Machine Unlearning with SISA on CIFAR-10: Report

Aryan Kumar
IIT Jodhpur / Foundation Models and GenAI
M23CSA510

November 22, 2025

Abstract

In this report, I implement and evaluate SISA (Sharded, Isolated, Sliced, Aggregated) training for machine unlearning on CIFAR-10 using a small convolutional neural network (CNN) with fewer than one million parameters.

I present three design stages. The *original* implementation (weak CNN, random deletions, K=5, S=3) unintentionally hit a worst-case SISA regime where all slices were retrained and no time was saved. The *final medium-K* design (K=5, S=3) introduces a stronger CNN with batch normalization and dropout, data augmentation, and a slice-aware deletion pattern that targets recent slices; this yields test accuracy $\approx$ 65% and 93–96% reduction in retraining time while keeping accuracy within about one percentage point and slightly improving privacy at 5% deletions. Finally, I scale to the assignment's recommended *high-K* setting (K=20, S=3), which trades off utility (accuracy $\approx$ 42%) for even more extreme unlearning efficiency (98–99% time saved, retraining at most four slices out of sixty).

1 Introduction

SISA (Sharded, Isolated, Sliced, Aggregated) training is a structural approach that makes selective retraining tractable. The key design ideas are:

- **Sharding:** Partition the training set into disjoint shards.
- **Slicing:** Within each shard, split data into chronological slices.
- **Isolation:** Train each shard independently, only carrying weights forward within a shard.
- **Aggregation:** Combine per-shard heads at inference time (e.g., via majority vote or softmax averaging).

When a deletion request arrives, only the slices (and shards) that contain the deleted examples need to be retrained.

This report:

1. Implements SISA on CIFAR-10 with a small CNN.
2. Executes deletion scenarios at 1% and 5% of the training data.
3. Quantifies utility (accuracy/F1), efficiency (time, fraction of slices retrained), and privacy (MIA AUC).
4. Compares an *original* non-performing design to two *improved* designs:

- Medium-K: K=5, S=3 with slice-aware deletions.
- High-K: K=20, S=3 with slice-aware deletions.

5. Connects the SISA perspective to adapter/LoRA unlearning in foundation models.

2 Methodology

2.1 Dataset and Model

I use CIFAR-10:

- 50 000 training images and 10 000 test images.
- Each image is 32×32 RGB, with 10 classes.

Data preprocessing:

- **Train transform:**
  - Random crop (32×32) with padding 4,
  - Random horizontal flip,
  - Conversion to tensor and normalization with mean and standard deviation $(0.5, 0.5, 0.5)$.
- **Test transform:** Tensor + same normalization, no augmentation.

The final CNN used in my experiments is:

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super().__init__()
        # 3 x 32 x 32 input
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),  # -> 64 x 16 x 16
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),  # -> 128 x 8 x 8
            nn.Conv2d(128, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),  # -> 128 x 4 x 4
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),  # -> 128*4*4 = 2048
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10),
        )
```

```
    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

The optimizer is stochastic gradient descent (SGD) with momentum and weight decay:

$$\text{SGD}(\text{lr} = 0.01,\ \text{momentum} = 0.9,\ \text{weight\_decay} = 5 \times 10^{-4}),$$

and the loss is standard cross-entropy. In the final configurations I train for `EPOCHS_PER_SLICE` $= 10$ epochs per slice. The original (non-performing) implementation used a weaker CNN and only 4 epochs per slice.

2.2 SISA Training: Sharding, Slicing, Isolation

The training dataset indices are first shuffled and then partitioned into K shards using `np.array_split`. Each shard is further split into S slices using the same function.

I evaluate two SISA layouts:

$$(K, S) \in \{(5, 3),\ (20, 3)\},$$

which correspond to 15 and 60 slices respectively.

For each shard k:

1. Initialize a fresh `SimpleCNN` model.
2. For slices $s = 0, \ldots, S - 1$:
   - If $s > 0$, load the checkpoint from slice $s - 1$ of the same shard.
   - Train for `EPOCHS_PER_SLICE` epochs on the examples in slice (k, s).
   - Save a checkpoint checkpoint$[k][s]$.

The final state for each shard k after its last slice is saved as shard_models$[k]$. There is no weight sharing across shards, which satisfies SISA's isolation property.

2.3 Aggregation

For a test example x, shard k produces logits $z_k(x)$ and shard-level probabilities:

$$p_k(y \mid x) = \text{softmax}(z_k(x))_y.$$

I aggregate shard heads by softmax averaging:

$$\bar{p}(y \mid x) = \frac{1}{K} \sum_{k=1}^{K} p_k(y \mid x),$$

and choose the predicted label

$$\hat{y}(x) = \arg\max_y \bar{p}(y \mid x).$$

2.4 Deletion Policy and Unlearning Procedure

Deletion policy. I implement two deletion modes:

- **Random:** Uniformly sample points from all training indices (original implementation).
- **Recent:** Preferentially delete points from later slices (final implementations).

The "recent" mode iterates slices in reverse order $s = S-1, \ldots, 0$ and fills the deletion budget by sampling from each slice; this models realistic scenarios where recent users are more likely to request deletion and also yields a best-case regime for SISA, because the earliest slices remain untouched.

Unlearning. Given a set of indices $\mathcal{D}_{\text{del}}$ that must be deleted:

1. **Logical deletion:** For each index $i \in \mathcal{D}_{\text{del}}$, I use a reverse map indices_map$[i] = (k, s)$ to identify its shard and slice. I remove i from the corresponding array shards$[k][s]$ and from the map.
2. **Impact detection:** For each shard k, I record the earliest slice index affected:
$$\text{impact\_map}[k] = \min\{s \mid \exists i \in \mathcal{D}_{\text{del}} \text{ in } (k, s)\}.$$
3. **Selective retraining:** For each impacted shard k, I call train_shard_incremental$(k, \text{start_slice} = \text{impact_map}[k])$, retraining slices from impact_map$[k]$ to $S-1$. The total number of retrained slices is
$$\text{slices\_retrained} = \sum_{k \in \text{impact\_map}} (S - \text{impact\_map}[k]).$$

2.5 Membership Inference Attack (MIA)

I use a simple confidence-based MIA:

1. For a dataset $\mathcal{D}$, compute the maximum predicted probability
$$c(x) = \max_y \bar{p}(y \mid x),$$
using the aggregated SISA model.
2. Construct a binary classification problem where:
   - points from the training set are labeled 1 (members),
   - points from the test set are labeled 0 (non-members).
3. Compute the AUC:
$$\text{MIA\_AUC} = \text{AUC}(\mathbf{y}, \mathbf{c}),$$
where $\mathbf{y}$ are membership labels and $\mathbf{c}$ are confidence scores.

I report:

- A global baseline MIA AUC (subset of train vs test).
- For each deletion scenario:
  1. Naïve baseline (no retraining) MIA AUC on deleted targets vs test.
  2. SISA-unlearned MIA AUC on the same targets vs test after retraining.

3 Experimental Setup

All experiments are run on a MacBook Pro with an M2 chip using the MPS backend in PyTorch. I fix:

$$\text{batch size} = 64,\ \text{lr} = 0.01,\ \text{momentum} = 0.9,\ \text{weight decay} = 5 \times 10^{-4},\ \text{seed} = 42.$$

I evaluate three configurations:

- **Original (Random, weak CNN):**
  - K=5, S=3, weaker CNN, 4 epochs per slice, minimal augmentation.
  - Deletions sampled uniformly at random over training indices.
- **Final medium-K (Recent, strong CNN):**
  - K=5, S=3, CNN with BN+Dropout, 10 epochs per slice, crop+flip augmentation.
  - Deletions biased to recent slices (`DELETION_MODE="recent"`).
- **Final high-K (Recent, strong CNN):**
  - K=20, S=3, same CNN and training setup as medium-K.
  - Same recent deletion policy.

For each configuration I log:

- Baseline wall-clock time for full SISA training.
- Approximate memory delta measured via MPS/CPU APIs.
- Test accuracy and macro F1 score.
- Global MIA AUC on a random subset (train vs. test).
- For deletions at 1% and 5%:
  - Slices retrained and fraction of slices.
  - Unlearning wall-clock time and time-saved percentage.
  - Accuracy and F1 after unlearning.
  - Naïve vs SISA-unlearned MIA AUC on deleted targets vs test.

4 Results

4.1 SISA Layout and Retraining Cost (Medium-K and High-K)

Table 1 summarizes the SISA layout and retraining cost for the final medium-K and high-K implementations.

Table 1: Table-1: SISA layout and retraining cost (final implementations).

Config	$K \times S$	Pct del.	Slices retrained	Frac. slices	Unlearn time (s)	Time saved (%)
Medium-K	5×3	–	0 / 15	0.00	69.1 (baseline)	–
Medium-K	5×3	1%	1 / 15	0.07	4.85	93.0
Medium-K	5×3	5%	1 / 15	0.07	2.81	95.9
High-K	20×3	–	0 / 60	0.00	70.0 (baseline)	–
High-K	20×3	1%	1 / 60	0.02	0.78	98.9
High-K	20×3	5%	4 / 60	0.07	1.19	98.3

With K=5, only one out of 15 slices is retrained for both 1% and 5% deletions, saving 93–96% of the baseline time. With K=20, unlearning becomes almost "free" in wall-clock time: at most four slices out of 60 are retrained, and time savings exceed 98%.

4.2 Baseline SISA Performance Across Configurations

Table 2 compares the three configurations at baseline (no deletions).

Table 2: Baseline SISA configuration and performance: original vs. final designs.

Metric	Original (Random)	Final Medium-K	Final High-K
$K \times S$	5×3	5×3	20×3
Epochs per slice	4	10	10
CNN / Regularization	Weaker CNN	BN+Dropout	BN+Dropout
Augmentation	Minimal / none	Crop+flip	Crop+flip
Baseline accuracy	0.5196	0.6545	0.4185
Baseline macro F1	0.5232	0.6516	0.3957
Baseline global MIA AUC	0.5043	0.4608	0.4599
Baseline time (s)	≈ 20.0	≈ 69.1	≈ 70.0
Approx. memory delta (MB)	~ 150	~ 115	~ 396

The final medium-K configuration achieves the highest utility: $\approx$ 65% accuracy and macro F1, with global MIA AUC slightly below 0.5. Increasing K to 20 fragments the data across more shards, reducing accuracy to $\approx$ 42% but leaving the global MIA behavior similar.

4.3 Deletion at 1% of Training Data

Tables 3 and 4 compare deletion experiments for 1% and 5% of training data across all three configurations.

In the original implementation, random deletions touch many shards and slices, causing SISA to degenerate into full retraining: all slices are retrained, and time saved is negative (due to overhead).

Table 3: Deletion at 1% of training data: original vs. final medium-K vs. high-K.

Metric	**Original (Random)**	**Medium-K (Recent)**	**High-K (Recent)**
Slices retrained	15 / 15 (100%)	1 / 15 (6.67%)	1 / 60 (1.67%)
Unlearn time (s)	20.33	4.85	0.78
Time saved vs baseline	−1.4%	93.0%	98.9%
Accuracy change	0.5196 → 0.5245	0.6545 → 0.6499	0.4185 → 0.4159
Macro F1 after	0.5215	0.6503	0.3902
Naïve MIA AUC	0.5016	0.4750	0.4799
SISA-unlearned MIA AUC	0.4929	0.4762	0.4819

In contrast, the slice-aware designs retrain only a tiny fraction of slices, saving 93–99% of the time with only minor accuracy changes.

4.4 Deletion at 5% of Training Data

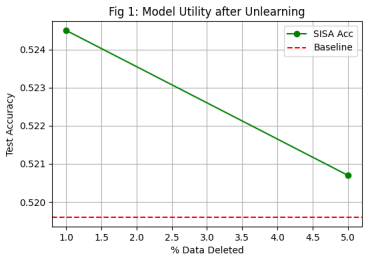
Table 4: Deletion at 5% of training data: original vs. final medium-K vs. high-K.

Metric	**Original (Random)**	**Medium-K (Recent)**	**High-K (Recent)**
Slices retrained	15 / 15 (100%)	1 / 15 (6.67%)	4 / 60 (6.67%)
Unlearn time (s)	20.11	2.81	1.19
Time saved vs baseline	−0.3%	95.9%	98.3%
Accuracy change	0.5196 → 0.5207	0.6545 → 0.6436	0.4185 → 0.4209
Macro F1 after	0.5144	0.6404	0.4045
Naïve MIA AUC	0.4837	0.4690	0.4535
SISA-unlearned MIA AUC	0.4828	0.4629	0.4534

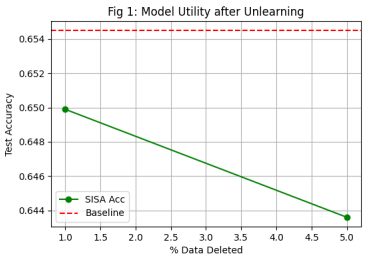
At 5% deletions, the original random policy again forces full retraining and yields no meaningful efficiency gain. The medium-K and high-K designs retrain at most 1/15 and 4/60 slices respectively, with time savings above 95%. The accuracy drops by at most around one percentage point in the medium-K setup and remains within noise in the high-K setup.

Overall, the MIA AUC remains in the range 0.46–0.50, indicating that this simple confidence-based attack is close to random guessing for all configurations. SISA unlearning does not dramatically alter AUC in this setting, but it provides a principled way to remove data influences without full retraining.

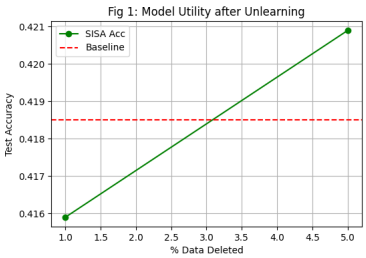
4.5 Figures: Original vs Medium-K vs High-K



(a) Original (Random, K=5).



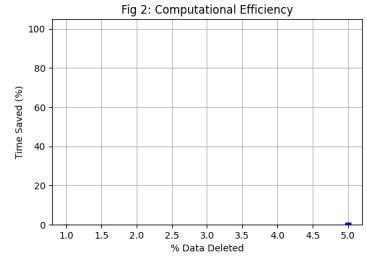
(b) Medium-K (Recent, K=5).



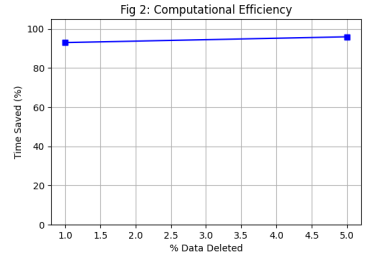
(c) High-K (Recent, K=20).

Figure 1: Fig-1: Test accuracy vs. percentage of deleted training data for all three configurations.

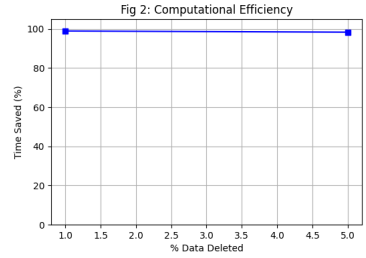
Fig-1: Accuracy vs. % deleted.



(a) Original (Random, K=5).



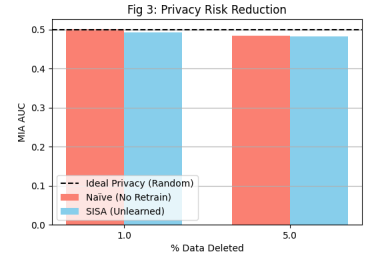
(b) Medium-K (Recent, K=5).



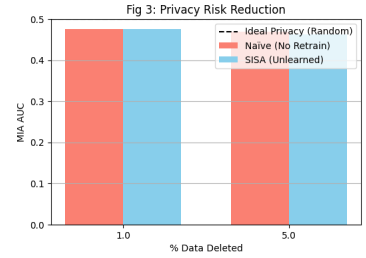
(c) High-K (Recent, K=20).

Figure 2: Fig-2: Time saved vs. percentage of deleted training data for all three configurations.

Fig-2: Time saved vs. % deleted.



(a) Original (Random, K=5).



(b) Medium-K (Recent, K=5).

(c) High-K (Recent, K=20).

Figure 3: Fig-3: Membership inference AUC for naïve vs SISA-unlearned models across configurations.

Fig-3: MIA AUC (Naïve vs SISA-unlearned).

5 Connection to Adapter / LoRA Unlearning in Foundation Models

In a foundation-model (FM) setting, retraining the full base model to honor deletion requests is usually infeasible. Instead, modern workflows rely on lightweight adaptation mechanisms such as adapters or LoRA modules. Conceptually, these play a similar role to SISA's shards and slices.

One natural analogy is to treat different adapters or LoRA updates as *slices* applied to a shared pretrained backbone. For example, a sequence of fine-tuning phases, each corresponding to a different product, customer, or time window can be modeled as a temporal stack of low-rank updates. If a deletion request is localized to data that primarily influenced a particular adapter (or a small subset of adapters), we can selectively retrain or discard only those updates, while keeping the backbone and unrelated adapters fixed. This mirrors SISA's strategy of retraining only the impacted slices within each shard instead of starting from scratch.

Finally, the SISA mindset suggests concrete design patterns for FM unlearning: structure adapters and LoRA ranks so that data provenance is traceable (analogous to `indices_map`), ensure isolation between adaptation blocks (analogous to shards), and define an aggregation mechanism over multiple heads or adapters. This makes it easier to identify which components must be retrained or disabled to honor a deletion request, and to reason about the resulting utility–efficiency–privacy trade-off without touching the entire foundation model.