing Ma, Ziyan Lei, Xiaofei Xie, Yige Wang, and Chao Shen. Holistic audit dataset generation for LLM unlearning via knowledge graph traversal and redundancy removal. *arXiv preprint arXiv:2502.18810*, 2025. URL https://arxiv.org/abs/2502.18810.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. TOFU: A task of fictitious unlearning for LLMs. *arXiv preprint arXiv:2401.06121*, 2024. URL https://arxiv.org/abs/2401.06121.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rucklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, 2021. URL https://aclanthology.org/2021.eacl-main.39/.

Ethan Roland, Murat Cubuktepe, Erick Martinez, Stijn Servaes, Keenan Pepper, Mike Vaiana, Diogo Schwerz de Lucena, Judd Rosenblatt, Addie Foote, Cem Anil, and Alex Cloud. Modular pretraining enables access control. *arXiv preprint arXiv:2607.08077*, 2026. URL https://arxiv.org/abs/2607.08077.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7930–7946. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.626. URL https://aclanthology.org/2021.emnlp-main.626/.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. URL https://arxiv.org/abs/1606.04671.

Weijia Shi, Jaechan Lee, Yangsibo Huang, Sadhika Malladi, Jieyu Zhao, Ari Holtzman, Daogao Liu, Luke Zettlemoyer, Noah A. Smith, and Chiyuan Zhang. MUSE: Machine unlearning six-way evaluation for language models. *arXiv preprint arXiv:2407.06460*, 2024. URL https://arxiv.org/abs/2407.06460.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL https://jmlr.org/papers/v15/srivastava14a.html.

A TRAINING CHECKS, SEED SELECTION, AND UNCERTAINTY

A.1 SEED SELECTION AND REPEATED RUNS

Before running the primary experiment, we fixed the candidate seeds as 101 through 120, testing them in order using the pre-training accuracy check described in Experimental Setup. Seeds 101 through 104, 108, and 109 all failed this check. The first five seeds to pass were 105, 106, 107, 110, and 111, and we stopped searching for additional seeds after obtaining these ones. Since this all happened before training started, seed-specific training performance was not a factor in our decision.

For each selected seed, we used the same datasets for all four module-level dropout probabilities $p = \{0, .25, .5, .75\}$ as well as for independent training. All 25 runs reached the fitting target without technical failures, and we then repeated these conditions at 360M and 1.7B parameters on the same datasets, giving us a total of 75 completed runs. The primary experiment's results on the 135M model size were reused in the model-size comparison results as well.

Prior to running the primary sweep, we also ran the $p = .75$ condition twice in separate processes with the same datasets to check whether the results were the same. They produced identical lane parameters, fitting histories, stopping steps, and evaluation results. Since this was a repeated run, we did not count this as an additional seed in our calculations.

The model revisions are pinned in the release manifests. The Hugging Face checkpoints and revisions are:

- `HuggingFaceTB/SmolLM2-135M`: `93efa2f097d58c2a74874c7e644dbc9b0cee75a2`.
- `HuggingFaceTB/SmolLM2-360M`: `f8027fd0eaeea54caa13c31d31b9fdc459c38b49`.
- `HuggingFaceTB/SmolLM2-1.7B`: `effd688a12921b4cc83e3312b6feb579f70f9c71`.

We also saved the source code hashes, training configurations, and environment information alongside the experiment records. Although these repeated runs did produce identical results, they do not explain the difference between the historical runs and the current-code reruns, which we discuss further in Appendix E.

A.2 LANE IMPLEMENTATION AND FITTING

Each lane uses two trainable matrices at each decoder block, with the first one projecting the feed-forward input into a lower-dimensional space, and the second one projecting it back into the model's hidden dimension. We add this result to the frozen feed-forward output. For source j at layer ℓ, we write the lane's output as

$$L_{j,\ell}(h) = \frac{\alpha}{r} B_{j,\ell} \operatorname{Dropout}_{.05}(A_{j,\ell} h), \tag{2}$$

where h is the feed-forward input, $A_{j,\ell}$ and $B_{j,\ell}$ are the two trainable matrices, r is the low-rank dimension, and α controls the scaling factor through $\frac{\alpha}{r}$. We use $r = 32$ and $\alpha = 32$, with no nonlinear activation between the two matrices.

At each layer, all active lanes receive the same feed-forward input independently, and we add each lane's output to the backbone's feed-forward output. Although their outputs are not shared within the same layer, their combined output changes the computation that reaches deeper layers, where later-trained lanes can learn to depend on and use those outputs.

We initialize the first matrix with Kaiming-uniform weights and the second to zero, so a new lane initially does not change the model's output. We keep lane parameters and their optimizer state in FP32 and update only the lane corresponding to the source being trained. For module-level dropout, we choose which earlier-trained lanes to disable once per training batch, while keeping the current lane active. This is different from the .05 dropout applied between A and B.

We use the AdamW settings described in Experimental Setup and clip the gradient norm to a maximum of 1 before each optimizer update. We check fitting accuracy every 200 steps and stop once it

reaches 90%, with a step cap of 2400 per lane. Our reported fitting steps identify when a run passes these checks, not the exact step when the accuracy reached the fitting target. If a run does not end up reaching the target accuracy within the step cap, we report that run as a failure and do not try another seed. The larger caps used for the eight-source experiments are described in Appendix C.

A.3 Statistical comparisons

We report the mean and sample standard deviation across seeds for each training condition. When comparing training conditions, we first calculate the difference between their results for each seed since the same datasets are used within seeds, and we then average the differences and calculate a 95% confidence interval:

$$\bar{d} \pm t_{.975,n-1} \frac{s_d}{\sqrt{n}}, \tag{3}$$

where $\bar{d}$ is the mean difference, s_d is the sample standard deviation of the differences, and n is the number of seeds. The term $t_{.975,n-1}$ is the Student's t critical value used for a two-sided 95% interval. We use five seeds for both the primary and matched-scale experiments and three for the historical experiments with shared information and post-deletion learning unless stated otherwise.

We use seeds, not individual questions, to calculate these intervals. This is because each run answers many questions, including five versions of each factual question, so these measure one trained model's performance instead of separate training repetitions.

For the experiment where we train a new lane after deletion, we divide the fitting steps after deletion by the fitting steps without deletion per seed. We then report the mean of these ratios, and calculate its confidence interval across seeds. This can differ from simply dividing the mean post-deletion steps by the mean no-deletion steps. The one run that did not reach the fitting target is included using its step cap, as described in Appendix C.

When the confidence interval for a difference includes zero, we cannot conclude from the runs that any one condition outperforms the other. This does not mean that they perform equally, and is different from observing exactly zero change in accuracy in every tested seed, as we did observe when removing independently trained lanes while evaluating each surviving source with only its corresponding lane active.

A.4 Seed selection in the earlier experiments

We also checked which runs were excluded from the earlier three-seed experiments. Of the total 71 attempted runs, 32 were rejected by the pre-training accuracy check, and 39 completed training. None of the runs that passed the pre-training check failed to reach the fitting target. For each dataset type and seed, the same sources were rejected across the training conditions, so all training conditions got datasets that passed our criteria.

These experiments used seeds 30, 33, and 34 for sources without overlapping facts; 32, 33, and 34 for the duplicated sources; and 33, 34, and 35 for conflicting facts as well as partial source overlap. During seed selection, we also ran ten additional checks that tested the backbone's accuracy prior to training. All of them passed, but they did not train any lanes and were not counted as additional experimental repetitions.

These counts specifically describe earlier experiments and not the 75-run comparison, eight-source experiments, or separate $p = 1$ diagnostic. The five-seed results are used for the primary comparison, while the earlier experiments with shared information are reported separately.

B Independent acquisition at full module-level dropout

We also tested module-level dropout with $p = 1$, where every earlier-trained lane is disabled while the new lane trains. The original runs did not reach the fitting target for the second lane trained when evaluated with the first lane active. We therefore tested whether the second lane had failed to learn its source or if activating the first lane was what caused the drop in accuracy. This was a separate three-seed diagnostic sweep using the 135M-parameter model and seeds 30, 33, and 34, and was not included in the primary comparison.

Since the original runs stopped before saving the second lane's checkpoint, we had to train it again in two runs with the same training settings but different evaluation settings, starting from the same saved state. Both conditions used $p = 1$ during training. At evaluation time, one activated the newly trained lane, while the other activated the earlier-trained one as well. We disabled early stopping and trained both conditions for 4800 steps, recording accuracy at both the original 2400-step cap as well as the 4800-step cap (Table 5).

Table 5: Accuracy (%) for the $p = 1$ development diagnostic, where learned lane parameters were identical between the two conditions within each seed. Only active lanes differed during evaluation. These runs are separate from the primary comparison.

	2400 steps		4800 steps	
Seed	Lane alone	Both lanes	Lane alone	Both lanes
30	100.00	10.00	100.00	17.00
33	100.00	15.50	100.00	21.50
34	100.00	16.50	100.00	24.00

We initially compared the complete checkpoint files and found that they differed. Upon further investigation, we found that the files contained evaluation histories and information about active lanes during evaluation which were different across conditions. Comparing the learned parameters directly showed that every stored parameter value matched within each seed, along with the recorded training losses, gradient norms, and learning rates.

With only the new lane active, accuracy reached 100% across all seeds. With the earlier lane active, accuracy remained between 10 and 16.5% after 2400 steps, and after 4800 steps was between 17 and 24%. The new lane had learned its source, but adding its output to the earlier-trained lane's output reduced performance below the fitting target even with double the training budget. This demonstrates a failure to combine independently trained lanes with just ungated addition.

C Eight-source deletion and subsequent learning

C.1 Sources, deletion policies, and dropout probabilities

For the eight-source experiments, we used the 135M-parameter model and seeds 140, 142, and 144, all of which passed the pre-training accuracy check. Each set contained eight trained sources, and one source held out from training. Sources 0 and 1 contained the exact same training texts, as did sources 2 and 3. Sources 4 to 7 did not contain any overlapping facts.

Sources 0, 3, 4, and 7 were designated for possible deletion, while sources 1, 2, 5, and 6 were designated to survive. This way, both groups had sources with overlapping and non-overlapping facts. This allowed us to vary whether a source was designated for deletion separately from whether its facts were shared with another source.

During each lane's training, we disabled earlier-trained lanes designated for deletion with probability $p = .75$, and earlier-trained lanes that were designated to survive with probability $p = .25$. The probability depended on the earlier lane's designation only, not whether its facts overlapped with the currently learning source. For an earlier-trained lane i and a later training lane j, this gives

$$p_{ij} = \begin{cases} .75, & i \in \{0, 3, 4, 7\}, \\ .25, & i \in \{1, 2, 5, 6\}, \end{cases} \qquad i < j. \tag{4}$$

We also tested setting p to the same value for all earlier lanes, comparing these conditions on the same eight-source datasets.

For this comparison, we removed each source designated for possible deletion separately, and measured the accuracy drop on each source trained after the removed source. If removing a lane improved accuracy on a later trained source, we recorded that as zero damage. We then averaged recorded damage across all tested removals and later sources within the seed. The reported mean and confidence interval are calculated across the three seeds.

Table 6: Dependency graphs in a separate historical eight-source experiment, three seeds. Functional edges require a five-percentage-point accuracy drop when a lane is disabled. Additional sources removed are averaged over single-source requests and seeds. Each exposure graph has 28 edges and removes 3.50 additional sources per request on average. The $p = .75$ condition uses an extended 4800-step cap. Oracle adaptive assigns different dropout probabilities by source policy (Appendix C).

Policy	Functional edges	Exposure edges retained (%)	Additional sources removed
Uniform $p = 0$	18.3	65.5	2.50
Uniform $p = .5$	3.7	13.1	.50
Uniform $p = .75$	2.0	7.1	.25
Oracle adaptive	7.0	25.0	1.62

With $p = .75$ for all earlier-trained lanes, none of the three runs fitted all eight sources within the per-lane 2400-step cap. Using different dropout probabilities p by source did complete training, although mean deletion damage was $5.90\,\mathrm{pp}$ with a 95% confidence interval of $[4.65, 7.16]$. This exceeded our $5\,\mathrm{pp}$ limit set before training started, while the $p = .75$ experiments also failed the fitting requirement, so we could not provide the planned comparison.

We then doubled the per-lane step cap to 4800 for the $p = .75$ condition for all earlier-trained lanes. All three runs reached the fitting target under this budget, with mean deletion damage of $5.75\,\mathrm{pp}$ and a 95% confidence interval of $[4.60, 6.90]$. We did not find that using different dropout probabilities by source reduced damage more than just using $p = .75$ for all lanes. We used these same successfully trained lanes for the post-deletion learning experiments with $p = .75$.

C.2 Comparing direct and cascade deletion

We tested removing sources in the order 0, 7, 2, and then 5. Sources 0 and 7 were designated for possible deletion, while sources 2 and 5 were designated to survive. This allowed us to test deletion requests that followed the original deletion policy as well as ones that did not. For cascade deletion, we used the functional graph with $\tau = .05$ to find and remove any lanes recorded either directly or indirectly as dependent on the requested sources.

Since cascade deletion can remove sources beyond the initial request, we compared it with direct removal on the sources that survived both deletion conditions. We evaluated those same sources before deletion as well. Otherwise, removing a poorly performing source could increase the average accuracy without improving performance on any of the surviving sources. If the cascade removed all sources, we marked the accuracy on retained sources as undefined.

After all four deletion requests, cascade deletion removed all eight lanes in every seed when p differed per source. With $p = .75$ for all earlier-trained lanes, it removed six lanes. On the two sources that survived, both direct removal and cascade deletion increased accuracy by $6.00\,\mathrm{pp}$ compared with before deletion. Removing additional lanes through cascade deletion did not give any further accuracy benefit on the surviving sources.

With $p = .5$ for all earlier-trained lanes, direct removal reduced accuracy on the surviving sources by $2.75\,\mathrm{pp}$, while cascade deletion reduced it by $15.17\,\mathrm{pp}$. Therefore, cascade deletion did not consistently improve performance on the remaining sources in these experiments, despite removing the recorded dependent sources as well.

C.3 Fitting the ninth source

We next tested training a ninth lane on the source that was originally held out from training. We used the same examples and optimizer settings across training conditions within each seed, and kept the model backbone and earlier-trained lanes frozen.

We trained this new lane with either all eight of the earlier-trained lanes active, sources 0 and 7 being removed, sources 0, 7, 2, and 5 being removed, or those four sources being removed along with the

sources recorded as dependent on them. During training, the surviving lanes used the same dropout probabilities p as before deletion, while the removed lanes stayed disabled. We used the same new-source data across these conditions, but initialized the new lane separately each time. As the training seed was set after initialization and differed between conditions, removing earlier-trained lanes was not the only difference between the training runs. The new lane had a 2400-step cap when p differed by source, and a 4800-step cap when $p = .75$ for all the earlier-trained lanes.

Table 7 shows the fitting steps and number of earlier lanes remaining under each condition. We compare each removal condition with its corresponding no-deletion baseline.

Table 7: Training needed to fit the ninth source across three seeds. We report the mean fitting steps, rounded to whole steps, and the number of earlier-trained lanes remaining. For each seed, we divide fitting steps by those without deletion under the same dropout policy, then average these ratios. Starred values use the 4800-step cap for a single no-deletion run that did not reach the fitting target.

Earlier training	Removal	Lanes left	Mean steps	Mean ratio
Source-specific p	None	8	2133	1.000
	Two requested	6	1333	.627
	Four requested	4	1333	.627
	Four plus cascade	0	933	.436
Uniform $p = .75$	None	8	3933*	1.000
	Two requested	6	3000	.786*
	Four requested	4	2267	.597*
	Four plus cascade	2	1867	.488*

When p differed by source, training after removing two sources took an average of 62.7% of the steps needed without deletion, with a 95% confidence interval of $[43.4\%, 82.0\%]$. After cascade deletion, this dropped to 43.6%, with a 95% confidence interval of $[35.8\%, 51.5\%]$, although cascade deletion removed all eight earlier-trained lanes here. When $p = .75$ for all eight earlier-trained lanes, two lanes survived, and training took 48.8% of the steps needed without deletion, with an interval of $[22.2\%, 75.4\%]$.

One no-deletion run, on seed 144 with $p = .75$, did not reach the fitting target within 4800 steps. We kept this seed and used the step cap in our calculations, as we do not know how many steps it would have needed in total. All other training conditions with deletion on that seed did reach the target.

C.4 Computation estimate and interpretation

We also estimated the matrix-multiplication computation needed to train the new lane, since removing earlier-trained lanes could make each step cheaper as well as reduce the total number of fitting steps. We calculated this by using the cost of a forward pass through the model backbone, F_B, the cost of a forward pass through one lane, F_L, and the average number of earlier-trained lanes active during training, $\bar{k}$. Including the estimated backward-pass costs gives

$$C_{\text{step}} = 2F_B + 3F_L + 2\bar{k}F_L. \tag{5}$$

Although the model backbone and earlier-trained lanes are frozen, training still requires passing gradients through them in order to update the new lane. We estimate their forward and backward computation as double their forward-pass cost. For the new lane, we also calculate gradients for its trainable parameters, giving an estimated cost of three times its forward pass.

We multiplied the estimated computation per step by the total number of fitting steps, and then compared deletion against no deletion within each seed. After cascade deletion, the mean computation was 42.3% of that needed without deletion when p differed per source, and 48.2% when all the earlier-trained lanes used $p = .75$. These estimates exclude optimizer updates, evaluation, and data loading, and do not account for differences in hardware utilization. They estimate computation, not measured training time.

D ROUTING AND ANSWERS SUPPORTED BY RETAINED SOURCES

For the routing experiments with overlapping sources, we used seeds 33, 34, and 35 for conflicting facts and partial overlap, and seeds 32, 33, and 34 for duplicated sources. We trained a router to select which lane to activate for each question, keeping the model backbone and the trained lanes frozen. Every prompt began with a tag naming its source (`Context source: source_k`), and the router received the full prompt so it did not need to infer the source from the content alone.

The router used the model backbone's representations of the question as input to an MLP with hidden width of 256. We used dropout $= .1$, learning rate $= 3 \times 10^{-3}$, and a 1000-step cap. We used 640 questions for router training, with 160 held out to check how often it selected the intended lane. This split is separate from the lane-fitting check, as these questions were held out from router training, not lane training. The held-out questions could refer to entities also present during training, so this did not test routing for entirely new entities.

We checked the router's source-selection accuracy on these held-out questions every 25 steps, with a target accuracy of 98%. All routers reached 100% training accuracy, but none reached the validation target within the step cap. We report both how often the router selected the intended source, and how often the model answered the question correctly in Table 8, since selecting another source may lead to the same answer if facts are shared between sources.

Table 8: Routing and answer accuracy on held-out questions, reported as mean $\pm$ sample standard deviation across a three-seed sweep. Correct source measures how often the router selected the intended lane. Oracle task measures answer accuracy (%) when we select the intended lane manually. Routed task uses the router's selection. None of the routers reached the 98% validation target.

Source relationship	Correct source	Oracle task	Routed task
Conflicting facts	$95.83 \pm .72$	98.33 ± 1.80	94.17 ± 2.53
Duplicated sources	93.12 ± 1.25	$95.42 \pm .95$	95.62 ± 1.08
Partial overlap	$95.83 \pm .72$	95.62 ± 1.08	94.17 ± 1.57

We then removed each tested lane and prevented the router from selecting it. To check whether that lane's answers were still available, we evaluated the removed source's questions using the surviving lanes. We tested three deletion positions separately within each seed, took the average of their recovery accuracies, and then calculated the mean and standard deviation across seeds.

The three source relationships tested were conflicting facts, where accuracy was 0%; duplicated sources, where accuracy was $89.17 \pm 2.50\%$; and partial overlap, where accuracy was $41.39 \pm 2.93\%$. The requested lane was disabled in each case, but if surviving lanes contained the same information they could answer the question. As a result, these results measure whether answers remain available after removal, not whether the parameters were fully removed. We cannot establish from these results that the later-trained parameters contained no influence from the removed source.

E HISTORICAL RERUNS AND ARTIFACT PROVENANCE

We tried to reproduce an earlier experiment with non-overlapping sources using seed 30 and $p = .75$ with the current code. We repeated this experiment three times, and all three runs produced identical reported metrics and fitting steps. However, they differed from the original run's results, as shown in Table 9. Since these runs used the same seed, the difference from the original run was consistent across the three repeats.

We then checked why using the same seed had produced different results. In the earlier code, the seed controlled the dataset and which lanes were disabled during training, but not the lane parameter initialization or the order of the training examples. This meant that some parts of training could differ on a run-to-run basis even with the same seed. When we controlled these as well, the later archived code and current code produced identical lane parameters in a comparison training the first lane only.

However, we did not save this random state in the original run, so we cannot use it to recreate the original checkpoints. We also compared the earlier and current evaluation code on 24 questions,

Table 9: Comparison between the original run and three repeated runs using the current code, all with seed 30 and $p = .75$. Each repeat produced the values shown in the right column. Collateral measures the change in retained accuracy after deletion. These repeats are not additional seeds in the reported experiments.

Measurement	Original run	Each current rerun
Active accuracy (%)	96.00	97.67
Delete-first collateral (pp)	+1.00	−.75
Delete-first-two collateral (pp)	+6.50	+3.50
Total fitting steps	3600	3800

where one prediction differed. In doing so, we identified a problem with how the earlier code controlled randomness, but did not reproduce the full original run. The five-seed sweep is the primary experiment, and we keep the earlier results separate.

The new primary experiment gave the $93.70\,\mathrm{pp}$ accuracy drop and 58.3% increase in fitting steps. The current matched-scale experiment extended this comparison to the 360M and 1.7B parameter model sizes using the same five datasets. The earlier experiments with shared information, routing, and eight sources were not repeated as part of this confirmation.

E.1 Artifact index

We provide an anonymized supplementary package with two parts. The `code/` directory contains a self-contained implementation of the primary and matched-scale experiments, together with the recorded outputs of all 75 runs. It keeps only the code used in those runs and depends only on PyTorch, Transformers, and tqdm. We checked it against the original research code: it regenerates all eleven candidate datasets byte for byte, its aggregation script reproduces every value in Tables 1 and 4 from the recorded outputs, and running both implementations side by side on SmolLM2-135M (on CPU, with a reduced dataset) gave bitwise-identical lane parameters, fitting histories, and evaluation outputs. The `historical/` directory contains the records of the earlier experiments and the original research code used for them. That code was not preserved separately for each run, so it documents their methods but is not provided as a runnable release.

We will release the trained lane checkpoints after the review period due to their size. The recorded outputs in `code/results/` are sufficient to recompute the primary and matched-scale experiment results, and `code/modular_deletion/reevaluate.py` evaluates the saved lanes including those that were produced by rerunning the code. We do not include the pretrained base model weights but do specify the model versions so that these can be downloaded separately. All paths below are relative to the extracted supplementary package.

- Primary and matched-scale experiments: `code/results/smollm2_135m/`, `code/results/smollm2_360m/`, and `code/results/smollm2_1p7b/`, including seed selection (`discovery/`) and the repeated-run checks (`acceptance.json`). `code/scripts/aggregate.py` recomputes the reported summaries, and `code/scripts/run_experiments.sh` reruns all 75 runs.
- Seed selection in the earlier experiments: `historical/records/seed_selection/`.
- Sequentially merged LoRA adapters: `historical/records/merged_lora/`.
- Sources with shared information and the earlier disjoint sweep: `historical/records/correlated_sources/`.
- Dependency graphs: `historical/records/dependency_graphs/`.
- Eight-source policies, cascade deletion, and new-lane fitting: `historical/records/eight_source/`, with protocols in `historical/protocols/`.
- Full-dropout diagnostic: `historical/records/full_dropout_p1/`.

- Routing and answers supported by retained sources: `historical/records/routing/`. `historical/code/recompute_recovery_by_seed.py` computes the recovery values in Table 2 from the seed-level records.
- Historical reruns and reproduction audit: `historical/records/reproduction_audit/`.