Supplementary Materials for CLUE: Bringing Machine Unlearning to Mobile Devices

1. Related Work on Machine Unlearning

The first work in machine unlearning was published in 2015 [5], where Cao et al. proposed to decompose a machine learning model into a series of sums and eliminate some sum operations that are affected by the forget class. Later work focused on the topic of *exact unlearning*, which is the process of completely removing the knowledge so that the performance is exactly the same as retraining the model without the data to be unlearned [3, 14, 17, 24]. However, these works cannot be applied to deep neural networks (DNNs) due to the non-convex nature of the objective function.

The first exact unlearning framework for DNNs was SISA [2], which divides the dataset into multiple slices to create an ensemble of DNNs and uses majority voting for inference. This requires only the DNNs trained on the slices containing the samples of the forget dataset to be retrained. However, Koch et al. [20] highlighted SISA's limitations with class imbalance. Building on SISA, Gupta et al. [18] provided a framework with provable differential privacy guarantees when unlearning requests arrive in streams. Although other works have attempted certifiable unlearning [8, 23, 29], they often rely on strong assumptions about the learning algorithm and lack evaluation on standard benchmark datasets. This line of research has helped establish a foundation for certifiable unlearning with provable guarantees, particularly in privacy-preserving applications. Yet, for broader scenarios—such as defending against backdoor attacks, enhancing lifelong learning, or improving fairness in DNNs—more practical approaches have emerged. For instance, Golatkar et al. [15, 16] employed the Fisher information matrix (FIM) to identify critical weights for unlearning. Recently, Chundawat et al. [9] proposed an approach where knowledge of the data to be forgotten is distilled from a randomly initialized teacher, while Fan et al. [11] addressed unlearning for both classification and image generation tasks. To accelerate the process, Tarun et al. [28] introduced an incremental approach that adjusts parameters based on the removal of specific data points without a full update. The data points are removed by fine-tuning the DNN on noise samples generated by maximizing the loss for the forget dataset. *All the above approaches assume access to the retain dataset.*

The line of work in [7, 10, 12, 13, 27] has investigated machine unlearning without accessing the retain dataset. While Sekhari et al. [27] does not require storing the entire retain dataset but only its FIM, its computational complexity scales cubically with the number of parameters in the DNN since it needs to obtain the FIM. Foster et al. [13] tried to alleviate this issue by approximating only the diagonal of the FIM. While Foster et al. [12] tried to align the output of the model to that of the out-of-distribution (OOD) input, their approach is based on minimizing the Lipschitz constant of the DNN, which is orthogonal to the approach taken in CLUE. Chen et al. [7] proposed *Boundary Shrinking (BS)* and *Boundary Expanding (BE)*. BS relabels the data points of the forget dataset with the nearest neighbor class label and fine-tunes the DNN with the resulting forget dataset. BE adds an extra class and assigns the data points of the forget class to that. Then after fine-tuning, the extra node is removed. However, [7] uses small datasets (i.e., CIFAR10 and randomly sampled 10 classes from VGGFace2) and it is evaluated on outdated architectures (e.g. VGG, AllCNN).

Another line of work explores subspace-based unlearning. [25] introduces a training-free approach that directly identifies and removes low-dimensional subspaces associated with the forget set, efficiently eliminating knowledge without retraining. However, its reliance on clean subspace separation limits effectiveness when forget and retain knowledge are entangled. [6] leverages null-space constraints calibrated to retain data, suppressing forget-set knowledge while reducing over-unlearning, though it requires reliable pseudo-labeling and access to retain data. [31] employs a sparse autoencoder to decompose hidden representations into relevant and irrelevant subspaces, projecting forget-set gradients into the latter to improve the forget–retain trade-off, at the cost of additional overhead.

Beyond these approaches, recent work has studied *zero-shot unlearning*, which removes knowledge without access to explicit forget data. Methods include iterative null-space projection for concept erasure [26], direct parameter editing to overwrite or nullify specific facts [21, 22], and noise generation for selectively damaging information about forget class [10, 28]. These strategies highlight the growing interest in efficient, data-free unlearning, though they often trade

off between performance of forgetting and preservation of retained knowledge.

2. Details about the Baselines

We sweep through the learning rate $\mu \in [0.001, 0.00001]$ and number of iterations $E \in \{2, 5, 8, 10, 12, 14, 16, 18, 20\}$ for all the baselines. Additionally, we use Stochastic Gradient Descent Optimizer (SGD) with momentum value of 0.9 and 0 weight-decay for optimization of the DNN.

Retrain The DNN is retrained from scratch by using the retain dataset. Since this represents the upper bound of performance (i.e., exact unlearning), we use this as the *golden standard* and report the gap between *Retrain* and the other approaches in terms of the considered performance metrics;

Gradient Ascent (GA) This approach focuses on maximizing the training loss for the forget dataset. In classification tasks, this means maximizing the cross-entropy loss for the forget dataset. For a batch of samples $\mathbf{X}$, the DNN parameters are updated using the Eq. (1):

$$\theta^{\mathbf{i}} = \theta^{\mathbf{i-1}} + \mu \nabla \mathcal{L}_{cross\ entropy} \tag{1}$$

Influence Update (IU) This approach uses the influence function formulation from [19]. It measures the parameter change ($\Delta\theta$) in the DNN when the forget dataset is excluded from training. The change is estimated as $\mathbf{H}^{-1}\nabla L(\mathcal{D}_f; \theta_0)$, where:

- $\mathbf{H}^{-1}$ is the inverse Hessian matrix
- $\nabla^2_{\theta\theta} L(\mathcal{D}; \theta_0)$ evaluated at θ_0
- $\nabla L(\mathcal{D}_f; \theta_0)$ is the gradient of the loss function for the forget dataset.

The updated parameters (θ_{MU}) are given by $\theta_{MU} = \theta_0 + \Delta\theta$

Random Relabeling (RL) In this approach, samples of the forget dataset are randomly relabeled. The DNN is fine-tuned using this relabeled forget dataset. In our data-constrained settings, only the forget dataset with random labels is used for fine-tuning.

Boundary Expanding (BE) This method introduces an additional neuron in the final layer of the DNN to represent a "dummy" class. Samples of the forget class are assigned to this dummy class, and the resulting forget dataset is used to fine-tune the DNN. After fine-tuning, the dummy class is removed.

Boundary Shrinking (BS) In this approach, samples of the forget class are adversarially perturbed using algorithms like Fast Gradient Sign Method (FGSM) or Projected Gradient Descent (PGD). The samples are relabeled based on the predicted class of their perturbed counterparts. The relabeled dataset is then used to fine-tune the DNN.

Lipschitz Unlearning (LU) This method aims to align the representations of the forget dataset with their heavily corrupted counterparts. Corruption is introduced through strong Gaussian noise, and the approach minimizes the Lipschitz constant of the DNN. The Lipschitz constant is estimated as Eq. (2)

$$k = \frac{||f(\mathbf{x}) - f(\mathbf{x_{noisy}})||_2}{||\mathbf{x} - \mathbf{x_{noisy}}||} \tag{2}$$

Here, k is the Lipschitz constant, $\mathbf{x}$ is the clean input, and $\mathbf{x_{noisy}}$ is the noisy counterpart.

Unlearning with Single Pass Impair and Repair (UNSIR) This approach assumes no access to the forget dataset but uses the retain dataset. A proxy forget dataset is created from optimized noise to unlearn the forget class. The process involves two steps:

1. **Impair Step**: Noise is optimized to maximize training loss for the forget dataset. The noise is then used to fine-tune the DNN.
2. **Repair Step** (omitted in our setup): Fine-tuning the DNN on the retain dataset to preserve generalization. For our setting, only the impair step is used.

3. Discussion of Results

In this section, we will discuss the observed performance of the baseline approaches in further detail.

Gradient Ascent (GA) The *GA* approach performs best for CIFAR100 with a ViT base architecture. By design, it effectively removes knowledge of the forget class by increasing its loss. However, indiscriminate weight updates can harm the retain set's accuracy, as observed for CIFAR10, CIFAR100, and ResNet-20.

Random Labels (RL) is conceptually very close to the BE and BS approaches. In RL, the samples of the forget class are relabeled randomly. Fine-tuning with this relabeled forget class likely breaks the previously learned association of the samples of the forget class with the class label. This causes the DNN to perform poorly on the forget class. At the same time, this starts associating samples of the forget class to class labels of the retain class. Depending on the relabelling (which is randomized) process, the decision space might need to change radically - a sample from the forget class may get assigned to a distant class. This can reduce the accuracy of the DNN on the retain set. This is observed in the results - a good Unlearning Accuracy (UA) performance is accompanied by a drop in Remaining Accuracy (RA) and Testing Accuracy (TA) for RL.

Influence Unlearning (IU) The *IU* approach performs poorly, especially for ViT architectures. While it shows comparable performance for ResNet on CIFAR100, its inconsistency across datasets and architectures indicates weak generalization.

BE and BS The BE and BS approaches are the closest ones to CLUE. These two approaches perform consistently across model architectures and datasets. This indicates that a crucial component of class-level machine unlearning (CMU) is to remove the forget samples from the decision space. The challenge these approaches face is to retain the generalization capability of the DNN. We hypothesize that the direct assignment of the samples of the forget set to the nearest class confuses the DNN. This can happen because semantically similar samples from the forget set are assigned to different retain set classes. We get support for this hypothesis from the observation that even with ViT and CIFAR10 datasets, these approaches lose accuracy on the retain training set and test set by up to 10.53% and 11.58% respectively. Another support for this hypothesis comes from the observation that the RL-a more extreme version of BS and BE approaches performs even worse and loses 15.4% accuracy on the test set of the CIFAR10 retain dataset. In this work, we search for the optimal learning rate for unlearning between 10^{-3} and 10^{-5}. We choose 10^{-3} as the last learning rate for our DNN during training is 10^{-3}. We stop our search at 10^{-5} following the original work [7].

Lipschitz Unlearning (LU) The *LU* method generally underperforms. It aligns DNN outputs with corrupted inputs by minimizing the Lipschitz constant, which negatively impacts retain set accuracy. Two possible reasons are:

1. Direct estimation of the Lipschitz constant may fail to protect retain set information.

2. Batch normalization statistics align with corrupted inputs, disrupting retain set performance.

Unlearning by Selective Impair and Repair (UNSIR) This approach performs poorly across metrics for ViT but shows potential with ResNet on CIFAR100. As in our setting, we do not use retain set samples for the repair step, training with noise reduces generalization significantly.

4. Implementation and Hardware Details

The training of the models mentioned in section 4 of the manuscript uses the following configurations:

Hardware Details We have performed our experiments on a Dell Precision Tower 3650. The machine has 16 CPU cores with 32GB RAM. The machine also has NVIDIA RTX A4000 GPU with 16 GB memory.

The experiment on VGGFace2 dataset is performed on a machine with 48 Cores and 512GB RAM. We use NVIDIA A100 GPU with 80GB memory.

Model Training

- We train the resnet model from scratch on CIFAR benchmarks. We use a batch size of 256, a learning rate of 0.1 with multi-step learning rate reductions to one-tenth of the current value at step 91 and 136, and train for 182 epochs. We use the Stochastic Gradient Descent (SGD) optimizer with momentum 0.9 to minimize the cross-entropy loss.

- For the transformer models, we use pretrained models from the timm [32] repository of the hugging-face library. We adapt them to run on the CIFAR benchmarks. We change the patch size to 4 and the image size to 32. We fine-tune them for 50 epochs with SGD optimizer with momentum 0.9 minimizing the cross-entropy loss. We use a learning rate of 0.01.

- We use pretrained models from the timm [32] repository of the hugging-face library. We adapt them to run on the VGGFace2 dataset. We use an image size of 224×224. We fine-tune them for 20 epochs with SGD optimizer with momentum 0.9 minimizing the cross-entropy loss. We use a learning rate of 0.01.

Retraining from Scratch While retraining from scratch, we use the same configuration as training from scratch (for Resnet architectures) or fine-tuning (for transformer architecture).

Evaluation Protocol CMU algorithms need to work for different classes with the same hyperparameter setting. In order to realize this, **we enforce the following protocol for evaluation - (step 1) perform hyper-parameter tuning for one class and (step 2) evaluate for the rest of the classes with the same hyper-parameter setting.**

We randomly choose the class#2 as for hyper-parameter tuning. We follow this protocol for each baseline as well as CLUE and report their best performances from step 1 in the manuscript. We use the obtained hyper-parameters and evaluate the CMU approaches on additional classes. The evaluation results are reported here.

Hyperparameters The values of the hyperparameters used in this work are as follows-

- For the gaussian noise added to the input of the teacher DNN, we use $\mu = 0$. We randomly vary the strength of the noise from $\sigma^2 = 0.5$ to $\sigma^2 = 1$. This is because a small value of sigma may not resemble OOD while a large value can possibly resemble random soft labels. In order to balance these two extremes, we randomly vary the noise strength so that the information about the training data is not entirely removed while resembling OOD data as closely as possible.

- For the value of the hyperparameter λ in balancing the knowledge distillation (KD) and energy loss in equation 5 of the manuscript, we set it to fixed value of 0.1.

- For the threshold τ for generating gradient masks, we set it to 99th quantile of $R(w)$ in equation 6 of the

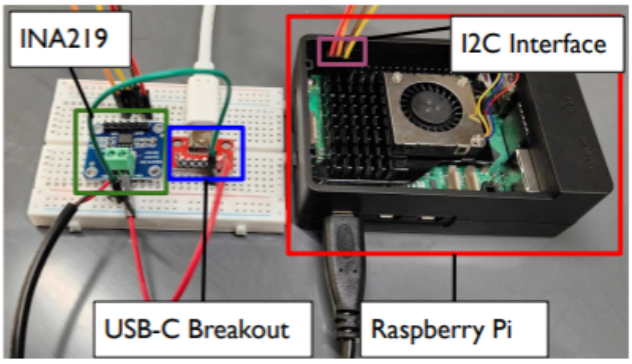


Figure 1. The setup for power and latency measurement.

manuscript. Essentially, CLUE updates only the few most important weights for the forget set.

Number of Iterations for Unlearning We highlight that a major reason for the run-time-efficiency of CLUE is the low number of iterations required by CLUE. For example, for ViT base architecture CLUE requires only 8 iterations for both CIFAR10 and CIFAR100 datasets. On the same architecture and datasets, boundary unlearning requires 16 iterations and RL requires 14 iterations. Similarly, for the Resnet20 architecture, CLUE requires 5 iterations while boundary unlearning and RL requires 8 and 12 iterations respectively.

Details about Visualization of Decision Space In order to visualize the decision space, we use the output of the penultimate layer. In order to reduce the resulting high-dimensional (number of dimension is 64) feature map obtained from the penultimate layer, we utilize the t-SNE [30] algorithm. We use the scikit-learn [4] for the implementation of t-SNE where we set the perplexity value to 3 and represent the data with two components leaving the rest of the hyper-parameters unchanged. The two axes in the figure 2 in the manuscript refer to the two components returned by the algorithm.

Details about Power Measurement on Edge Devices We considered Raspberry-Pi-5 as edge device for running mobile computer vision (CV) applications. Our experimental setup is shown in Figure 1. The Raspberry Pi runs a quad-core ARM A76 SoC running at upto 2.4 GHz with 8 GB LPDDR4 memory. We use an INA-219 sensor to sample the current and use it to calculate the power averaged over all the test samples. For latency, we report the time including any pre- or post-processing involved. We intentionally include cold-start effects in our measurements to reflect real-world edge deployment scenarios where devices may frequently restart or handle sporadic requests. All batches, including initial ones, are included in latency reporting. The latency measurements were averaged over 5 runs.

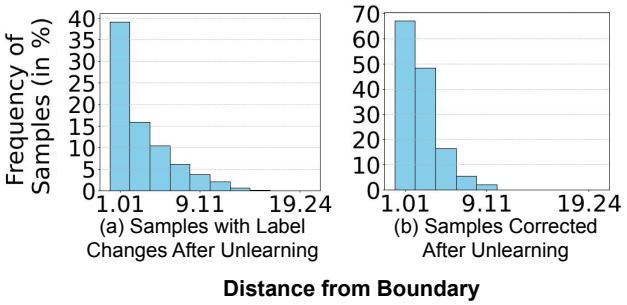


Figure 2. Subfigure (a) shows the percentage of samples that change their labels due to unlearning against their distance from the decision boundary. Subfigure (b) shows the percentage of samples that were misclassified before but are corrected now after unlearning. All the samples belong to the retain set.

5. Understanding Effect on Hard Inputs from Retain Set

To understand how CLUE affects hard examples—samples closer to the decision boundary—in the retain set, we conduct an experiment to measure the percentage of samples at varying distances from the boundary that change labels due to unlearning. We perform this experiment using CIFAR-10 and a ViT-base DNN.

To provide a comprehensive view, we plot two metrics against the distance from the nearest decision boundary-

- The percentage of samples that change labels after unlearning.
- The percentage of samples that are corrected after unlearning.

Our results reveal that samples closest to the decision boundary are the most affected. As shown in Subfigure 2(a), these hard samples change labels far more frequently than easier ones. Additionally, unlearning leads to a rearrangement of the decision space, allowing samples near the boundary to be corrected more often.

6. Comparison with SalUn

We compare CLUE with SalUn [11] in Table 1. For fair comparison, we adapt SalUn so that it does not use the data from the retain set. We use the gradient mask generation setup and the random labeling approach from SalUn and while unlearning, we use only the forget set for model update and leave out the retain set. Specifically, we modify the objective function to $\min_{\Delta\boldsymbol{\theta}} \; L^{(1)}_{\text{SaUn}}(\boldsymbol{\theta}_u) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_f,\, y'\neq y}\left[\ell_{\text{CE}}(\boldsymbol{\theta}_u;\mathbf{x},y')\right]$. Here, $\mathcal{D}_f$ is the forget set and y' is the random label other than the label of the forget class. While sweeping for optimal hyperparameter, we follow the same range as reported in [11]. We observe

that CLUE outperforms SalUn by up to 12.57%. CLUE performs much better than SalUn especially for CIFAR10 while the margin shrinks with increase in class number for CIFAR100.

Table 1. Comparison of CLUE with SalUn: To ensure a fair comparison, we adapt SalUn to refrain from using data from the retain set.

Dataset/ Architecture	**Unlearning Methods**	**UA**	**RA**	**TA**	**MIA**	**Average Gap**
CIFAR10/ViT Base	Retrain	0	99.8	96.92	100	
	CLUE	**2.04 (2.04)**	**98.42 (1.38)**	95.86 (1.06)	**97.95 (2.05)**	**1.63**
	SalUn	11.55 (11.55)	97.80 (2)	**96.72 (0.2)**	56.95 (43.05)	14.2
CIFAR100/ViT Base	Retrain	0	99.24	85.72	100	
	CLUE	0.22 (0.22)	**91.24 (8)**	**80.77 (4.95)**	**99.77 (26.33)**	**3.37**
	SalUn	**0 (0)**	81.53 (17.71)	71.92 (13.8)	100 (0)	4.42
CIFAR10/Resnet20	Retrain	0	99.75	90.18	100	
	CLUE	**10.48 (10.48)**	**96.61 (3.14)**	**89.7 (0.48)**	**89.50 (10.50)**	**6.15**
	SalUn	18.33 (18.33)	95.51 (4.24)	89.07 (1.11)	81.66 (18.34)	10.505
CIFAR100/Resnet20	Retrain	0	87.84	62.85	100	
	CLUE	**3.77 (3.77)**	**68.11 (19.73)**	**54.75 (8.10)**	**96.22 (3.78)**	**8.84**
	SalUn	5.33 (5.33)	67.52 (20.32)	54.31 (8.51)	94.88 (5.12)	9.82

7. Unlearning Results on Additional Classes

We provide unlearning results on additional classes in Table 2 through Table 9.

Table 2. Additional result for ViT base architecture trained on CIFAR10 and forget class#0.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	99.73	97.15	100	0
GA	**0 (0.00)**	11.11 (88.62)	11.11 (86.04)	**100 (0.00)**	43.67
RL	21.51 (21.51)	81.22 (18.51)	80.56 (16.59)	59.63 (40.37)	24.25
IU	98.04 (98.04)	**99.31 (0.42)**	**96.91 (0.24)**	1.95 (98.05)	49.19
BE	29.5 (29.50)	93.58 (6.15)	90.73 (6.42)	69.24 (30.76)	18.21
BS	31.24 (31.24)	93.87 (5.86)	91.18 (5.97)	68.75 (31.25)	18.58
LU	**0 (0.00)**	11.11 (88.62)	11.11 (86.04)	**100 (0.00)**	43.67
UNSIR	99.15 (99.15)	99.02 (0.71)	96.42 (0.73)	0.84 (99.16)	49.94
CLUE	**0 (0.00)**	98.6 (1.13)	96.07 (1.08)	**100 (0.00)**	**0.55**

Table 3. Additional result for ViT base architecture trained on CIFAR10 and forget class#6.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	99.73	96.72	100	
GA	**0 (0.00)**	11.11 (88.62)	11.11 (85.61)	**100 (0.00)**	43.56
RL	23.52 (23.52)	86.31 (13.42)	85.22 (11.50)	63.52 (36.48)	21.23
IU	99.2 (99.20)	**99.29 (0.44)**	94.74 (1.98)	0.8 (99.20)	50.21
BE	43.29 (43.29)	96.58 (3.15)	**95.83 (0.89)**	55.24 (44.76)	23.02
BS	42.93 (42.93)	97.59 (2.14)	94.75 (1.97)	57.06 (42.94)	22.50
LU	**0 (0.00)**	11.11 (88.62)	11.11 (85.61)	**100 (0.00)**	43.56
UNSIR	98.02 (98.02)	92.28 (7.45)	88.95 (7.77)	1.97 (98.03)	52.82
CLUE	**0 (0.00)**	98 (1.73)	94.77 (1.95)	**100 (0.00)**	**0.92**

Table 4. Additional result for ViT base architecture trained on CIFAR100 and forget class#0.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	**0**	99.51	86.57	100	0
GA	91.77 (91.77)	92.58 (6.93)	81.98 (4.59)	8.22 (91.78)	48.77
RL	**0 (0.00)**	55.12 (44.39)	49.62 (36.95)	**100 (0.00)**	20.34
IU	94.44 (94.44)	**92.65 (6.86)**	**82 (4.57)**	5.5 (94.50)	50.09
BE	**0 (0.00)**	78.3 (21.21)	68.54 (18.03)	**100 (0.00)**	9.81
BS	**0 (0.00)**	68.68 (30.83)	61.58 (24.99)	**100 (0.00)**	13.96
LU	**0 (0.00)**	1.01 (98.50)	1.01 (85.56)	0 (100.00)	71.02
UNSIR	46.14 (46.14)	81.33 (18.18)	40.43 (46.14)	18.66 (81.34)	47.95
CLUE	**0 (0.00)**	89.67 (9.84)	79.2 (7.37)	**100 (0.00)**	**4.30**

Table 5. Additional result for ViT base architecture trained on CIFAR100 and forget class#12.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	99.26	85.21	100	0
GA	74.66 (74.66)	92.68 (6.58)	82.13 (3.08)	25.33 (74.67)	39.75
RL	**0 (0.00)**	71.35 (27.91)	66.23 (18.98)	**100 (0.00)**	11.72
IU	84.66 (84.66)	**92.72 (6.54)**	**82.18 (3.03)**	15.33 (84.67)	44.73
BE	**0 (0.00)**	53.79 (45.47)	47.51 (37.70)	**100 (0.00)**	20.79
BS	**0 (0.00)**	86.16 (13.10)	75.92 (9.29)	**100 (0.00)**	5.60
LU	**0 (0.00)**	1.01 (98.25)	1.01 (84.20)	0 (100.00)	70.61
UNSIR	26.22 (26.22)	41.99 (57.27)	36.12 (49.09)	73.77 (26.23)	39.70
CLUE	**0 (0.00)**	89.19 (10.07)	79.24 (5.97)	**100 (0.00)**	**4.01**

Table 6. Additional result for Resnet 20 architecture trained on CIFAR10 and forget class#0.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	99.69	89.53	100	0
GA	11.92 (11.92)	93.54 (6.15)	84.61 (4.92)	88.55 (11.45)	8.61
RL	8.56 (8.56)	81.23 (18.46)	78.55 (10.98)	88.3 (11.70)	12.43
IU	17.77 (17.77)	**94.05 (5.64)**	**87.87 (1.66)**	82.22 (17.78)	10.71
BE	14.8 (14.80)	92.15 (7.54)	83.8 (5.73)	85.2 (14.80)	10.72
BS	24.6 (24.60)	90.74 (8.95)	83.37 (6.16)	75.4 (24.60)	16.08
LU	**0 (0.00)**	11.11 (88.58)	11.11 (78.42)	0 (100.00)	66.75
UNSIR	0.02 (0.02)	13.22 (86.47)	12.9 (76.63)	**99.97 (0.03)**	40.79
CLUE	11.64 (11.64)	92.65 (7.04)	87.02 (2.51)	88.35 (11.65)	**8.21**

Table 7. Additional result for Resnet20 architecture trained on CIFAR10 and forget class#6.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	99.78	89.53	100	0
GA	12.53 (12.53)	91.52 (8.26)	80.66 (8.87)	85.22 (14.78)	11.11
RL	20.11 (20.11)	83.14 (16.64)	81.23 (8.30)	82.11 (17.89)	15.74
IU	27.56 (27.56)	90.05 (9.73)	82.34 (7.19)	83.72 (16.28)	15.19
BE	6.48 (6.48)	80.44 (19.34)	73.56 (15.97)	**93.51 (6.49)**	12.07
BS	15.2 (15.20)	86.5 (13.28)	79.12 (10.41)	87.79 (12.21)	12.78
LU	**0 (0.00)**	11.11 (88.67)	11.11 (78.42)	0 (100.00)	66.77
UNSIR	6.86 (6.86)	11.78 (88.00)	11.48 (78.05)	93.11 (6.89)	44.95
CLUE	10.77 (10.77)	**93.19 (6.59)**	**86.3 (3.23)**	89.4 (10.60)	**7.80**

8. Effect of Varying Forget Set Size

To examine the sensitivity of unlearning performance to the size of the forget set, we conduct experiments using different fractions of the full forget set. Specifically, we vary the proportion of samples to be forgotten across $\{10\%, 25\%, 50\%, 75\%, 95\%, 100\%\}$. This setup allows us to characterize how effectively the unlearning method scales as more or fewer samples are designated for removal.

Table 10 reports results on CIFAR-10 and CIFAR-100

Table 8. Additional result for Resnet20 architecture trained on CIFAR100 and forget class#0.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	87.71	61.94	100	0
GA	**0 (0.00)**	52.74 (34.97)	43.2 (18.74)	100 (0.00)	13.43
RL	14.44 (14.44)	61.8 (25.91)	50.96 (10.98)	85.55 (14.45)	16.45
IU	**0 (0.00)**	55.33 (32.38)	41.28 (20.66)	88.25 (11.75)	16.20
BE	2.22 (2.22)	63.65 (24.06)	51.56 (10.38)	97.77 (2.23)	9.72
BS	1.77 (1.77)	**64.86 (22.85)**	**52.37 (9.57)**	**98.22 (1.78)**	**8.99**
LU	4.66 (4.66)	0.59 (87.12)	0.63 (61.31)	95.33 (4.67)	39.44
UNSIR	**0 (0.00)**	2.25 (85.46)	2.22 (59.72)	0 (100.00)	61.30
CLUE	3.77 (3.77)	55.49 (32.22)	46.63 (15.31)	96.22 (3.78)	13.77

Table 9. Additional result for Resnet20 architecture trained on CIFAR100 and forget class#12.

Unlearning Methods	UA	RA	TA	MIA	Gap
Retrain	0	87.52	62.45	100	0
GA	13.55 (13.55)	61.9 (25.62)	48.22 (14.23)	83.61 (16.39)	17.45
RL	0.55 (0.55)	45.16 (42.36)	43.67 (18.78)	81.16 (18.84)	20.13
IU	**0 (0.00)**	55.33 (32.19)	41.28 (21.17)	88.25 (11.75)	16.28
BE	1.55 (1.55)	58.99 (28.53)	48.63 (13.82)	**98.44 (1.56)**	11.37
BS	1.55 (1.55)	**62.81 (24.71)**	50.93 (11.52)	**98.44 (1.56)**	**9.84**
LU	**0 (0.00)**	0.82 (86.70)	0.85 (61.60)	0 (100.00)	62.08
UNSIR	**0 (0.00)**	1.56 (85.96)	1.66 (60.79)	0 (100.00)	61.69
CLUE	7.55 (7.55)	62.51 (25.01)	**50.73 (11.72)**	92.44 (7.56)	12.96

with ViT-Base. We observe that CLUE maintains stable performance across a wide range of forget set sizes. In particular, the *unlearned accuracy (UA)* remains consistently close to zero, indicating that forgotten classes are successfully suppressed. Meanwhile, the *retained accuracy (RA)* and *test accuracy (TA)* experience only modest degradation, confirming that knowledge of the retained data is largely preserved. The *membership inference attack (MIA)* success rate also remains low, suggesting that unlearning reduces privacy leakage even under adversarial evaluation.

Importantly, the *average gap* remains consistently small across all settings. On CIFAR-10, the variance of the gap is only 0.32 (std. 0.56), while on the more challenging CIFAR-100 it is 0.93 (std. 0.96). These results demonstrate that CLUE is substantially more **stable across different forget set sizes** than prior baselines, maintaining near-retrain performance even as the fraction of forgotten data varies. This highlights the robustness of our approach and shows that unlearning can be applied reliably without requiring access to the entire forget set at once.

9. Comparison with Additional SOTA

Table 11 shows that CLUE consistently outperforms prior methods - zero-shot unlearning (ZSU) [10] and source-free unlearning (SFU) [1] - in both utility preservation and privacy. On CIFAR10, CLUE achieves near-retrain performance, with only minor drops in RA (–1.4%) and TA (–1.1%), while maintaining high MIA robustness and a very low gap (1.63). Competing approaches (ZSU, SFU) either catastrophically degrade retained knowledge (e.g., ZSU RA falls below 25%) or incur large gaps ($> 10\%$).

On the more challenging CIFAR100, CLUE again yields the strongest trade-off: UA is nearly zero, RA and TA remain close to retrain, and the gap is only 3.37—substantially lower than ZSU (5.61%) and SFU (6.95%). Importantly, MIA performance stays at near-ideal levels (≈ 100).

Overall, these results demonstrate that CLUE provides balanced, stable unlearning across datasets, while ZSU and SFU either unlearn at the cost of the retain set performance or reduce privacy guarantee.

10. Sequential Unlearning

The results in Table 12 highlight the effectiveness of CLUE in sequential unlearning compared to Boundary Shrink (BS), a state-of-the-art baseline. On both CIFAR10 and CIFAR100 with ViT-Base, CLUE consistently achieves near-zero Unlearned Accuracy (UA), closely matching the retraining oracle. Importantly, this suppression of forgotten classes is achieved without sacrificing Retained Accuracy (RA) or Test Accuracy (TA): performance degradation relative to retrain is marginal ($\leq 2.5\%$).

By contrast, BS incurs substantial drops in RA and TA (10–12% on average), reflecting significant leakage of unlearning into the retained set. This instability is further captured by the Average Gap metric, where CLUE maintains values under 1, while BS shows large gaps exceeding 8. Moreover, CLUE preserves the Membership Inference Attack (MIA) robustness of retraining ($\sim 100\%$), whereas BS substantially weakens privacy guarantees.

Overall, the results demonstrate that CLUE not only scales well under sequential unlearning (multiple classes forgotten in order) but also delivers robust and stable performance across datasets, outperforming BS in both utility preservation and privacy protection.

11. Effect of Noise Variance

Table 13 shows that the performance of CLUE depends on the strength of the injected noise. With very low variance ($\sigma^2 = 0.1$), forgetting is incomplete (UA remains high at 38.37), despite strong RA. Moderate variance ($\sigma^2 = 0.5$) reduces UA but still leaves a noticeable gap. At higher variance ($\sigma^2 = 1.0$), forgetting is achieved (UA ≈ 0) but at the cost of significant RA/TA degradation. Assigning random variance in the range $[0.5, 1.0]$ for each image achieves the best trade-off, with near-zero UA, strong RA/TA, high MIA, and the smallest gap (3.37). This is because the damaging effect of some samples exposed to high noise variance is compensated by others receiving lower noise strength, leading to a more balanced outcome.

12. Experiment on Midsized Dataset

To evaluate the performance of CLUE on a mid-sized dataset, we use Caltech256 with the ViT-Base-16 architec-

Dataset	Architecture	Method	Forget %	UA	RA	TA	MIA	Avg Gap
CIFAR-10	ViT-Base	Retrain	–	0	99.8	96.92	100	–
		CLUE	100%	0 (0)	95.36 (4.44)	92.77 (4.15)	100 (0)	2.15
			95%	0 (0)	94.85 (4.95)	92.11 (4.81)	100 (0)	2.44
			75%	2.04 (2.04)	98.42 (1.38)	95.86 (1.06)	97.95 (2.05)	1.63
			50%	1.52 (1.52)	96.92 (2.88)	95.86 (1.06)	98.24 (1.76)	1.81
			25%	0.53 (0.53)	94.51 (5.29)	92.04 (4.88)	99 (1)	2.92
			10%	2.89 (2.89)	96.96 (2.84)	94.24 (2.68)	95.6 (4.4)	3.20
CIFAR-100	ViT-Base	Retrain	–	0	99.24	85.72	100	–
		CLUE	100%	0.22 (0.22)	91.24 (8.0)	80.77 (4.95)	99.77 (26.33)	3.37
			95%	0.23 (0.23)	91.07 (8.17)	80.71 (5.01)	99.77 (0.23)	3.41
			75%	0 (0)	90.77 (8.47)	85.49 (0.23)	100 (0)	2.18
			50%	0 (0)	90.82 (8.42)	80.49 (5.23)	100 (0)	3.42
			25%	0 (0)	87.13 (12.11)	76.43 (9.29)	100 (0)	5.35
			10%	0 (0)	89.04 (10.2)	78.97 (6.75)	100 (0)	4.24

Table 10. Unlearning results on CIFAR-10 and CIFAR-100 with ViT-Base by varying the forgert set size. UA = Unlearned Accuracy, RA = Retained Accuracy, TA = Test Accuracy, MIA = Membership Inference Attack success, Avg Gap = average accuracy gap.

Table 11. Comparison of unlearning methods on CIFAR10 and CIFAR100 with ViT-Base. Numbers in parentheses denote deviation from Retrain.

Dataset/ Architecture	Unlearning Method	UA	RA	TA	MIA	Avg. Gap
CIFAR-10 ViT-Base	Retrain	0	99.80	96.92	100	0
	CLUE	2.04 (2.04)	98.42 (1.38)	95.86 (1.06)	97.95 (2.05)	1.63
	ZSU	1.32 (1.32)	24.54 (75.26)	21.78 (75.14)	10.20 (89.80)	60.38
	SFU	1.55 (1.55)	85.30 (14.50)	80.45 (16.47)	92.33 (7.67)	10.05
CIFAR-100 ViT-Base	Retrain	0	99.24	85.72	100	0
	CLUE	0.22 (0.22)	91.24 (8.00)	80.77 (4.95)	99.77 (26.33)	3.37
	ZSU	3.77 (3.77)	89.46 (9.78)	78.60 (5.12)	96.22 (3.78)	5.61
	SFU	2.33 (2.33)	87.24 (12.00)	76.38 (9.34)	95.88 (4.12)	6.95

Table 12. Comparison of unlearning methods on CIFAR10 and CIFAR100 with ViT-Base architecture. Numbers in parentheses indicate deviation from Retrain.

Dataset/ Architecture	Unlearned Classes	Unlearning Method	UA	RA	TA	MIA	Avg. Gap
CIFAR-10 ViT-Base	2, 8	Retrain	0	99.8	96.92	100	0
		CLUE	0.24 (0.24)	97.42 (2.38)	95.36 (1.56)	99.95 (0.05)	1.06
		BS	3.42 (3.42)	89.66 (10.14)	84.23 (12.69)	92.55 (7.45)	8.43
CIFAR-100 ViT-Base	2, 6	Retrain	0	91.84	80.85	100	0
		CLUE	0 (0)	90.67 (1.07)	79.6 (1.25)	100 (0)	0.58
		BS	2.4 (2.4)	80.24 (11.43)	69.59 (11.26)	90.22 (9.78)	8.72

Table 13. Effect of varying noise strength (variance) in CLUE on CIFAR-100 with ViT-Base. Numbers in parentheses indicate deviation from Retrain.

Dataset/ Architecture	Unlearning Method	UA	RA	TA	MIA	Avg. Gap
CIFAR-100 ViT-Base	Retrain	0	99.24	85.72	100	0
	CLUE ($\sigma^2 = 0.1$)	38.37 (38.37)	**95.33 (3.91)**	82.45 (3.27)	63.78 (36.22)	20.44
	CLUE ($\sigma^2 = 0.5$)	13.22 (13.22)	93.63 (5.61)	80.90 (4.82)	95.27 (4.73)	7.10
	CLUE ($\sigma^2 = 1.0$)	0 (0)	82.56 (16.68)	76.80 (8.92)	96.22 (3.78)	7.35
	CLUE (random $\sigma^2 \in [0.5, 1.0]$)	0.22 (0.22)	91.24 (8.00)	80.77 (4.95)	99.77 (26.33)	3.37

ture. We use the class id 6 as the forget set. The results in Table 14 show that Random Labelling, Boundary Expand, and Boundary Shrink either incur substantial drops in retained/test accuracy or fail to fully suppress the forgotten classes, leading to larger gaps (3.65–21.14). In contrast, CLUE achieves a balanced trade-off, combining low UA with minimal RA/TA degradation, and delivers the smallest gap (1.78), highlighting its superior stability and effectiveness.

Table 14. Comparison of unlearning methods for Caltech256 dataset and ViT base 16 architecture.

Unlearning Method	UA	RA	TA	MIA	Avg. Gap
Retrain	0	95.50	77.32	100	0
RL	1.25 (1.25)	89.40 (6.10)	72.85 (4.47)	98.60 (1.40)	3.65
BE	0 (0)	87.65 (7.85)	74.20 (3.12)	100 (0)	4.33
BS	3.80 (3.80)	90.72 (4.78)	75.15 (2.17)	96.05 (3.95)	5.12
ZSU	57.34 (57.34)	92.34(3.16)	72.44(4.88)	81.3(18.7)	21.135
SFU	10.34 (10.34)	92.33 (3.17)	74.67 (2.55)	93.4 (6.6)	5.665
CLUE	2.10 (2.10)	93.85 (1.65)	76.20 (1.12)	98.25 (1.75)	1.78

13. Additional Visualization for Subjective Evaluation

In order to better the visualization, we upsample the images and their corresponding attention map by 4 times. From the obtained visualizations, we can observe a shrink in the attention map for the forget class while the attention maps of the rest are unchanged.

14. Skeletal Code Unit for the CLUE

We provide the code units for implementing CLUE in Fig. 3 through Fig. 7.

Figure 3. In this part, we create a mask based on the saliency of the weights. We utilize only the forget set but this can be utilized for any dataset or subset of data.

```python
# Creates a mask from the forget set
def create_mask_from_gradients(model, test_loader, threshold=0.1):
    """
    Creates a mask for the weights of a model based on gradient information from the test data.
    Args:
        model: The neural network model.
        test_loader: DataLoader providing test data.
        threshold: The threshold of gradient values for creating the mask.
    Returns:
        A dictionary of boolean masks for each parameter.
    """
    model.eval()
    # Initialize the mask dictionary
    mask = {}
    for name, param in model.named_parameters():
        if param.requires_grad:
            mask[name] = torch.zeros_like(param, dtype=torch.bool)

    # Iterate through the test data
    for inputs, targets in test_loader:
        inputs, targets = inputs.to(next(model.parameters()).device), targets.to(next(model.parameters()).device)  # Move inputs to the model's device
        model.zero_grad()

        # Forward pass
        outputs = model(inputs)

        # We minimize the norm of the logits (outputs)
        loss = torch.nn.functional.cross_entropy(outputs, targets)

        # Backward pass to compute gradients
        loss.backward()

        # Update the mask based on the layer-wise percentile of gradient values
        for name, param in model.named_parameters():
            if param.grad is not None:
                # Compute Taylor criterion
                taylor_criterion = (param.data * param.grad.data).abs()

                # Update the mask with the computed threshold
                mask[name] = mask[name] | (taylor_criterion > threshold)

    return mask
```

Figure 4. This is the code to add gaussian noise to input data. This is one of the two noises we have experimented with.

```
def add_gaussian_noise(images, std_devs):
    """
    Adds Gaussian noise to a batch of images with varying standard deviations.

    Args:
        images (torch.Tensor): A batch of images of shape (batch_size, channels, height, width).
        std_devs (torch.Tensor): A tensor of shape (batch_size,) containing the standard deviation
                                 of the Gaussian noise for each image in the batch.

    Returns:
        torch.Tensor: A batch of images with Gaussian noise added.
    """
    # Ensure std_devs is the same device as images and reshape it for broadcasting
    std_devs = std_devs.view(-1, 1, 1, 1).to(images.device)

    # Generate noise with the same shape as images, with varying std_devs
    noise = torch.randn_like(images) * std_devs

    # Add noise to the images
    noisy_images = images + noise

    return noisy_images.clip(min=0, max=1)
```

Figure 5. This is the implementation of the second noise we have experimented with.

```

def add_salt_and_pepper_noise_batch(images, salt_prob=0.01, pepper_prob=0.01):
    # Create a copy of the original batch
    noisy_images = images.clone()

    # Iterate through each image in the batch
    for i in range(images.shape[0]):
        # Get the current image
        image = images[i]

        # Get the total number of pixels
        num_pixels = image.numel()

        # Generate random numbers for salt
        num_salt = int(num_pixels * salt_prob)
        coords = [np.random.randint(0, j - 1, num_salt) for j in image.shape]
        noisy_images[i, coords[0], coords[1]] = 1  # Set to white (salt)

        # Generate random numbers for pepper
        num_pepper = int(num_pixels * pepper_prob)
        coords = [np.random.randint(0, j - 1, num_pepper) for j in image.shape]
        noisy_images[i, coords[0], coords[1]] = 0  # Set to black (pepper)

    return noisy_images
```

Figure 6. We implement the steps described in section 3 of the manuscript in this code block

```
def ood_assisted_unlearning(model, train_loader, mask, optimizer):
    # Copy the pretrained model for both teacher and student

    teacher = deepcopy(model)

    # Freeze the teacher network
    for param in teacher.parameters():
        param.requires_grad = False

    teacher.eval()
    model.train()

    loss_fn = nn.KLDivLoss(log_target=True, reduction="batchmean")

    start = time.time()

    for it, (image, target) in enumerate(train_loader):
        i = it + len(train_loader)
        image = image.cuda()
        target = target.cuda()

        noisy_input = add_salt_and_pepper_noise_batch(image, salt_prob=0.5, pepper_prob=0.5)

        # Forward pass through the teacher network
        with torch.no_grad():
            teacher_output = teacher(noisy_input)


        student_output = model(image)


        loss = (student_output/student_output.norm(2, dim=-1).view(-1, 1)).exp().sum(-1).
            mean() + loss_fn(nn.functional.log_softmax(teacher_output, dim=-1), nn.
            functional.log_softmax(student_output, dim=-1))

        # Backpropagation on the student network
        optimizer.zero_grad()
        loss.backward()

        if mask:
            for name, param in model.named_parameters():
                if param.grad is not None:
                    param.grad *= mask[name]

        optimizer.step()
```

Figure 7. This code block is called for E_{iter} number of times to perform the unlearning

```
def ood_unlearning(data_loaders, model, criterion, optimizer, use_mask=True):
    #We assume the data_loaders is a dictionary with
    forget_loader = data_loaders["forget"]

    if use_mask:
        mask = create_mask_from_gradients(model, forget_loader, threshold=args.
            mask_threshold)


    # switch to train mode
    model.train()

    start = time.time()

    ood_assisted_unlearning(model, forget_loader, mask, optimizer)
```

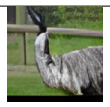
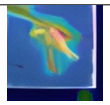
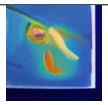
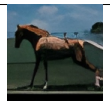
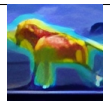
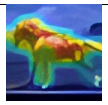
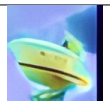
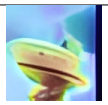
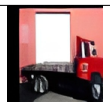
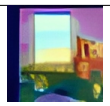
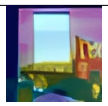
Class ID	Original Image	Attention Map Before Unlearning	Attention Map After Unlearning
Class:2 (Bird)			
Class:2 (Bird)			
Class:7 (Horse)			
Class:8 (Ship)			
Class:8 (Ship)			
Class:9 (Truck)			
Class:9 (Truck)			

Table 15. Visualization of the attention maps of ViT-Base for different classes of CIFAR10. Class #2 (Bird) is the forget class. We can observe that the gradient attention rollout map is significantly different for the forget class before and after unlearning while the same for the retain classes are almost the same.

References

[1] Sk Miraj Ahmed, Umit Yigit Basaran, Dripta S Raychaudhuri, Arindam Dutta, Rohit Kundu, Fahim Faisal Niloy, Basak Guler, and Amit K Roy-Chowdhury. Towards source-free machine unlearning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4948–4957, 2025. 6

[2] 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 (SP)*, pages 141–159, 2021. 1

[3] Jonathan Brophy and Daniel Lowd. Machine Unlearning for Random Forests. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1092–1104. PMLR, 18–24 Jul 2021. 1

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 4

[5] Yinzhi Cao and Junfeng Yang. Towards Making Systems Forget with Machine Unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480, 2015. 1

[6] Huiqiang Chen, Tianqing Zhu, Xin Yu, and Wanlei Zhou. Machine unlearning via null space calibration. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 358–366, 2024. 1

[7] Min Chen, Weizhuo Gao, Gaoyang Liu, Kai Peng, and Chen Wang. Boundary Unlearning: Rapid Forgetting of Deep Networks via Shifting the Decision Boundary. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7766–7775, June 2023. 1, 3

[8] Rishav Chourasia and Neil Shah. Forget Unlearning: Towards True Data-deletion in Machine Learning. In *International Conference on Machine Learning*, pages 6028–6073. PMLR, 2023. 1

[9] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Can Bad Teaching Induce Forgetting? Unlearning in Deep Networks Using an Incompetent Teacher. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7210–7217, 2023. 1

[10] Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Zero-shot Machine Unlearning. *IEEE Transactions on Information Forensics and Security*, 18:2345–2354, 2023. 1, 6

[11] Chongyu Fan, Jiancheng Liu, Yihua Zhang, Eric Wong, Dennis Wei, and Sijia Liu. SalUn: Empowering Machine Unlearning via Gradient-based Weight Saliency in Both Image Classification and Generation. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 4

[12] Jack Foster, Kyle Fogarty, Stefan Schoepf, Cengiz Öztireli, and Alexandra Brintrup. Zero-shot Machine Unlearning at Scale via Lipschitz Regularization. *arXiv preprint arXiv:2402.01401*, 2024. 1

[13] Jack Foster, Stefan Schoepf, and Alexandra Brintrup. Fast Machine Unlearning without Retraining through Selective Synaptic Dampening. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12043–12051, 2024. 1

[14] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making AI Forget You: Data Deletion in Machine Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 1

[15] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020. 1

[16] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pages 383–398. Springer, 2020. 1

[17] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified Data Removal from Machine Learning Models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3832–3842. PMLR, 13–18 Jul 2020. 1

[18] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive Machine Unlearning. *Advances in Neural Information Processing Systems*, 34:16319–16330, 2021. 1

[19] Jinghan Jia, Jiancheng Liu, Parikshit Ram, Yuguang Yao, Gaowen Liu, Yang Liu, Pranay Sharma, and Sijia Liu. Model Sparsity Can Simplify Machine Unlearning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 2

[20] Korbinian Koch and Marcus Soll. No Matter How You Slice It: Machine Unlearning with SISA Comes at the Expense of Minority Classes. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 622–637, 2023. 1

[21] Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 1

[22] Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023. 1

[23] Seth Neel, Aaron Roth, and Shahin Sharifi-Malvajerdi. Descent-to-Delete: Gradient-Based Methods for Machine

Unlearning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. 1

[24] Chao Pan, Jin Sima, Saurav Prakash, Vishal Rana, and Olgica Milenkovic. Machine Unlearning of Federated Clusters. In *The Eleventh International Conference on Learning Representations*, 2023. 1

[25] Chengyao Qian, Jing Wu, Trung Le, Dinh Phung, and Mehrtash Harandi. SUN: Training-free machine unlearning via subspace, 2024. 1

[26] Shauli Ravfogel, Yanai Elazar, Hila Gonen, Michael Twiton, and Yoav Goldberg. Null it out: Guarding protected attributes by iterative nullspace projection. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7237–7256, Online, July 2020. Association for Computational Linguistics. 1

[27] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember What You Want to Forget: Algorithms for Machine Unlearning. *Advances in Neural Information Processing Systems*, 34:18075–18086, 2021. 1

[28] Ayush K. Tarun, Vikram S. Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast Yet Effective Machine Unlearning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(9):13046–13055, 2024. 1

[29] Enayat Ullah, Tung Mai, Anup Rao, Ryan A. Rossi, and Raman Arora. Machine Unlearning via Algorithmic Stability. In Mikhail Belkin and Samory Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 4126–4142. PMLR, 15–19 Aug 2021. 1

[30] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 4

[31] Xu Wang, Zihao Li, Benyou Wang, Yan Hu, and Difan Zou. Model unlearning via sparse autoencoder subspace guided projections. In *ICML 2025 Workshop on Machine Unlearning for Generative AI*, 2025. 1

[32] Ross Wightman. PyTorch Image Models (timm). https://github.com/rwightman/pytorch-image-models, 2019. 3