

Machine Unlearning: Can we make a trained ML model “forget” specific data without retraining from scratch?

Tracy Cui
Boston University

Yuki Li
Boston University

Yang Lu
Boston University

Xin Wei
Boston University

Abstract

Machine Learning models increasingly train on user-generated data, creating a tension between model utility and user privacy when individuals request deletion of their records. Retraining from scratch after every deletion is the gold standard for correctness, but it is often too expensive to support at scale, especially under heavy-tailed workloads where a small number of users contribute a disproportionate share of data. Based on this situation, we built an end-to-end unlearning pipeline that treats user deletion as a first-class operation, producing a retain/forget split and updating the model using either a full retraining or an efficient approximation based on Fisher Scrubbing. Our results show that efficient unlearning can substantially reduce deletion-time cost while approximating the behavior of full retraining, with performance depending on how cleanly the deleted data aligns with model concepts and how concentrated user ownership is under realistic workloads.

1 Introduction

Modern ML systems are trained on large collections of user-contributed data, and as a result, they face a practical requirement that is easy to state but hard to implement: when a user asks to delete their data, the model should behave as if that user’s records were never used for training. This matters for privacy, compliance, and user trust, but it also presents a system challenge. The most efficient solution is full retraining on the remaining data, yet retraining is costly and can be infeasible when deletions are frequent, when datasets are large, or when user contributions follow a heavy-tailed distribution that makes “heavy-users” disproportionately expensive to remove.

Our work addresses this gap by designing an end-to-end unlearning pipeline that starts from a clear abstraction: a deletion request induces two sets of training data, the forget set(owned by the user) and the retrain set(all permitted data). On top of this contract, we compare two post-deletion update strategies. The first is full retraining(M0), which serves as a correctness reference. The second is Fisher Scrubbing(M2), which

updates the pre-deletion model using constrained parameter steps that aim to eliminate usable signal on the forget set while protecting performance on the retain set. A key idea in our design is to define forgetting as making the model uninformative on deleted inputs by pushing predictions toward a uniform distribution, rather than forcing specific misclassifications, which yields a stable objective for scrubbing.

To make the evaluation meaningful, we explicitly test two deletion regimes. In a mixed ownership setting, a user’s data spans multiple classes, which reflects realistic usage but can make “forget accuracy” hard to interpret because class-level features remain supported by retained data. In an exclusive-class setting, the deletion target aligns with a single concept, making forgetting easier to verify and reducing ambiguity in the metric. We also incorporate a Zipfian heavy-tailed ownership workload to reflect that deletion cost is often dominated by a small number of heavy users. The main limitation is that Fisher Scrubbing is an approximate method: it can trade retained accuracy for speed, especially under heterogeneous mixed deletions, and our accuracy-based forgetting measures may not capture all forms of memorization or privacy leakage beyond predictive performance.

2 Background

This project builds upon recent advancements in two critical areas: privacy-preserving data management and machine unlearning. Our core objective is to demonstrate the practical application of machine unlearning techniques in a scenario where data privacy is enforced at the database level.

2.1 Privacy-Compliant Data Storage: K9db

Our simulated data pipeline is inspired by K9db: Privacy-Compliant Storage For Web Applications By Construction (Albab et al., 2023). The K9db paper proposes a novel approach to data management where user data is inherently tied to user accounts through “physical ownership” and “cascading deletion.” This means deleting a user record automatically

triggers the deletion of all associated data, ensuring strong privacy guarantees by construction. Our K9dbMock class emulates this behavior, linking images to owner IDs and enabling automatic deletion.

2.2 Machine Unlearning: Fisher Scrubbing

The machine unlearning component of this project is based on Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks (Golatkar et al., 2020). This paper introduces "Fisher Scrubbing" as an efficient method for machine unlearning.

Machine unlearning aims to remove the influence of specific data from a trained model, as if it had never seen that data. Fisher Scrubbing achieves this by:

- **Estimating Data Importance:** It computes the diagonal of the Fisher Information Matrix (FIM) for the retained set (data not to be forgotten). This identifies important parameters that should be preserved.
- **Targeted Forgetting:** During unlearning, the model is exposed to the forget set (data to be removed). The objective is to make the model output uniform probability distributions for the forget set (making the model uncertain about it) while constraining updates based on Fisher information. This "scrubs" the forgotten data's influence without significantly impacting knowledge from the retained data.

Our implementation uses KL Divergence to a uniform distribution as the forgetting objective, combined with Fisher-weighted gradient updates.

3 Design

3.1 Design Description

Our system is designed to support user-level deletion for a trained model. From the end user's perspective, the workflow is simple: a user requests "delete my data," and the system returns an updated model that no longer relies on that user's training records. From an application developer's perspective, using the system requires only one commitment at ingestion time: every training record must be associated with an owner identifier (`user_id`). Once ownership is defined, a deletion request is expressed as a single operation on a `user_id`, and the system handles the rest.

The central abstraction in our design is the retain/forget split contract. Given a deletion request for a specific `user_id`, the system deterministically constructs two disjoint sets: a forget set containing all records owned by that user, and a retain set containing all remaining records that are still permitted for training. All downstream model update logic consumes only these two sets, which makes the design modular and auditable. Any storage backend or application can plug into the same

pipeline as long as it can materialize the retain and forget sets for a given user deletion request.

On top of this contract, the system exposes a post-deletion model update strategy abstraction. We support two strategies to make correctness and efficiency explicit. Full retraining (M0) is the gold-standard reference: we train a fresh model using only the retained set, which approximates the ideal outcome of "as if the deleted data never existed." Fisher Scrub (M2) is the efficient alternative: it starts from the pre-deletion model and applies a constrained parameter update intended to reduce the model's usable signal on the forget set while preserving performance on the retain set. This separation is a design choice to keep the system practically useful (fast unlearning) while still being evaluable against a clear correctness baseline (retraining).

A key idea in our design is how we define "forgetting" in a stable and interpretable way. Rather than forcing the model to output a particular incorrect label for deleted samples, we define forgetting as making the model uninformative on the forget set. Operationally, the unlearning objective pushes the model's predictions on forget-set inputs toward a near-uniform distribution over classes, which corresponds to maximum uncertainty. This design aligns with the intuition that deleted records should not remain recognizable, and it avoids unstable objectives that can lead to extreme or noisy parameter updates.

Because our task has only 10 discrete output classes, we also design the evaluation regime to avoid a misleading interpretation of forget-set accuracy. If we form the forget set by randomly deleting samples across all classes (a mixed deletion), the model can still show high accuracy on the forget set simply because the retained data continues to support the same class-level features. In this setting, a high forget accuracy does not necessarily imply the system failed; it may reflect that the deletion target is not a clean concept, and accuracy is an insensitive metric. To address this, we include two complementary deletion regimes by design: a mixed ownership regime that reflects realistic users with heterogeneous data, and an exclusive-class regime where the deleted user corresponds to a single class (digit 0), creating a clean concept-removal target where forgetting is directly interpretable.

Finally, we design for workload realism by modeling user ownership as heavy-tailed (Zipfian). In real applications, deletion cost is dominated by a small number of heavy users who contribute disproportionately many records. The Zipfian assumption makes the system's behavior meaningful under the regimes that matter operationally, and it motivates runtime-focused tests such as scaling total dataset size, increasing deletion magnitude, and deleting top-K heavy users. The concrete database mechanics and training procedures that realize these abstractions are described in the Implementation section; the design here specifies how users and developers interact with the system, and the key abstractions and algorithmic choices

that make deletion requests both meaningful and testable.

3.2 Data Infrastructure with K9dbMock

Central to simulating a privacy-centric architecture is the K9dbMock class. The in-memory SQLite database serves as a simplified substitute for a real-world privacy-preserving database system. Its design is predicated on the concept of “physical ownership”, where data records (individual MNIST images) are explicitly tied to `user_id`. A critical feature of K9dbMock is its support for cascading deletions, enabled by SQL foreign key constraints (ON DELETE CASCADE). This ensures that when a user entry is removed from the users table, all associated data points in the images table are automatically and irrevocably deleted, providing a robust mechanism for data removal requests.

Data distribution with K9dbMock mirrors real-world scenarios. Upon initialization, the entire MNIST dataset is loaded, and its indices are distributed among a configurable number of `total_users` following a Zipfian distribution parameterized by `zipf_param`. This process realistically allocates data unevenly, where a small subset of users holds a disproportionately large share of the dataset. K9dbMock exposes interfaces such as `get_full_dataset()`, `get_data_map()`, and `execute_delete_command(user_id)` to manage the dataset, map user identifiers to their associated data, and process data deletion requests, respectively.

3.3 Fisher Scrubbing Algorithm

The core of our unlearning mechanism is an enhanced Fisher Scrubbing algorithm, instantiated through two primary utility functions:

1. **`compute_fisher_diagonal(model, loader, num_samples)`**. This function estimates the diagonal components of the Fisher Information Matrix (FIM). The FIM provides a second-order approximation of the curvature of the loss surface and, in this context, quantifies the importance of each model parameter with respect to the retain set. The function operates by computing and accumulating the squared gradients of the cross-entropy loss over a subset of the retained data specified by `num_samples`. The resulting `fisher_diag` serves as a critical scaling factor in subsequent parameter updates, ensuring that parameters essential for preserving information about the retained data are minimally modified during the unlearning process.
2. **`fisher_scrubbing_step(model, inputs, labels, fisher_diag)`**. This function performs a single unlearning step on a batch of data from the forget set. Unlike conventional unlearning strategies that maximize the loss on forgotten data and can introduce instability, our approach employs a KL-divergence-to-uniform objective. For each forgotten sample, the model’s

output probabilities (computed via `F.log_softmax`) are compared against a uniform target distribution (e.g., $[0.1, \dots, 0.1]$ for a 10-class problem). By minimizing the KL divergence to this uniform target, the model is driven toward equal uncertainty across all classes for the forgotten data, thereby erasing sample-specific knowledge. The resulting gradient is applied to update the model parameters, with updates scaled by $(1.0/(F_{ii} + \alpha))$, where F_{ii} denotes the corresponding diagonal element of the Fisher Information Matrix and α (configured as `CONFIG["alpha"]`) controls the regularization strength. A `clamp_` operation is additionally applied to parameter updates to ensure numerical stability.

3.4 Experimental Protocols

Our experimental methodology involves a rigorous comparison against established baselines:

- **Initial Model Training:** A `base_model` is initially trained on the complete dataset (all MNIST digits). This model serves as the starting point from which unlearning operations are performed.
- **M0 (Full Retraining):** This acts as the “gold standard” for unlearning. A new model (`m0_model`) is trained from scratch exclusively on the retain set (i.e., the original dataset with the data to be completely removed). Its performance and training time represent the ideal outcome for unlearning.
- **M2 (Fisher Scrubbing):** The unlearning model (`m2_model`) is initialized as a deep copy of the `base_model`. Fisher Scrubbing steps are then iteratively applied using data from the `forget_loader`, for a predetermined number of `unlearn_epochs` or until a maximum of `max_scrub_steps` is reached. The `fisher_diag` is computed once on the `retain_loader` at the outset of the M2 process.

4 Implementation

Our project focused on developing a machine unlearning framework using Python and PyTorch for deep learning tasks. A key extension was the use of a K9dbMock to manage and track data ownership, enabling realistic testing of unlearning requests. The central algorithmic component of our work is an advanced Fisher Scrubbing algorithm, carefully designed to ensure both effectiveness and computational efficiency when removing specific data from trained models.

4.1 Codebase Metrics

The entire framework, encompassing the definition of our neural network models, various data utility functions, the comprehensive K9dbMock database simulation, and the intricate logic of the Fisher Scrubbing algorithm, was developed

within approximately 400-500 lines of Python code. This figure deliberately excludes external library code (e.g., PyTorch, torchvision, numpy, matplotlib) but includes all custom classes, helper functions, and the scripts orchestrating our experiments.

4.2 Implementation and Engineering Challenges

During the implementation phase, we navigated several significant technical and engineering hurdles:

1. **Ensuring Stable Forgetting Objectives:** Our initial attempts at directly maximizing the loss for forgotten data proved highly unstable, often leading to exploding gradients and compromised model stability. This was successfully addressed by pivoting to a KL divergence to uniform objective for the forget loss, which provided a far more stable and bounded target, enabling controlled unlearning without system collapse.
2. **Managing Fisher Information Matrix Scalability:** The full computation and storage of the Fisher Information Matrix (FIM) are notoriously resource-intensive for modern deep learning models. To overcome this, we implemented a diagonal approximation of the FIM, drastically reducing both computational overhead and memory footprint while retaining sufficient fidelity for effective regularization during unlearning.
3. **Balancing Forgetting Efficacy with Retained Utility:** A delicate balance had to be struck in tuning the α parameter, which modulates the influence of the Fisher information. Achieving effective forgetting without unduly degrading the model’s performance on the remaining, retained data required extensive iterative experimentation and fine-tuning.
4. **Realistic Data Management Simulation:** Crafting K9dbMock to accurately simulate real-world data ownership and cascading deletions within a deep learning pipeline presented unique engineering challenges. Ensuring data integrity and the correct dynamic subsetting of data for retain and forget sets, especially with a Zipfian distribution of user data, was paramount.
5. **GPU Resource Constraints:** A prevalent challenge, particularly within a shared Colab environment, was the limitation of available GPU resources. This often restricted our experiments to smaller subsets of large datasets (e.g., MNIST or reduced CIFAR-10) and more compact model architectures. These constraints necessitated careful experimental design to achieve meaningful results within practical execution times, even if full-scale industrial scenarios could not be replicated.

6. **Interactive Elements Integration:** Integrating the unlearning logic seamlessly with ipywidgets for an interactive user demonstration required meticulous state management and careful consideration of asynchronous execution flows within the Jupyter notebook environment.

5 Evaluation

To start with, the evaluation steps of our unlearning pipeline span three axes: utility, forgetting, and efficiency. Utility is measured by retention accuracy, computed on data that should remain in the training set after a deletion request. Forgetting is measured by forget accuracy, computed on the removed subset, where lower values are better and ideally match a model trained as if the deleted data never existed. Efficiency is measured by the end-to-end runtime of the deletion operation. All results and figures are generated by our main notebook (`Unlearning_Pipeline.ipynb`), which contains the full evaluation code, metric calculations, and experiment configurations.

5.1 Evaluation Results

To present the evaluation from a more statistical perspective, we assess our unlearning pipeline along three metrics: retain accuracy, forget accuracy, and runtime, comparing Full Retrain (M0) as the gold-standard baseline against Fisher Scrub (M2) as our unlearning method.

In the first scenario, where users are randomly assigned to images, Full Retrain achieves strong performance (retain accuracy 98.46%) but requires substantially longer runtime (35.51 s). Fisher Scrub is dramatically faster (1.43 s) but exhibits a moderate utility drop (retain accuracy 91.22%). This scenario reflects a harder mixed-ownership deletion setting, where the removed data does not correspond to a clean concept and approximate unlearning therefore trades accuracy for speed. We observe high forget accuracy for both models, which reflects a limitation of our experimental setting: the dataset contains only ten distinct classes. As a result, randomly “forgetting” a subset of data does not significantly reduce accuracy, since the model can still benefit from learning from data associated with other users.

The second scenario validates our hypothesis. When users are assigned to exclusive classes, the deletion target is much cleaner and forgetting becomes easier to verify. Full Retrain preserves retained-class accuracy (retain accuracy on digits 1–9: 98.91%) while driving the deleted-class accuracy close to zero (forget accuracy on digit 0: 0.00%), with a runtime of 54.65 s. Fisher Scrub achieves similar forgetting behavior (forget accuracy 0.03%) while maintaining high retained accuracy (95.6%), again completing in 1.43 s. Overall, these results demonstrate that Fisher Scrub provides an order-of-magnitude improvement in runtime, with the strongest effect

Table 1: Users are randomly assigned to images.

Metric	Full Retrain (M0)	Fisher Scrub (M2)
Retain Acc (%)	98.46	91.22
Forget Acc (%)	96.89	90.70
Runtime (s)	35.51	1.43

Table 2: Users are assigned to an exclusive class.

Metric	Full Retrain (M0)	Fisher Scrub (M2)
Retain Acc (%)	98.91	95.60
(Digits 1–9)		
Forget Acc (%)	0.00	0.03
(Digit 0)		
Runtime (s)	54.65	1.43

tiveness when the deletion target has clear structure, while mixed-ownership deletions expose the primary utility trade-off.

5.2 Zipfian evaluation design

To stress test unlearning under realistic heavy-tailed user ownership, we add a Zipfian workload evaluation that focuses on runtime behavior as the deletion workload and dataset scale change. The key motivation is that in practice, user data is not uniformly distributed: a small number of “heavy” users contribute a disproportionate share of samples, so deletion requests are cost-dominated by which users are removed, not just how many. We therefore evaluate M0 (full retrain) versus M2 (Fisher Scrub) under three Zipfian-style runtime experiments, all implemented in the notebook.

First, we test scalability with fixed delete size (Fig. 1): we hold the deletion size constant (100 points) and increase the total dataset size, then measure unlearning time. As expected, M0 retraining time grows strongly with dataset size, since it must re-optimize over the remaining data, while M2 remains almost flat because the scrub update cost depends far less on total data volume. This supports the claim that Fisher Scrub is better suited for scalable “forget a small set” operations when the overall training corpus is large.

Second, we test deletion amount versus time (Fig. 2): we hold the dataset size fixed and increase the percentage of data deleted, then measure runtime. Here, M0 runtime decreases as more data is deleted, because retraining becomes cheaper on a smaller remaining dataset, while M2 stays close to constant with only minor variation. This experiment highlights an important operational point: retraining is not always the worst case, because when deletions are massive, the remaining training set shrinks enough that restarting becomes relatively less expensive.

Third, we evaluate Zipfian cost by deleting top K heavy

Fig 1: Scalability (Fixed Delete=100)

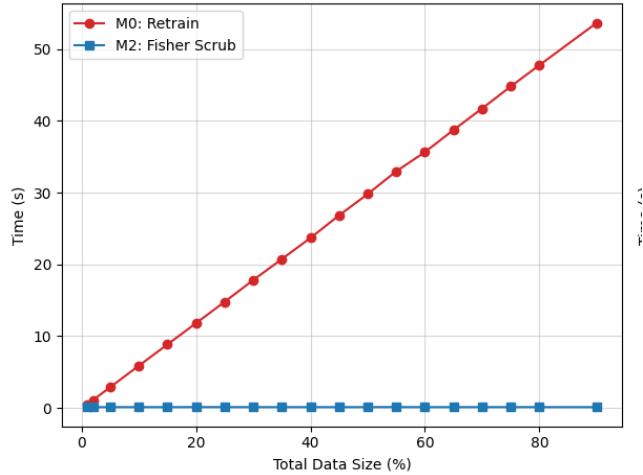


Fig 2: Deletion Amount vs Time

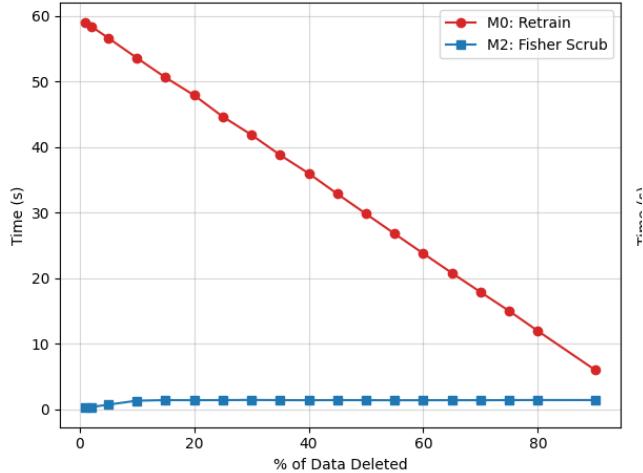
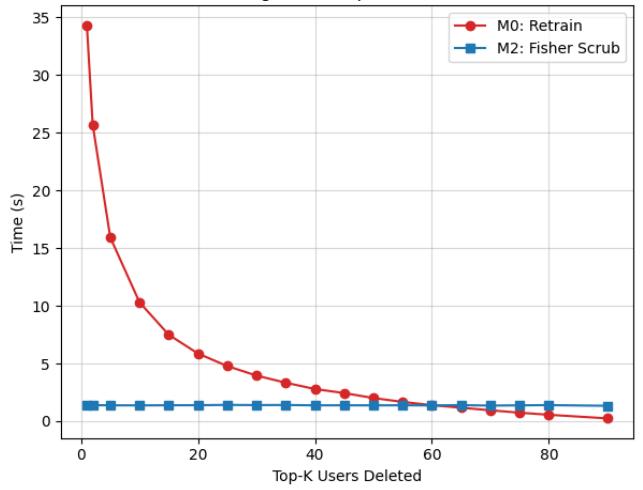


Fig 3: The Zipfian Cost



users (Fig. 3): we assume users follow a Zipfian contribution distribution and delete the top K users ranked by data vol-

ume, then measure runtime. For small to moderate K , which corresponds to realistic deletion patterns, M_2 is consistently faster and nearly constant, clearly outperforming M_0 . However, in the extreme regime where K approaches the full user set (for example, close to 99), the cost of M_0 retraining can become slightly lower than M_2 , because at that point the remaining dataset is so small that “start over” is the natural and cheaper strategy. Overall, this Zipfian evaluation shows that our unlearning method is most beneficial in the common case of deleting a few users, while also revealing a sensible boundary condition: when deletions approach “delete almost everything,” full retraining can be the more efficient choice.

5.3 Summary

Overall, our evaluation indicates that Full Retrain (M_0) remains the correctness reference but is expensive, while Fisher Scrub (M_2) delivers large runtime savings with measurable tradeoffs depending on the deletion structure. In clean deletion targets (exclusive class), Fisher Scrub achieves near ideal forgetting with strong retained accuracy and an order of magnitude faster runtime. In mixed ownership deletions, it still provides major efficiency gains but can reduce overall model utility. Because real-world user ownership is typically heavy-tailed, our Zipfian-based evaluation suggests practical outcomes will often fall between these two extremes, and Fisher Scrub offers a scalable unlearning option with predictable performance behavior under realistic user distributions.

References

- [1] DAK ALBAB, K., SHARMA, I., ADAM, J., KILIMNIK, B., JEYARAJ, A., PAUL, R., AGVANIAN, A., SPIEGELBERG, L., AND SCHWARZKOPF, M. K9db: Privacy-compliant storage for web applications by construction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)* (Boston, MA, July 2023), USENIX Association, pp. 99–116.
- [2] GOLATKAR, A., ACHILLE, A., AND SOATTO, S. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 9304–9312.

Notes

All group members contributed equally