TOWARDS SCALABLE EXACT MACHINE UNLEARNING USING PARAMETER-EFFICIENT FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine unlearning is the process of efficiently removing the influence of a training data instance from a trained machine learning model without retraining it from scratch. A popular subclass of unlearning approaches is *exact machine unlearning*, which focuses on techniques that explicitly guarantee the removal of the influence of a data instance from a model. Exact unlearning approaches use a machine learning model in which individual components are trained on disjoint subsets of the data. During deletion, exact unlearning approaches only retrain the affected components rather than the entire model. While existing approaches reduce retraining costs, it can still be expensive for an organization to retrain a model component as it requires halting a system in production, which leads to service failure and adversely impacts customers. To address these challenges, we introduce an exact unlearning framework – **S**equence-aware **S**harded **S**liced Training (S^3T), which is designed to enhance the deletion capabilities of an exact unlearning system while minimizing the impact on model's performance. At the core of S^3T, we utilize a lightweight parameter-efficient fine-tuning approach that enables parameter isolation by sequentially training layers with disjoint data *slices*. This enables efficient unlearning by simply deactivating the layers affected by data deletion. Furthermore, to reduce the retraining cost and improve model performance, we train the model on multiple data sequences, which allows S^3T to handle an increased number of deletion requests. Both theoretically and empirically, we demonstrate that S^3T attains superior deletion capabilities across a wide range of settings.

1 INTRODUCTION

In recent years, the growing success of machine learning (ML) has led to its widespread deployment across a range of applications (Achiam et al., 2023; Team et al., 2023; Qayyum et al., 2020; Surden, 2021). Once a machine learning model has been trained, it is often necessary to *unlearn* specific training data instances for various reasons, like complying with user data deletion requests (Mantelero, 2013; European Parliament & Council of the European Union; Shastri et al., 2019; Achille et al., 2024), removing stale or corrupt data (Biggio et al., 2012; Steinhardt et al., 2017), etc. Retraining an ML model entirely from scratch with each deletion request is expensive, especially for modern large-scale models (Brown et al., 2020; Achiam et al., 2023; Team et al., 2023). Machine unlearning (Nguyen et al., 2022; Xu et al., 2023) techniques focus on efficiently unlearning the influence of a data instance from a trained machine learning model.

Machine unlearning techniques are classified into two broad categories: approximate and exact unlearning (Xu et al., 2024). Approximate unlearning techniques (Guo et al., 2020; Liu et al., 2024a) modify the parameters of a trained model to reduce the influence of the deleted data instance. While cost-effective, approximate unlearning cannot guarantee the complete removal of an instance's influence and it may still retain non-zero influence on the model. Moreover, auditing approximate unlearning is challenging due to the stochastic nature of ML optimization (Thudi et al., 2022). An alternative approach is exact unlearning (Cao & Yang, 2015; Bourtoule et al., 2021; Golatkar et al., 2023), which can guarantee the removal of a data instance's influence from a trained model. Exact unlearning techniques use a modular system, where different components within the system are trained on disjoint data subsets. When a deletion request occurs, only the affected component needs to be retrained. However, in real-world settings, halting a production system to even retrain a single component can result in service failure. The alternative is to function without the affected component, which may result in reduced performance, ultimately impacting consumers. To address these challenges, we introduce a novel exact unlearning framework, **S**equence-aware **S**harded **S**liced Training (S^3T), which enhances the deletion capability while minimizing performance impact.

The key idea behind our S^3T framework is to perform additional offline training before deploying the initial model to reduce retraining costs. At the core of S^3T, we leverage a novel lightweight fine-tuning approach that allows parameter isolation by sequentially training model layers using disjoint data slices. Due to this parameter isolation, we efficiently perform exact unlearning by deactivating the layers associated with the deleted instance, rather than discarding the entire checkpoint. We efficiently train multiple models using different sequences of the same data slices depending on a training budget. We show that increasing the training budget before deployment can significantly reduce retraining costs and improve the model's performance. We also observe that it is important to train the model using diverse sequences and provide several approaches for selecting diverse sequences using graph matching (Cormen et al., 2022). Furthermore, we theoretically show that S^3T achieves provably better deletion guarantees than existing approaches. We conduct extensive empirical evaluations to evaluate the effectiveness of S^3T using Transformers with parameter counts ranging from 86M to 13B on a range of tasks. Additionally, we empirically validate the sequence selection algorithm and show that S^3T has superior deletion performance compared to existing methods.

The rest of the paper is organized as follows: **(a)** We introduce the prior literature related to approximate and exact machine unlearning (Section 2), **(b)** We describe the problem setup for exact unlearning (Section 3.1), **(c)** We introduce the fine-tuning approach, S^3T, and sequence selection algorithm under budget constraints (Section 3.2 & 3.3), **(d)** We theoretically analyze several properties of S^3T (Section 3.4), and **(e)** We present experiments to evaluate the effectiveness of S^3T's fine-tuning approach, deletion performance and sequence selection algorithm (Section 4).

2 Background

Machine unlearning techniques for deep learning is broadly classified into two categories: *approximate* and *exact* unlearning (Xu et al., 2024). Approximate unlearning techniques focus on reducing the influence of a deleted instance from a model after it has been trained. Exact unlearning techniques provide unlearning guarantees by ensuring model components trained on a deleted instance are not used during inference. In this section, we discuss each of these categories in detail.

Approximate Machine Unlearning. These techniques focus on approximating the model parameters as if the deleted data instance was not there in the training set from the beginning (Guo et al., 2020). These techniques typically quantify the influence of an instance (Koh & Liang, 2017) and perform gradient ascent for unlearning (Golatkar et al., 2020a;b; Neel et al., 2021; Sekhari et al., 2021; Gupta et al., 2021; Suriyakumar & Wilson, 2022; Liu et al., 2024a). In contrast to these approaches, (Graves et al., 2021) stores the exact gradients encountered during training and uses them directly for gradient ascent. Another line of work (Tarun et al., 2023a;b; Jia et al., 2023; Chen & Yang, 2023; Eldan & Russinovich, 2023; Patil et al., 2023; Kurmanji et al., 2024; Liu et al., 2024b) focuses on unlearning in a batch setting, where they assume access to both a retention set and a forget set of data instances for approximate unlearning. While efficient in practice, auditing approximate unlearning techniques is challenging due to the stochastic nature of the optimization process (Thudi et al., 2022; Wang et al., 2024) and may have weak privacy guarantees in practice (Hayes et al., 2024).

Exact Machine Unlearning. These techniques focus on developing a modular machine learning system, where individual components are trained using disjoint subsets of the data. Such a system offers the advantage that when a deletion request is received for an input instance, we only need to retrain the affected component rather than the entire model. However, these systems require modifying the original training process of the model. The seminal work for such a modular unlearning system is Sharded, Isolated, Sliced, and Aggregated training (SISA) (Bourtoule et al., 2021). SISA uses an ensemble of models each trained on a disjoint shard of the dataset as shown in Figure 1 (left). To further reduce retraining costs, each shard is divided into slices, and the models are incrementally trained on these slices, with their checkpoints stored sequentially (shown in Figure 1 (right)). If a deletion request is received for a data instance within 4^{th} slice of a shard, we must retrieve the checkpoint from the 3^{rd} slice and retrain using the remaining data within that shard. Several approaches focus on improving the components within SISA in application-specific settings like enhancing the dataset partitioning mechanism (Aldaghri et al., 2021; Yan et al., 2022), the retraining efficiency using light-weight adapters (Kumar et al., 2023; Dukler et al., 2023), or extending the

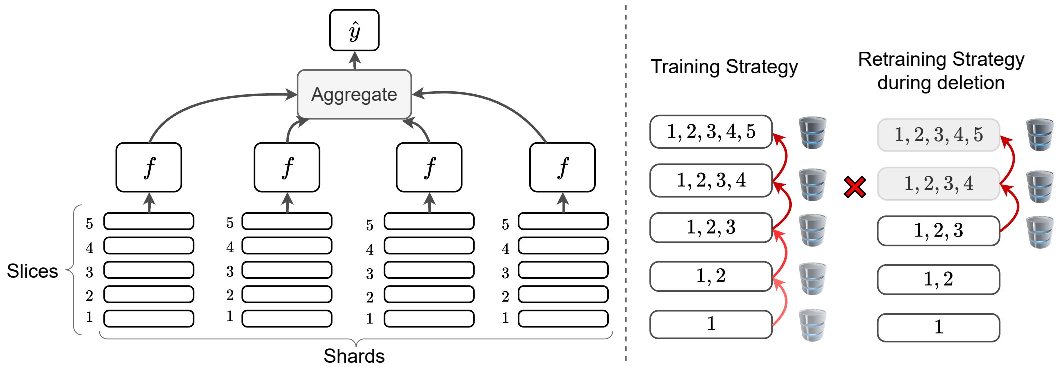

Figure 1: Schematic diagram of the Sharded, Isolated, Sliced, and Aggregated training (SISA) (Bourtoule et al., 2021) framework. An ensemble of models is individually trained on disjoint shards. (*Left*) Each shard is further divided into slices. (*Right*) Each model is sequentially trained on the slices and checkpoints are stored. After deletion, retraining resumes from the best available checkpoint.

modular approach for vision tasks by using compartmentalized diffusion models (Golatkar et al., 2023). However, these approaches are orthogonal to our work since they do not fundamentally modify the functioning of SISA and therefore can be easily incorporated within our proposed framework.

SISA is a well-known framework that can guarantee exact unlearning and has found widespread applications. However, using SISA within production systems is challenging because retraining even a single component would result in system downtime. Furthermore, in the worst-case scenario, if deletion requests impact the first slice in all the shards, the entire service goes down, necessitating retraining the model from scratch. In this work, we leverage parameter-efficient fine-tuning to introduce a framework that improves upon the service availability and deletion capabilities of SISA.

3 Sequence-aware Sharded Sliced Training (S^3T)

In this section, we describe the functioning of our proposed exact unlearning framework, **S**equence-aware **S**harded **S**liced Training (S^3T).

3.1 Problem Setting

We consider the general setting where the user fine-tunes a pre-trained model like BERT (Devlin et al., 2019) or Llama (Touvron et al., 2023) on private data using PEFT techniques. We assume that the deletion requests affect only the private fine-tuning data, not the pre-training data. In S^3T, we partition a dataset $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ into m disjoint shards. Each shard is further divided into L slices: $\mathcal{D}_i = \{S_1, \ldots, S_L\}$. S^3T trains a separate model per shard and uses their aggregate decision.

Existing unlearning frameworks SISA (Bourtoule et al., 2021) use a similar setup described above. In SISA, within each shard, the model is trained in multiple stages sequentially on the slices (training stages are Slice 1, Slice 1+2, and so on), and their checkpoints are stored. However, a key weakness of SISA is that if deletion requests affect Slice 1 of all shards, then the entire service goes down necessitating retraining from scratch. Another drawback of SISA is that individual models within the ensemble need to be retrained whenever a deletion request is executed. Retraining even on a single slice is expensive for large-scale models in production serving a huge customer base. A naive alternative would be to use the last known usable checkpoint and perform retraining after regular intervals. For example, in Figure 1 (right), if a deletion request arrives for a data instance in Slice 4, the model in production can be replaced with the checkpoint obtained after Slice 3. It is easy to see that the performance of the overall model will degrade with the number of deletion requests.

We present an exact unlearning framework, S^3T, to address these challenges. The core idea involves training several copies of each model (within the ensemble) that are trained using different slice sequences. When deletion requests occur, we utilize the model that minimizes performance degradation. To further reduce the training cost, we leverage PEFT techniques and present a novel sequential slice-wise fine-tuning strategy in Section 3.2. Our training strategy allows us to use the same model by deactivating certain layers without the need to swap checkpoints in case of a deletion. In the following sections, we introduce the fine-tuning strategy and sequence selection process.

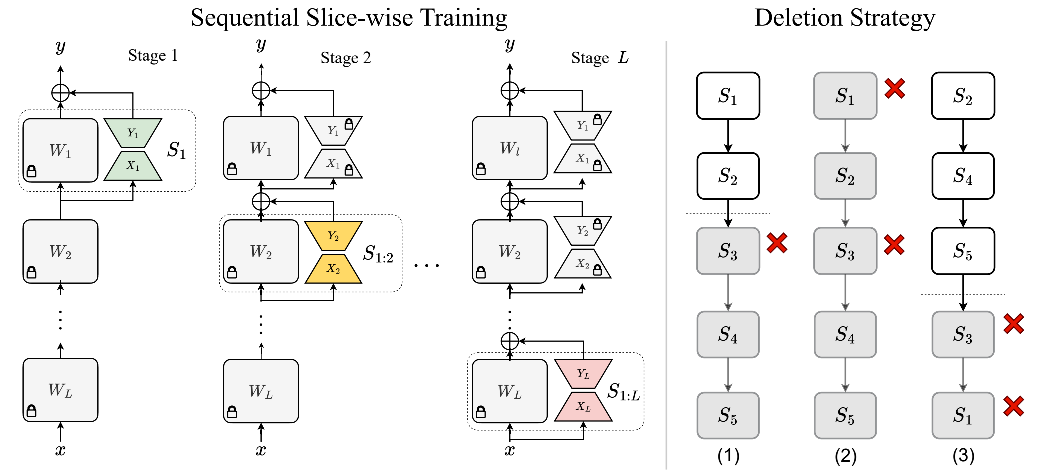

Figure 2: (*Left*) We show the schematic diagram of the slice-wise training strategy in S^{3}T. We incrementally train the model – i^{th} layer (from the top) using slices $S_{1:i}$ while keeping the other layers fixed. (*Right*) We show the impact of deletion on models trained on different permutations of slices.

3.2 Sequential Slice-wise Training

In this section, we introduce **S**equence-aware **S**harded **S**liced Training (S^{3}T) a lightweight fine-tuning approach for efficient exact unlearning. This fine-tuning approach enables parameter isolation by sequentially training PEFT layers using different data slices. Due to this parameter isolation, it is possible to efficiently handle deletion requests by deactivating layers associated with the instance.

We describe S^{3}T using the PEFT technique, LoRA (Hu et al., 2021), but our method is general can be easily extended to other PEFT techniques. LoRA introduces a small number of trainable low-rank ($r \ll d$) parameters, $(\mathbf{X}, \mathbf{Y})$, while the pre-trained weights $\overline{\mathbf{W}}$ remains fixed as shown below:

$$\mathbf{W} = \overline{\mathbf{W}} + \mathbf{X}^\top \mathbf{Y}, \text{ where } \mathbf{X}, \mathbf{Y} \in \mathbb{R}^{r \times d}, \overline{\mathbf{W}} \in \mathbb{R}^{d \times d}. \tag{1}$$

Our key idea involves training different LoRA layers using different data slices. This approach allows us to selectively deactivate (zero out) the LoRA parameters ($\mathbf{X}$ or $\mathbf{Y}$) associated with a particular layer in the event of data deletion from that slice. In Figure 2 (left), we illustrate the training process in detail, where we follow a sequential top-to-bottom training approach. At stage 1, we train the final model layer (Layer 1 in the figure) using slice 1 while LoRA parameters from all other layers are switched off. In the next stage, we train second last layer (Layer 2) using slices 1 & 2, while keeping the LoRA parameters from the Layer 1 frozen. This process continues for the rest of the layers. Note that the training does not need to proceed in a single layer-wise fashion, we can even train multiple LoRA layers per slice. We discuss more details about the design choice in Appendix C.2.

The sequential slice-wise training process ensures that the LoRA parameter updates at the i-th layer are a function of the data instances within slices $\{1, \ldots, i\}$. Therefore, if a deletion request affects the i-th slice, the same model can still be used by deactivating the LoRA parameters corresponding to slices $\{i, \ldots, L\}$ (see details in Algorithm 4). For example, if a deletion request affects slice S_3 only the subsequent LoRA layers need to be deactivated to ensure exact unlearning, as shown in the first example of Figure 2 (right). This is because during training the parameters of layers 1 & 2 were not affected by instances in S_3. During this deletion process, we use the same model checkpoint and switch off LoRA layers, resulting in an L-time reduction in storage cost compared to SISA.

Now we consider the scenario where deletion requests affect multiple slices. This is shown in the 2$^{\text{nd}}$ sequence of Figure 2 (right), where slices S_1 and S_3 are affected. In this case, we observe that a model trained on the default ordering of the sequence $\{S_1, \ldots, S_L\}$ is rendered useless when S_1 and S_3 are affected. This motivates us to train multiple models using different permutations of the slices. This would enhance the service time and system performance by selecting a model trained with the most effective ordering (e.g., the 3$^{\text{rd}}$ sequence in Figure 2 (right) yields the best-performing model). However, training on all $L!$ slice permutations is prohibitively expensive. In the following section 3.3, we present strategies to select a diverse set of permutations under budget constraints.

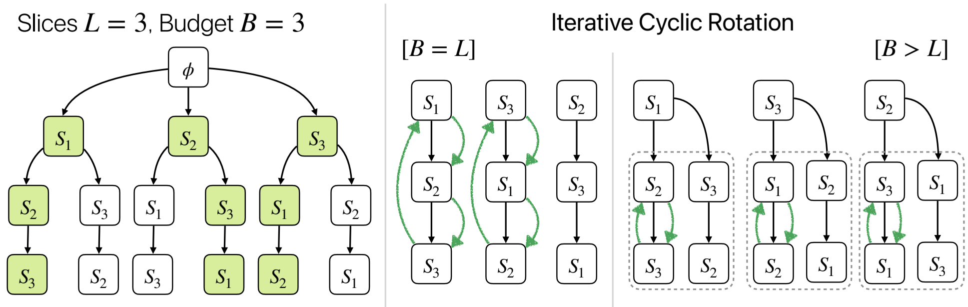

Figure 3: Illustration of the slice sequence selection problem with uniform deletion prior under a budget constraint, B. (*Left*) A permutation tree with $L = 3$ and a diverse set of sequences for budget $B = 3$ is shown in green. (*Center*) We show the functioning of the cyclic rotation algorithm, where we generate cyclic permutations of the original sequence. (*Right*) We iteratively extend the algorithm when budget $B > L$ by generating cyclic rotations of the subsequences.

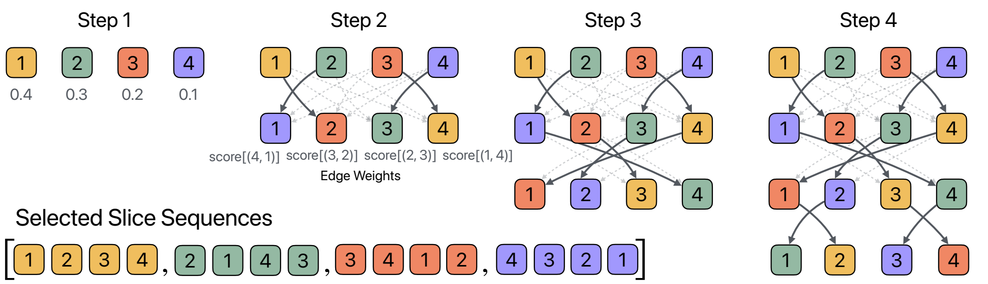

Figure 4: Illustration of the BMS algorithm. BMS selects one element for each permutation at a time. This is done by constructing a bipartite graph with all feasible edges to the next node, where edge weights are the current sequence scores. We compute the maximum weight matching on this graph. The dark gray arrows ($\rightarrow$) indicate the selected edges and dotted arrows ($\dashrightarrow$) the feasible ones.

Using these selection approaches, in Section 3.4 we theoretically show that we do not need more than L sequences to achieve the optimal deletion performance.

3.3 Training under Budget Constraints

In this section, we discuss strategies to select sequences under a budget, B (maximum number of sequences that can trained). First, we show that there exists an optimal subset of sequences and randomly selecting B sequences may not be effective. To illustrate this idea, we use a permutation tree (Bhattacharya, 1994), where all possible permutations are embedded into a tree structure.

In Figure 3 (left), we show an example of a permutation tree with $L = 3$ slices, paths from the root to the leaves correspond to unique permutation sequences (S_1, S_2, S_3), (S_1, S_3, S_2), and so on. We know that the topmost slice is most sensitive because if a deletion request affects the topmost slice the entire model needs to be retrained (shown in Figure 2 (right)). To address this and reduce the retraining cost, we should ensure we train models on sequences with different top slices. Building on this intuition, in the general setting we should train the model on diverse permutation sequences. Two sequences are considered *diverse if no element appears at the same position*, e.g., (S_1, S_2, S_3) and (S_2, S_3, S_1). An example is illustrated in Figure 3 (left), where a diverse set of 3 sequences is marked in green (where no identical slices occupy the same position). Selecting diverse permutations is challenging as relying solely on random sampling may not always yield the best results. Moreover, in certain scenarios, it is possible to have prior knowledge about the deletion probabilities of data slices; for example, younger users might be more likely to request data deletion than older users. Therefore, we present two strategies for selecting diverse permutation sequences for a budget B, depending on whether or not prior deletion probabilities are available.

Uniform Deletion Prior. In the setting, where each slice has a uniform (or unknown) deletion prior, we can generate diverse sequences by using cyclic permutations of the original sequence. Given a sequence (S_1, S_2, S_3), the cyclic permutations are (shown in Figure 3 (middle)):

$$(S_1, S_2, S_3) \rightarrow (S_3, S_1, S_2) \rightarrow (S_2, S_3, S_1). \quad (2)$$

The above approach guarantees that no element is repeated in the same position. However, it can only generate up to L different sequences. For budget $B > L$, we extend the cyclic rotation algorithm to generate more sequences. In Figure 3 (right), we generate new sequences by iterating over the existing sequences and performing cyclic rotations of the subsequences. For example, for (S_1, S_2, S_3) we perform cyclic rotation of the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ element to obtain the sequence: (S_1, S_3, S_2) (more examples in Figure 8). We provide the general *iterative cyclic rotation* algorithm in Appendix B.1.

Non-uniform Deletion Prior. In scenarios, where we have prior knowledge of the deletion probabilities, sequences generated by cyclic rotation may not be ideal. For example, consider the deletion probabilities are: (S_1: 0.5, S_2: 0.4, S_3: 0.1). Then, the first sequence in Eq. 2 is a bad choice because two of the slices most likely to be deleted are placed at the top. It is possible to select better sequences while satisfying the diversity criteria (no repeated slices at the same position). We score a sequence with deletion probabilities: $\mathcal{S} = (p_1, \ldots, p_L)$ by computing the expected number of functioning slices after t deletions: $\text{score}[\mathcal{S}, t] = \sum_{i=1}^{L} i. \left(1 - \sum_{j=1}^{i} p_j\right)^t$.

We present a bipartite-matching based sequence (BMS) selection algorithm. We will provide an intuitive explanation of the algorithm here and refer to Appendix B.2 for complete details. An illustration of BMS is shown in Figure 4. BMS iteratively selects elements of sequences by constructing a bipartite graph between one level to the next one (where edges are incident on feasible elements for the current sequence). The edges are weighted by the score of the current sequence, $\text{score}[\mathcal{S}, t]$. Selecting the next element then is equivalent to finding a maximum weight perfect matching (Galil, 1986) using the Hungarian algorithm (Kuhn, 1955) shown by the bold lines in Figure 4. This continues till all sequences have L elements. For a budget $B > L$, we use conditional sampling to randomly generate sequences according to their deletion probabilities (see Appendix B.2).

3.4 Theoretical Analysis

In this section, we theoretically analyze the performance of exact unlearning systems. For this, introduce the definition of *deletion rate* for exact unlearning systems.

Definition 1 (Deletion Rate)**.** *The deletion rate, $\delta(S)$, of an exact unlearning system S, is the expected number of deletion requests until the system needs to be retrained from scratch.*

The deletion rate captures the effectiveness of an exact unlearning system by quantifying the expected number of deletion requests it can handle. Next, we quantify the deletion rate for S^{3}T and SISA.

Lemma 1. *For dataset size $N \gg r$, where r is the number of deletion requests, the deletion rate of S^{3}T is $\delta(S^3T) \sim O(mL\log(m\min(B, L)))$ and for SISA it is $\delta(\text{SISA}) \sim O(mL\log m)$, where m is the number of shards and L is the number of slices per shard.*

This result shows that the deletion rate doesn't improve by increasing the budget B beyond L (proof in Appendix A.1). This shows that the optimal deletion rate can be achieved using *only L sequences* (instead of $L!$). Next, we analyze the impact of deletion requests on the system's performance. We perform a fine-grained analysis focusing on the performance of an individual shard. In this setting, we consider the real-world scenario where we do not retrain every time a slice is impacted instead work with the best available model. For S^{3}T, this means switching off the necessary PEFT layers, while for SISA, it means reverting to the best available checkpoint. The unlearning system experiences performance degradation with an increasing number of deletion requests, as we are compelled to utilize a model trained on fewer data slices (we show this empirically in Section 4). To quantify the performance retention we use a monotonically increasing function $F(k)$, which indicates a model's performance when trained on k slices. The exact formulation of $F(\cdot)$ depends on several factors like the dataset, model size, etc. We analyze the performance retention while processing deletion requests.

Lemma 2 (Performance Retention)**.** *Given a set of randomly selected $B \geq 1$ sequences and uniform deletion prior of slices, the difference between the probability that a shard retains a performance, $F_r(\cdot)$, of at least $F(k)$ after r deletion requests between S^{3}T and SISA is shown below*

$$\forall k \in [1 \ldots L],\ |\mathbb{P}\left[F_r(S^3T) \geq F(k)\right] - \mathbb{P}\left[F_r(\text{SISA}) \geq F(k)\right] = \zeta(1 - \zeta^{B'-1}), \tag{3}$$

where $\zeta = 1 - (1 - k/L)^r$ is a positive fraction and $B' = \min\left\{B, \frac{L!}{(L-k)!}\right\}$.

This above result shows that compared to SISA, S^{3}T enhances performance by increasing the probability that the system maintains at least $F(k)$ by a factor of B' (proof in Appendix A.2).

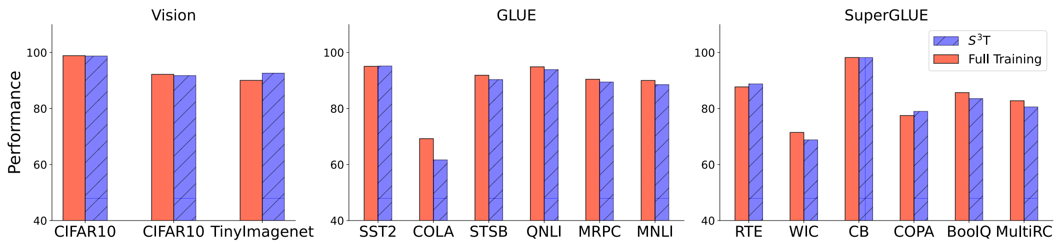

Figure 5: Comparison of the performance between S^{3}T training and full training (FT) on vision, GLUE & SuperGLUE benchmarks. We report the Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. We observe that S^{3}T achieves similar performance to FT.

4 Experiments

In this section, we outline the experimental setup and evaluate the unlearning performance of S^{3}T across various setups. Specifically, we design experiments to answer the following research questions:

(**RQ1**) Does S^{3}T training (Section 3.2) impact the model's performance compared to full training?
(**RQ2**) Does S^{3}T enhance the deletion capabilities of unlearning, and what is its cost tradeoff?
(**RQ3**) Is the sequence permutation selection algorithm (Section 3.3) effective in practice?

Sequential Slice-wise Training Performance. The objective of the experimental setup is to demonstrate that S^{3}T can achieve performance comparable to full training. The goal of S^{3}T is to achieve parameter isolation for data slices without impacting the overall performance. We perform a range of experiments with different Transformer model sizes ranging from 86M up to 13B. The details of the experimental setup are in Appendix C.1.

In Figure 5, we report the performance on vision, GLUE, and SuperGLUE benchmarks. We use ViT$_{\text{BASE}}$ (Dosovitskiy et al., 2020) (for CIFAR10 & CIFAR100 (Krizhevsky et al., 2009)), ViT$_{\text{LARGE}}$ (for Tiny ImageNet (Le & Yang, 2015)), and RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019) (for GLUE (Wang et al., 2018) & SuperGLUE (Wang et al., 2019)). We observe that S^{3}T achieves comparable performance to full training (FT) across all settings. In some of the settings, we also observe that S^{3}T is able to outperform FT (e.g., S^{3}T obtains 2.5% accuracy gain on TinyImagenet using `ViT-large`). Next, we conduct experiments to evaluate the effectiveness of S^{3}T while using large language models (LLMs). Specifically, we perform instruction tuning of Llama2-7B (Touvron et al., 2023), Llama2-13B, and Llama3-8B using Alpaca dataset (Taori et al., 2023). Then, we evaluate each instruction-tuned model on a range of tasks to evaluate the model's general/world knowledge (MMLU (Hendrycks et al., 2020), OpenBookQA (Mihaylov et al., 2018)), truthfulness in QA (TruthfulQA (Lin et al., 2022)), and commonsense reasoning (PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2021), ARC (Clark et al., 2018)). We use LLM-evaluation suite (Gao et al., 2023) to report the performance and report the zero-shot performance for all datasets. Similar to the previous setup, in Table 3, we observe that S^{3}T achieves comparable performance to full finetuning across all datasets and even outperforms FT on many datasets across different model sizes. These experiments provide the answer to (**RQ1**) demonstrating that S^{3}T is an effective way to achieve parameter isolation for data slices without impacting the model's performance.

Deletion Performance. In this section, we evaluate the performance of S^{3}T as deletion requests are received. In Figure 6 (left), we report the performance of S^{3}T and baselines SISA (Bourtoule et al., 2021) and ProtoSISA (Yan et al., 2022) ($m = 5$ shards, $L = 4$ slices) on CIFAR-10 and CIFAR-100 datasets. In this experiment, we use a uniform deletion prior over slices. We also report the performance of full re-training, which retrains the model after each deletion request and serves as an upper performance bound. We report S^{3}T's performance under various budgets. We observe that S^{3}T can achieve very close performance to the full budget ($B = 24$) with a significantly smaller budget, $B = 4$. For SISA and ProtoSISA, we observe that the plot ends after approximately 40 deletion requests as these systems do not have functioning models beyond this point. We observe that S^{3}T can handle more deletion requests while consistently outperforming the baseline approaches. Note that increasing the budget $B > L$ does not help improve the deletion rate but increases the probability of a better-performing model (as observed in Lemma 1). Next, we extensively evaluate the impact of increasing the training budget B on the deletion rate (with $m = 5$ shards & $L = 64$ slices). In Figure 6 (right), we observe that there is a steady growth in the deletion rate with an increasing

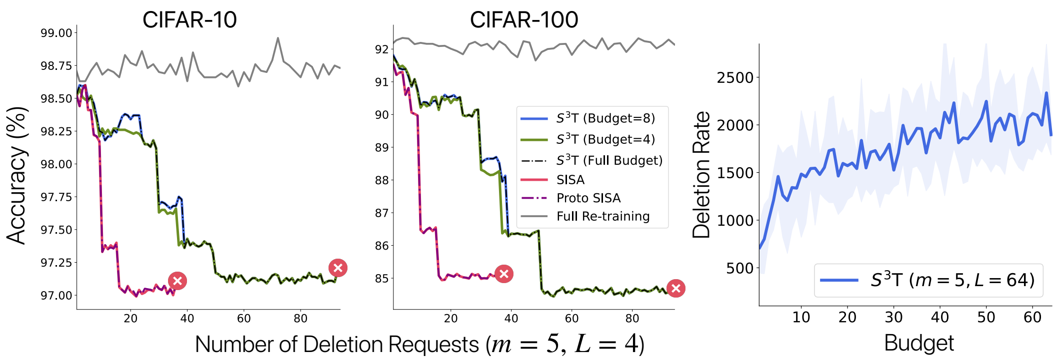

Figure 6: (*Left*) We report the impact on performance of S³T and baselines with an increasing number of deletion requests. S³T handles a higher number of deletion requests while maintaining high performance with a relatively low budget (⊗ indicates the failure point for the system). (*Right*) We report the deletion rate of S³T with an increasing budget and observe steady growth.

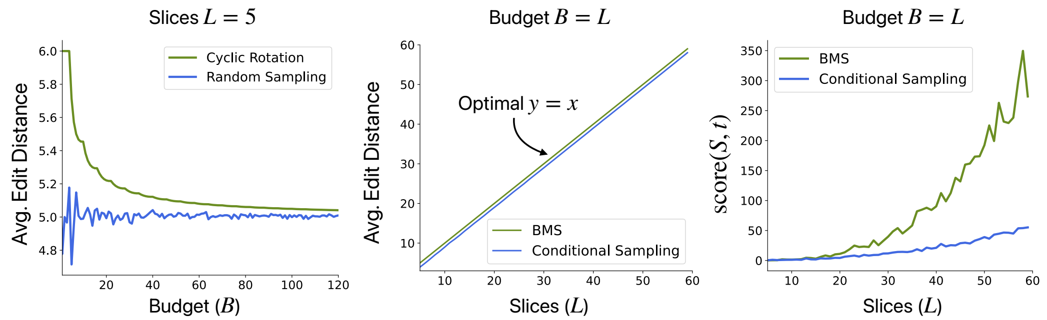

Figure 7: We evaluate the performance of iterative cyclic rotation and bipartite matching-based selection (BMS). (*Left*) We observe that cyclic rotation selection consistently outperforms random sampling for all budgets, $1 \leq B \leq 120$ (with fixed $L = 5$). (*Center*) We evaluate the average edit distance of the sequences generated by BMS and observe that it achieves the optimal edit distance (L). (*Right*) We also observe that sequences from BMS achieves higher scores than random sampling.

budget. The growth is slightly higher in the initial stages when the budget is low and slows down gradually. This experiment provides empirical evidence to our theoretical result in Lemma 2, which claims that the performance improves with an increased budget.

Sequence Selection. We evaluate the quality of the sequences generated by the iterative cyclic rotation and BMS algorithm (Section 3.3). Ideally, we want the selected sequences to be diverse and have a high edit distance between sequences. Therefore, we report the average edit distance within a selected subset, O: $\mathbb{E}_{o \in O}[d_{\text{edit}}(o, o')]$. First, when the deletion prior is uniform and compare cyclic rotation with random sampling. In Figure 7 (left), we report the average edit distance with varying budget (B) while the slice count (L) is fixed. We observe that cyclic rotation produce significantly better sequences than random sampling. Second, we consider slices associated with a deletion prior. We present the results by averaging over 10 deletion priors sampled from a Dirichlet distribution. In Figure 7 (center & right), we observe that BMS consistently outperforms random sampling both in terms of diversity (avg. edit distance) and chosen sequence scores ($\text{score}[\mathcal{S}, t]$).

5 Conclusion

In this paper, we introduced S³T, an effective framework for performing exact unlearning. S³T uses a modular machine learning model that is trained using disjoint shards of the data. Each shard is used to train a different model using a lightweight fine-tuning approach that enables parameter isolation, which allows us to execute unlearning requests efficiently. The key idea behind S³T is to train multiple models using different sequences of slices before deploying the system. This helps reduce retraining costs and improve the model's performance while the model is in production. Both theoretically and empirically, we show that S³T has significantly improved deletion capabilities compared to existing approaches. Future work can focus on developing techniques for finer-grained parameter isolation to further improve S³T's performance.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Alessandro Achille, Michael Kearns, Carson Klingenberg, and Stefano Soatto. Ai model disgorgement: Methods and choices. *Proceedings of the National Academy of Sciences*, 121(18): e2307304121, 2024.

Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. Coded machine unlearning. *IEEE Access*, 9:88137–88150, 2021.

P Bhattacharya. The representation of permutations by trees. *Computers & Mathematics with Applications*, 28(9):67–71, 1994.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Gunnar Blom, Lars Holst, and Dennis Sandell. 7.5 coupon collecting i, 7.6 coupon collecting ii, and 15.4 coupon collecting iii. *Problems and Snapshots from the World of Probability, New York: Springer-Verlag*, pp. 85–87, 1994.

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)*, pp. 141–159. IEEE, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

Jiaao Chen and Diyi Yang. Unlearn what you want to forget: Efficient unlearning for llms. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Yonatan Dukler, Benjamin Bowman, Alessandro Achille, Aditya Golatkar, Ashwin Swaminathan, and Stefano Soatto. Safe: Machine unlearning with shard graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17108–17118, 2023.

Ronen Eldan and Mark Russinovich. Who's harry potter? approximate unlearning in llms. *arXiv preprint arXiv:2310.02238*, 2023.

European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council. URL https://data.europa.eu/eli/reg/2016/679/oj.

Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38, 1986.

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

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*, pp. 9304–9312, 2020a.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *European Conference on Computer Vision*, pp. 383–398, 2020b.

Aditya Golatkar, Alessandro Achille, Ashwin Swaminathan, and Stefano Soatto. Training data protection with compositional diffusion models. *arXiv preprint arXiv:2308.01937*, 2023.

Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11516–11524, 2021.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In *International Conference on Machine Learning*, pp. 3832–3842. PMLR, 2020.

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.

Jamie Hayes, Ilia Shumailov, Eleni Triantafillou, Amr Khalifa, and Nicolas Papernot. Inexact unlearning needs more careful evaluations to avoid a false sense of privacy. *arXiv preprint arXiv:2403.01218*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2020.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

Jinghan Jia, Jiancheng Liu, Parikshit Ram, Yuguang Yao, Gaowen Liu, Yang Liu, Pranay Sharma, and Sijia Liu. Model sparsification can simplify machine unlearning. *arXiv preprint arXiv:2304.04934*, 2023.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Vinayshekhar Bannihatti Kumar, Rashmi Gangadharaiah, and Dan Roth. Privacy adhering machine un-learning in nlp. In *Findings of the Association for Computational Linguistics: IJCNLP-AACL 2023 (Findings)*, pp. 268–277, 2023.

Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. *Advances in Neural Information Processing Systems*, 36, 2024.

Jeffrey Lagarias. Euler's constant: Euler's work and modern developments. *Bulletin of the American Mathematical Society*, 50(4):527–628, 2013.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL https://aclanthology.org/2022.acl-long.229.

Jiaqi Liu, Jian Lou, Zhan Qin, and Kui Ren. Certified minimax unlearning with generalization rates and deletion capacity. *Advances in Neural Information Processing Systems*, 36, 2024a.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Alessandro Mantelero. The eu proposal for a general data protection regulation and the roots of the 'right to be forgotten'. *Computer Law & Security Review*, 29(3):229–235, 2013.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018. URL https://api.semanticscholar.org/CorpusID:52183757.

Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pp. 931–962. PMLR, 2021.

Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Vaidehi Patil, Peter Hase, and Mohit Bansal. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. In *The Twelfth International Conference on Learning Representations*, 2023.

Adnan Qayyum, Junaid Qadir, Muhammad Bilal, and Ala Al-Fuqaha. Secure and robust machine learning for healthcare: A survey. *IEEE Reviews in Biomedical Engineering*, 14:156–180, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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.

Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. The seven sins of {Personal-Data} processing systems under {GDPR}. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in neural information processing systems*, 30, 2017.

Harry Surden. Machine learning and law: An overview. *Research Handbook on Big Data Law*, pp. 171–184, 2021.

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

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023a.

Ayush Kumar Tarun, Vikram Singh Chundawat, Murari Mandal, and Mohan Kankanhalli. Deep regression unlearning. In *International Conference on Machine Learning*, pp. 33921–33939. PMLR, 2023b.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

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

Nobuaki Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Cheng-Long Wang, Qi Li, Zihang Xiang, and Di Wang. Has approximate machine unlearning been evaluated properly? from auditing to side effects. *arXiv preprint arXiv:2403.12830*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, and Philip S. Yu. Machine unlearning: A survey. *ACM Comput. Surv.*, 56(1), aug 2023. ISSN 0360-0300. doi: 10.1145/3603620. URL https://doi.org/10.1145/3603620.

Jie Xu, Zihan Wu, Cong Wang, and Xiaohua Jia. Machine unlearning: Solutions and challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2024.

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. Arcane: An efficient architecture for exact machine unlearning. In *IJCAI*, volume 6, pp. 19, 2022.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

A MATHEMATICAL PROOFS

CONTENTS

A.1 PROOF OF LEMMA 1

In S^{3}T, since the sequences are selected to be diverse (Figure 4), the topmost slice in each sequence is different. Therefore, we have $B' = \min(B, L)$ different slices at the topmost position. This implies that for S^{3}T to encounter service failure, a total of mB' slices must be affected by deletion requests, where m is the number of shards. Considering deletion requests affect all slices uniformly we need to compute the expected time till all slices are affected. This setup is similar to the coupon collector problem (Blom et al., 1994).

Proof. The deletion rate is the expected number of requests to delete all mB' slices (m shards, B' unique slices per shard). Using linearity of expectation, the deletion rate or total time is:

$$\delta(S^3T) = \mathbb{E}[T] = \mathbb{E}[t_1 + \ldots + t_{mB'}] \tag{4}$$
$$= \mathbb{E}[t_1] + \ldots + \mathbb{E}[t_{mB'}], \tag{5}$$

where $\mathbb{E}[t_i] = 1/p_i$, where p_i is the probability that the i-th slice is affected after $(i-1)$ slices are deleted. Let $N = |\mathcal{D}|$ the dataset size and r be the number of deletion requests seen so far. The expression of p_i is shown below:

$$p_i = \frac{\{mB' - (i-1)\}s_b}{N - r} \tag{6}$$
$$p_i = \frac{N\min(B/L, 1) - (i-1)s_b}{N - r} \tag{7}$$
$$p_i = \frac{\min(B/L, 1) - (i-1)/(mL)}{1 - r/N} \tag{8}$$
$$p_i \geq \frac{mB' - (i-1)}{mL}, \tag{9}$$

where s_b is the size of each slice. Replacing this result in Eq. 5, we get:

$$\delta(S^3T) \leq \frac{mL}{mB' - 0} + \frac{mL}{mB' - 1} + \ldots + \frac{mL}{1} \tag{10}$$
$$= mL\left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{mB'}\right) \tag{11}$$
$$= mL.H_{mB'} \tag{12}$$
$$= mL\log(mB') + \gamma mL + \frac{1}{2} + O\left(\frac{L}{mB'}\right), \tag{13}$$

where $H_{mB'}$ denotes the mB'-th Harmonic number and γ is the Euler's constant (Lagarias, 2013). The above result proves the first portion of the lemma, $\delta(S^3T) \sim O(mL\log mB')$.

The second part of the lemma is about SISA. For SISA to experience failure, only the first slice of each shard (total of m slices) needs to be affected by deletion. In this case, we can write:

$$p_i = \frac{\{m - (i-1)\}s_b}{N - r} = \frac{N/L - (i-1)s_b}{N - r} = \frac{1 - (i-1)/m}{L(1 - r/N)} \geq \frac{m - (i-1)}{mL} \tag{14}$$

Replacing the above result in Eq. 5, we get:

$$\delta(\text{SISA}) \leq mL\left(\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{m}\right) \tag{15}$$
$$= mL.H_m \tag{16}$$
$$= mL\log(m)+\gamma m+\frac{1}{2}+O\left(\frac{1}{m}\right). \tag{17}$$

This proves the second portion of the lemma: $\delta(\text{SISA}) \sim O(mL\log m)$. □

A.2 Proof of Lemma 2

We begin by proving the retention result for SISA as it is simpler to understand. Each model within SISA is trained on a single sequence of slices.

Proof. The probability it maintains performs better than $F(k)$ is equivalent to showing that none of the top-k elements (out of L) are affected after r deletion requests:

$$\mathbb{P}\left[F_r(\text{SISA}) \geq F(k)\right] = \left(1-\frac{k}{L}\right)^r. \tag{18}$$

In S^3T, each model is trained on B such sequences. Therefore, we need to compute the probability that at least one sequence is better than $F(k)$, which is:

$$\mathbb{P}\left[F_r(S^3T) \geq F(k)\right] = 1-\left(1-\left(1-\frac{k}{L}\right)^r\right)^B. \tag{19}$$

However, the above result suggests that the probability can be increased indefinitely by increasing B. This is inaccurate because to maintain a performance of $F(k)$ there has to be at least one prefix of length k that has not been affected. Since there only $P(L,k)=\frac{L!}{(L-k)!}$ permutations of length k extending the budget beyond $P(L,k)$ doesn't work. Therefore, the correct probability is:

$$\mathbb{P}\left[F_r(S^3T) \geq F(k)\right] = 1-\left(1-\left(1-\frac{k}{L}\right)^r\right)^{B'}, \tag{20}$$

where $B' = \min\{B, P(L,k)\}$.

Taking the difference between Eq. 20 and Eq. 18 and setting $\alpha = (1-k/L)^r$, we get:

$$\begin{aligned}\mathbb{P}\left[F_r(S^3T) \geq F(k)\right] - \mathbb{P}\left[F_r(\text{SISA}) \geq F(k)\right] &= 1-(1-\beta)^{B'}-\beta \\ &= (1-\beta)\{1-(1-\beta)^{B'-1}\} \\ &= \zeta(1-\zeta^{B'-1}),\end{aligned} \tag{21}$$

where $\zeta = 1-\alpha = 1-(1-k/L)^r$. This completes the proof. □

Discussion. In the above proof, we assumed that B sequences are selected randomly. In practice, we select diverse sequences using the iterative cyclic rotation algorithm. However, deriving a closed-form theoretical performance bound for the sequences generated using cyclic rotation is non-trivial. Intuitively, we expect the performance to be better as selecting diverse sequences means that the probability of a length-k prefix getting affected is reduced. Empirically, we observe performance improvements in Figure 7. We leave the theoretical proof of these results to future work.

A.3 Proof of Lemma 3

This lemma states that BMS selects the most diverse sequences. First, we revisit our definition of sequence diversity. Two sequences are considered diverse if no element appears at the same position, e.g., (S_1, S_2, S_3, S_4) and (S_2, S_3, S_4, S_1). Since we want diverse sequences, BMS performs maximum weight perfect matching, ensuring that no two elements appear in the same position. Therefore, in this proof, we show that perfect matching exists at all levels, $[1, L]$, in Algorithm 2.

Proof. Let the bipartite graph, G, consist of vertices $V \cup V'$, such that the edges $E \subseteq V \times V'$. For a perfect matching to exist, every subset $W \subset V$ should satisfy:

$$|W| \leq N_G(W), \tag{22}$$

where $N_G(\cdot)$ is the neighbourhood defined using the graph, G.

In our setup, the graph G has vertices $V = \{1, \dots, L\}$ and $V' = \{1', \dots, L'\}$, which indicate the elements in the permutation sequence. Every vertex $v \in V$ has $(L - i + 1)$ edges to unique vertices in V' at the i-th iteration of the algorithm. Therefore, for all iterations $i \in \{2, \dots, L\}$ (shown in Line 6 in Algorithm 2) the condition in Eq. 22 is satisfied and a perfect matching exists.

Since perfect matching selects a unique element in V', therefore no element is repeated at the same position in the BMS output. Therefore, BMS generates diverse sequences. $\square$

Algorithm 1 Iterative Cyclic Rotation

1: **function** CYCLICPERMUTATION(Permutation P)
2: $\quad\mathcal{R} = \{\}$ // set of all rotations
3: $\quad$**for** $i \in \{1, \dots, P.\text{length}\}$ **do**
4: $\quad\quad\mathcal{R} = \mathcal{R} \cup P$
5: $\quad\quad P = \text{rotateRight}(P)$
6: $\quad$**end for**
7: $\quad$**return** $\mathcal{R}$
8: **end function**
9:
10: **function** ITERATIVECYCLICROTATION(Slice count: L, Budget: B)
11: $\quad\mathcal{O} = \text{CyclicPermutation}([1, \dots, L])$ // initialize the set with L cyclic permutations
12: $\quad n_{\text{iter}} = 0$ // set the iteration count
13: $\quad$**while** $|\mathcal{O}| < B$ **do**
14: $\quad\quad n_{\text{iter}} = n_{\text{iter}} + 1$
15: $\quad\quad$**for** $o \in \mathcal{O}$ **do** // iteratively expand each of the existing permutation
16: $\quad\quad\quad \text{prefix}, \text{suffix} = o[: n_{\text{iter}}], o[n_{\text{iter}} :]$ // set the prefix and suffix
17: $\quad\quad\quad$// rotate the suffix with same prefix
18: $\quad\quad\quad \mathcal{P} = \{\text{prefix} \cup p$ for $p \in \text{CyclicPermutation}(\text{suffix})\}$
19: $\quad\quad\quad \mathcal{O} = \mathcal{O} \cup \mathcal{P}$
20: $\quad\quad$**end for**
21: $\quad$**end while**
22: $\quad$**return** $\mathcal{O}[: B]$ // return B output permutations
23: **end function**

B IMPLEMENTATION DETAILS

In this section, we discuss the details of various algorithms and workflows within S^{3}T.

CONTENTS

B.1 ITERATIVE CYCLIC ROTATION

In the setting, where the deletion probabilities are uniform we use the cyclic rotation algorithm to select diverse permutations. In Figure 3 (middle), we observe that we can easily generate L diverse sequences using cyclic permutation of the original sequence. We can iteratively expand on this idea to generate more sequences when the budget, $B > L$. An illustration of the different sequences selected using iterative cyclic rotation for different budgets is shown in Figure 8.

We present the pseudocode of the iterative cyclic rotation mechanism in Algorithm 1. First, we set the initial set to cyclic permutations of the original sequence (Line 11). If the budget exceeds the current set of output permutations, we iteratively expand the selected permutations by rotating their suffixes (Line 18). This continues till the output set has at least B sequences. Please note that Algorithm 1 is a simplified version of the actual algorithm. For some corner case budget values, we apply certain heuristics to select the right permutations to expand (Line 18). Empirically, we observe that iterative cyclic permutation significantly outperforms random sampling (Figure 7). We conjecture that iterative cyclic rotation generates the most diverse set of B permutations when all L slices have equal deletion probabilities. We will leave the theoretical proof to future research.

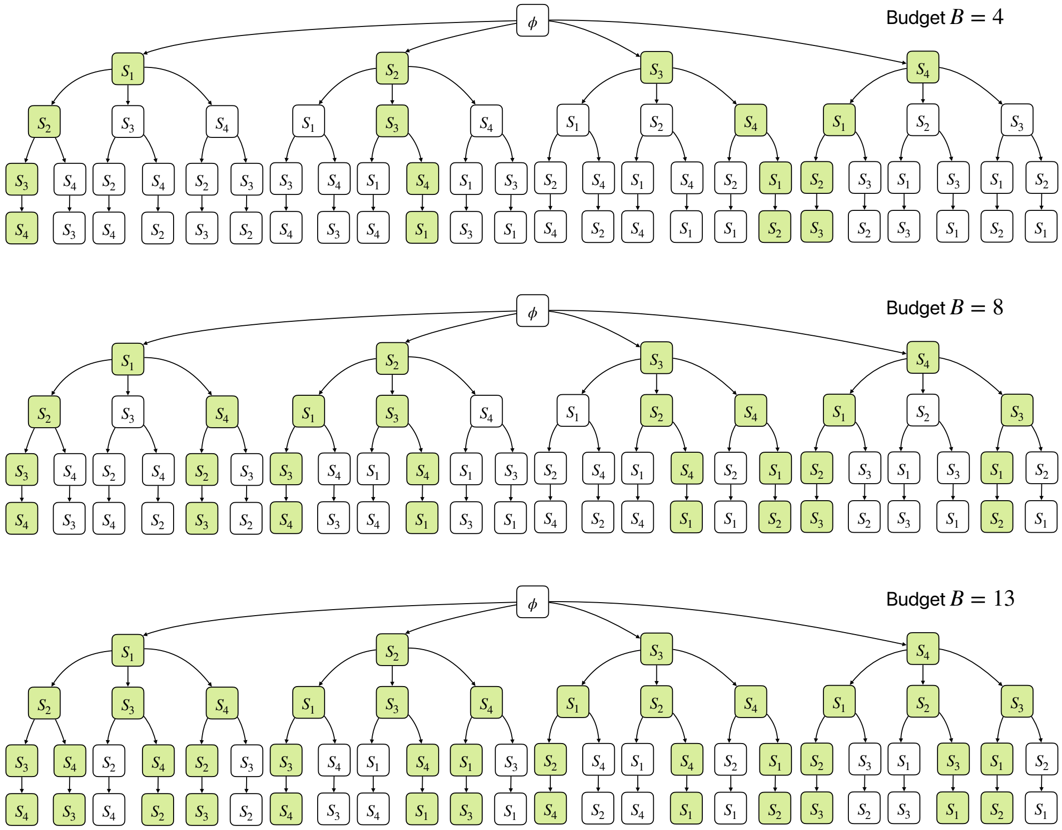

Figure 8: An illustration of the sequences selected by the iterative cyclic rotation algorithm for different budgets. We observe that for budget $B \leq L$ the algorithm selects sequences generated by rotating the entire sequence. For budget $L < B \leq L(L-1)$, the algorithm generates newer sequences by rotating the rotating subsequences starting from the second element. This continues as the budget increases and smaller subsequences are rotated.

B.2 Bipartite-matching based Sequence Selection

Before describing the bipartite selection algorithm, we discuss the scoring function for sequences with given deletion probabilities. Given a slice sequence with deletion probabilities $\mathcal{S} = (p_1, p_2, p_3, p_4)$, the scoring function computes the expected number of surviving slices after t deletions:

$$\text{score}[\mathcal{S}, t] = 4.(1-p_1-p_2-p_3-p_4)^t + 3.(1-p_1-p_2-p_3)^t + 2.(1-p_1-p_2)^t + (1-p_1)^t. \quad (23)$$

The above equation is a function of t, which needs to be set by the user. In the general case, Eq. 23 can be written as:

$$\text{score}[\mathcal{S}, t] = \sum_{i=1}^{n} i. \left(1 - \sum_{j=1}^{i} p_j\right)^t. \quad (24)$$

Next, we describe the details of the Bipartite-matching based selection (BMS) algorithm in Algorithm 2. The objective of BMS is to select a set of B diverse permutation sequences where sequences have a high score (Eq. 24). This algorithm starts by selecting L different starting elements for the sequences in Line 4. Then, BMS iteratively selects the next element within each sequence. This involves constructing a bipartite graph between the last elements of the sequences seen so far and the next set of elements. The edge weights are set to the score of the sequence that is a concatenation of the current sequence, o, and the next node, v'. The edges incident on feasible next elements (elements not seen in a permutation sequence so far) as shown in Line 12. We perform a maximum weight perfect graph matching on the graph, G, using the Hungarian algorithm (Kuhn, 1955). Based on the

Algorithm 2 BMS Sequence Selection Algorithm

1: **function** BMS(Slice count: L, Budget: B)
2: $\quad \mathcal{O} = \{\}$
3: $\quad$ **for** $l \in \{1, \ldots, L\}$ **do** // initializing the sequence set with first element
4: $\quad\quad \mathcal{O} = \mathcal{O} \cup \{l\}$
5: $\quad$ **end for**
6: $\quad$ **for** $i \in \{2, \ldots, L\}$ **do** // iterations to select the 2nd to the L-th element
7: $\quad\quad G = \{\}$ // Initialize Graph
8: $\quad\quad$ **for** $o \in \mathcal{O}$ **do** // Iterate over sequences
9: $\quad\quad\quad v = o.\mathrm{pop}()$ // Collect final element of each sequence
10: $\quad\quad\quad$ **for** $v' \in \{1, \ldots, L\}$ **do** // Iterate and add all feasible edges
11: $\quad\quad\quad\quad w = \mathrm{score}[o \cup v']$ // compute score for the sequence $o \cup v'$ (Eq. 24)
12: $\quad\quad\quad\quad$ **if** $v' \notin o$ **then** G.add_edge(v, v', w) // check for feasibility
13: $\quad\quad\quad$ **end for**
14: $\quad\quad$ **end for**
15: $\quad\quad (V, V') = \mathrm{MaximumWeightMatching}(G)$ // use Hungarian algorithm (Kuhn, 1955)
16: $\quad\quad$ **for** $o \in \mathcal{O}$ **do**
17: $\quad\quad\quad$ // select vertex associated with each sequence from perfect matching
18: $\quad\quad\quad v^* = \{v' | v' \in V' \wedge v = o.\mathrm{pop}()\}$
19: $\quad\quad\quad o = o \cup v^*$ // add to sequence
20: $\quad\quad$ **end for**
21: $\quad$ **end for**
22: $\quad \mathcal{O}_B = \{o \in \mathcal{O} | o$ is among top B sequence with highest $\mathrm{score}[o]\}$
23: $\quad$ **return** $\mathcal{O}_B$
24: **end function**

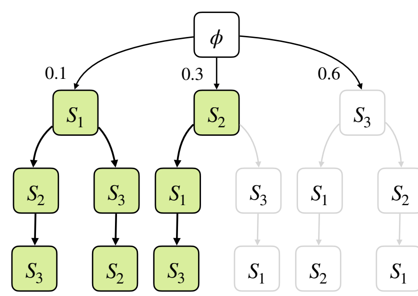

Figure 9: Illustration of conditional sampling of slice sequences. For each sequence, one slice is sampled at a time based on their deletion probabilities. We observe that conditional sampling selects sequences that has slices with relatively lower deletion probabilities towards the top.

matching, we select the next element for each permutation sequence (Line 18). BMS continues this process till all sequences $o \in \mathcal{O}$ have L elements. Based on the budget, we output B sequences from $\mathcal{O}$ that have the highest sequence scores. For maximum weight perfect matching, (Tomizawa, 1971) provides an efficient algorithm with time complexity $O(n^3)$, where n is the number of nodes. Using this algorithm, the overall time complexity of BMS is $O(L^4)$, where L is the number of slices. The following result shows that BMS generates the most diverse sequence set.

Lemma 3. *For a budget $B \leq L$, BMS returns the most diverse set of B permutations.*

We also validate the above result empirically and find that BMS produces a range of sequences with high scores (Figure 7). However, it remains an open question whether these outputs are optimal (while considering both diversity and scores).

The BMS algorithm can output up to L diverse permutations. Extending it for a budget $B > L$ is non-trivial. Therefore, in this setting, we use a conditional sampling approach to generate diverse sequences. In conditional sampling, for each sequence, we sample slices one at a time based on their deletion probabilities (low deletion probabilities have a higher chance of being sampled). If a sampled sequence already exists in the selected set, it is discarded. An illustration of the conditional

Algorithm 3 S³T Training Procedure

1: **Input**: Dataset $\mathcal{D}$, Shard Count: m, Slice Count: L, Budget: B
2: $\mathcal{F} = \{\}$ // initializing model set
3: **for** $d \in \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$ **do** // partition the dataset into shards and iterate over them
4: $\quad \mathcal{S} = \{S_1, \ldots, S_L\}$ // partition into slices, $\cup_i \mathcal{S} = d$
5: $\quad \Pi = \text{SelectTrainingSequences}(L, B)$ // using cyclic or BMS algorithm
6: $\quad$ **for** $\pi \in \Pi$ **do**
7: $\quad\quad \mathcal{S}_\pi = \pi(\mathcal{S})$ // order the slices according to the permutation π
8: $\quad\quad f = \text{SliceWiseTraining}(\mathcal{S}_\pi)$
9: $\quad\quad \mathcal{F} = \mathcal{F} \cup f$
10: $\quad$ **end for**
11: **end for**
12: **return** $\mathcal{F}$

Algorithm 4 S³T Deletion Procedure

1: **Input**: Model set $\mathcal{F}$, Deletion instance: x
2: $m' = \text{LocateShard}(x)$ // get original shard ID, m'
3: $\overline{\mathcal{F}} = \{\}$ // set of modified models
4: **for** $f \in \mathcal{F}_{m'}$ **do** // iterate over models trained on m'-th shard
5: $\quad l' = \text{LocateSlice}(x, f)$ // get original slice ID, l', of x within f
6: $\quad$ **for** $l \in \{l', \ldots, L\}$ **do**
7: $\quad\quad \text{DeactivateLayer}(f, l)$ // deactivate PEFT layers
8: $\quad$ **end for**
9: $\quad$ **if** $l' > 0$ **then** $\overline{\mathcal{F}} = \overline{\mathcal{F}} \cup f$ // ensuring all layers aren't switched off
10: **end for**
11: $\mathcal{F} = \mathcal{F} \setminus \mathcal{F}_{m'}$ // remove older models
12: $\mathcal{F} = \mathcal{F} \setminus \overline{\mathcal{F}}$ // introduce updated models
13: **return** $\mathcal{F}$

sampling approach is shown in Figure 9. We observe that the selected samples (shown in green) has slices with relatively lower deletion probabilities at the top.

B.3 S³T Training Procedure

In this section, we provide an outline for the training procedure within the S³T framework in Algorithm 3. S³T proceeds by dividing the entire dataset into m shards. Each shard is further divided into L disjoint slices. Based on the budget B, we obtain the permutation sequences to perform slice-wise training. We train each model f on a unique sequence of slice sequence, $\pi(\mathcal{S})$. We return the complete set of models, $\mathcal{F}$. During inference, the user selects the best performing model within each shard and deploys an ensemble of those models to production.

B.4 Deletion Procedure

In this section, we describe the procedure when a deletion request arrives for an instance x in Algorithm 4. We first locate the shard ID, m', where the instance x belonged and iterate over all models trained on that shard. For each of these models, we locate the slice ID, l', of x and deactivate all layers $\{l', \ldots, L\}$ (as shown in Line 7). After that, if the model f is still active, we add it to the set of modified models, $\overline{\mathcal{F}}$. S³T uses the updated set to perform inference till a new deletion request arrives. Note that the deletion process can occur entirely offline without impacting the production system, provided that the deleted instance does not influence any of the ensemble models.

Overall Workflow. So far, we have discussed the sequential training procedure and the sequence selection approach within a budget. Here, we will bring all of the components together and illustrate the functioning of S³T using a simple example. We consider a setting with $m = 3$ shards, $L = 4$ slices, and budget $B = 4$. Initially, for every shard, the available models are trained on sequences:

$$(1, 2, 3, 4), (4, 1, 2, 3), (3, 4, 1, 2), (2, 3, 4, 1)$$

Next, if a deletion request affects slice 1 for the 3rd shard. Then, the available models for the 3rd shard are: $(4), (3, 4), (2, 3, 4)$. Notice how all the PEFT layers at or below slice 1 have been switched off. In this scenario, S^3T doesn't perform any retraining but continues to function with the best available model $(2, 3, 4)$ (as it is trained on maximum slices). If the following deletion requests affect slice 2, then the available models are: $(4), (3, 4)$. S^3T continues to function with the best model $(3, 4)$. This continues until no models are available in any shard, at which point S^3T is retrained from scratch.

Table 1: We report the hyperparameters used in our experiments for fine-tuning S^{3}T across different benchmarks and Transformer model variants.

Dataset	Model	# Slices (L)	# Layers/ Slice	LoRA Rank (r)	LoRA (α)	Learning Rate	Epochs
GLUE Benchmark							
SST-2	RoBERTa$_{\text{LARGE}}$	7	3	16	32	10^{-5}	10
COLA	RoBERTa$_{\text{LARGE}}$	7	2	8	16	4.10^{-4}	30
STS-B	RoBERTa$_{\text{LARGE}}$	7	3	16	32	10^{-4}	30
QNLI	RoBERTa$_{\text{LARGE}}$	7	3	8	16	5.10^{-5}	30
QQP	RoBERTa$_{\text{LARGE}}$	7	3	16	32	5.10^{-5}	30
MRPC	RoBERTa$_{\text{LARGE}}$	7	3	16	32	5.10^{-5}	30
MNLI	RoBERTa$_{\text{LARGE}}$	7	3	16	32	10^{-4}	30
SuperGLUE Benchmark							
RTE	RoBERTa$_{\text{LARGE}}$	8	4	16	32	10^{-5}	30
WIC	RoBERTa$_{\text{LARGE}}$	12	2	16	32	10^{-4}	30
CB	RoBERTa$_{\text{LARGE}}$	12	2	16	32	10^{-4}	30
COPA	RoBERTa$_{\text{LARGE}}$	12	2	32	64	10^{-5}	30
BoolQ	RoBERTa$_{\text{LARGE}}$	7	3	16	32	10^{-4}	30
MultiRC	RoBERTa$_{\text{LARGE}}$	7	3	16	32	10^{-4}	30
Vision Benchmark							
CIFAR-10	ViT$_{\text{BASE}}$	6	2	16	32	2.10^{-3}	15
CIFAR-100	ViT$_{\text{BASE}}$	6	2	16	32	2.10^{-3}	15
TinyImagenet	ViT$_{\text{LARGE}}$	6	4	16	32	2.10^{-3}	15
Instruction Tuning							
Alpaca	Llama2-7B	4	8	32	64	2.10^{-5}	3
Alpaca	Llama2-13B	4	10	32	64	2.10^{-5}	3
Alpaca	Llama3-8B	4	8	32	64	2.10^{-5}	3

C Experiments

In this section, we describe our experimental setup and present additional analysis experiments to evaluate the functioning of S^{3}T.

C.1 Experimental Setup

We perform all experiments using PyTorch (Paszke et al., 2019) and Huggingface (Wolf et al., 2019) framework. Our experiments were run on NVIDIA A6000 GPUs. In Table 1, we report the common set of hyperparameters for S^{3}T fine-tuning experiments. All hyperparameters were set using a grid search with the Weights & Biases framework. We use an AdamW optimizer with the corresponding learning rates for each dataset (reported in Table 1). During fine-tuning of the models, we perform full-precision training for all settings except instruction tuning where we use 8-bit training.

C.2 Design Choices

We discuss the rationale behind the design choices for the proposed slice-wise training approach. First, we use top-to-bottom fine-tuning because we found the top layers were easier to train at the start and a bottom-up approach didn't converge. Note that our training process does not need to proceed in a layer-wise fashion, we can even train multiple LoRA layers per slice. Second, we train layers in a cumulative manner where the i-th layer is trained using slices $\{1, \ldots, i\}$. We found that this way of training helps in better convergence compared to the setup where we train every layer using a different slice. Third, we analyze the cost of slice-wise training and find it to be comparable to full training. Considering that training cost is $c = O(nl)$, where n is the dataset size and l is the number of trainable layers (empirical evidence in Appendix C.3). For slice-wise training, we observe that $c = \sum_{i=1}^{k} (\frac{in}{k})(\frac{l}{k}) = O\left(\frac{nl}{2}\right)$, which is of the same order as full training.

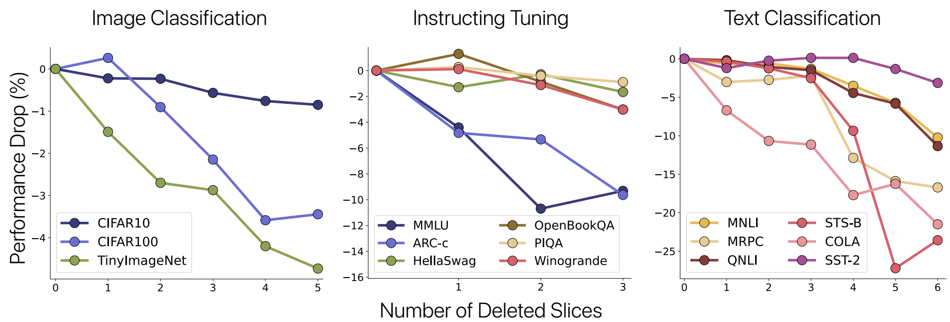

Figure 10: Relative performance drop with an increasing number of deleted slices. We observe a relatively small performance drop for image classification and instruction tuning tasks, while noticing a considerable drop for few of the text classification datasets.

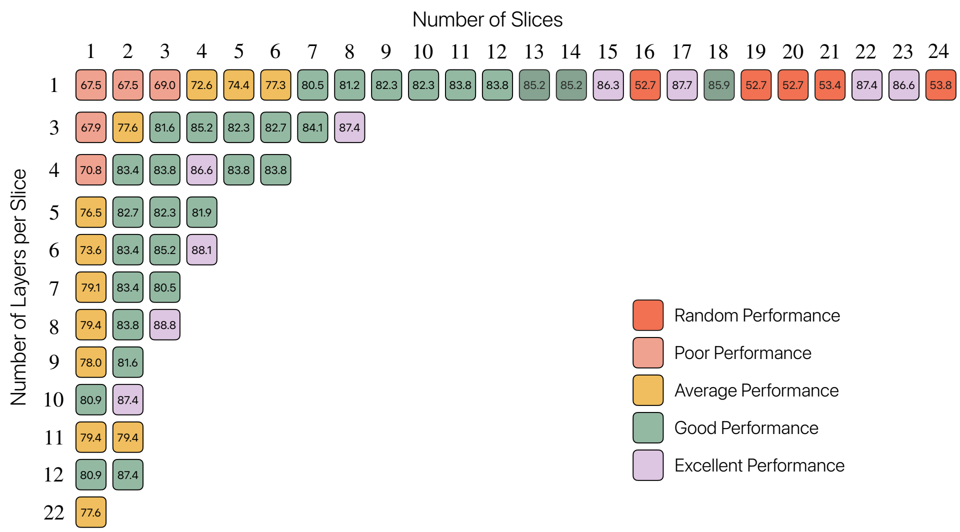

Figure 11: Performance impact of S^3T when allocated a different number of layers per slice: We observe that simply increasing the number of slices can lead to a significant performance drop. There is an optimal tradeoff between the total number of slices and the number of layers per slice.

C.3 Additional Experiments

In this section, we report analysis experiments to evaluate the S^3T's deletion capabilities and training.

Performance Degradation with Slice Deletion. In Figure 10, we report the relative performance drop with an increasing number of slices affected by deletion requests. For vision datasets in Figure 10 (left), we only observe a small drop in performance (<5%). For instruction tuning in Figure 10 (middle), we observe a negligible performance drop for most datasets except MMLU & ARC, which are more challenging open-ended QA benchmarks. For text classification in Figure 10 (right), we observe an increased drop in performance for a few of the datasets. We hypothesize that this occurs because the RoBERTa$_{\text{LARGE}}$ (used in text classification tasks) is a relatively weaker pre-trained model compared to Llama and ViT, which may explain this behavior. We also observe that the performance can vary based on the task and overall it is dependent on both the task and the model being fine-tuned. For unlearning, a lower performance drop is better as it suggests that we can continue using the same model without retraining for a longer time.

Layer Allocation per Slice. This experiment aims to answer the question: how many slices can we pack into a model while still achieving good performance? We conduct an experiment where we allocate different numbers of PEFT layers per slice and also vary the number of slices. We report the results on RTE dataset using RoBERTa$_{\text{LARGE}}$ in Figure 11. We observe that the performance is poor

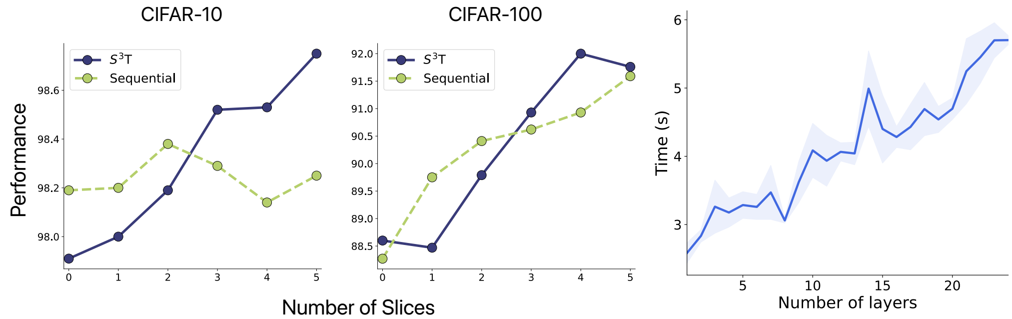

Figure 13: (*Left*) Comparison of sequential training with S^{3}T using a PEFT model. We observe that S^{3}T's performance gradually improves as it is trained on a larger number of slices, ultimately achieving similar performance to sequential training while being more efficient.

when the number of layers per slice is low. For example, when the number of layers per slice = 1, the performance improves as we increase the number of slices but again drops when the slice count is too high. This shows that it may not be feasible to train the model using a large number of slices without a drop in performance. Overall, the performance variation depends on the underlying task and model, so the developer should select these hyperparameters based on their performance requirements.

Sequence-based Training. In this experiment, we compare S^{3}T's performance with sequential training. Sequential training involves training the entire PEFT model on a sequence of slices. This was used in (Kumar et al., 2023) to improve the retraining efficiency of SISA. In Figure 13, we report the performance of S^{3}T and sequential training using ViT$_{\text{BASE}}$ on CIFAR-10 and CIFAR-100 datasets. We observe that the performance of S^{3}T gradually increases as it is trained on more slices. Overall, we observe that S^{3}T achieves quite similar or outperforms sequential training. It is important to note that S^{3}T trains a significantly smaller number of parameters (L times reduction compared to sequential training) but still achieves competitive performance.

BMS vs. Cyclic rotation with Deletion prior. In this setting, we compare the BMS selection algorithm with a variant of cyclic rotation when the prior deletion probabilities are available. We perform cyclic rotation by first sorting the slices based on their deletion probabilities. For example, we sort the sequence with deletion probabilities: (S_1: 0.5, S_2: 0.4, S_3: 0.1) as: (S_3, S_2, S_1). Then, for a budget $B = 3$, the stored sequences are: (S_3, S_2, S_1), (S_1, S_3, S_2), (S_2, S_1, S_3). In this variant, the slices most likely to be deleted are not at the top of any sequences. We follow the experimental setup and sample deletion priors using a Dirichlet distribution (over 10 runs). In Figure 12, we report the total sequence scores (Eq. 24) obtained by BMS and sorted cyclic rotation for a budget, $B = L$. We observe that BMS consistently outperforms sorted cyclic rotation for all budgets. Please note that the average edit distance achieved by both methods are the same as cyclic rotations are guaranteed to produce the maximum diversity for budgets: $B \leq L$.

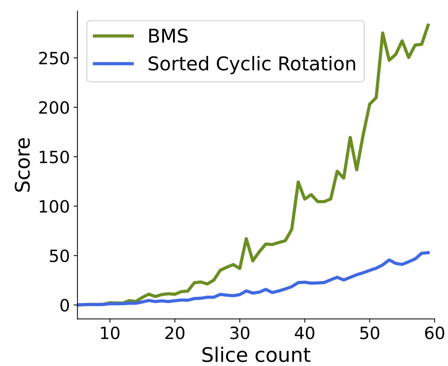

Figure 12: We compare the scores (Eq. 24) of the generated sequences by BMS and sorted cyclic rotation. We observe BMS consistently outperforms cyclic rotation.

Training Time. In this experiment, we evaluate the training time with a varying number of PEFT layers. In Figure 13 (right), we report the average training time over a constant number of steps using RoBERTa$_{\text{LARGE}}$ model. We observe that training time linearly increases as an increased number of LoRA layers are trained. This shows the effectiveness of our proposed S^{3}T framework, which only trains a small number of PEFT layers at each training stage.

Storage Costs. In this experiment, we evaluate how the storage cost of S^{3}T grows with an increasing budget and its effect on the overall deletion rate. In Table 2, we report the storage costs and deletion rate of training ViT$_{\text{LARGE}}$ model with $m = 5$ shards and $L = 6$ slices per shard. We observe that the

Table 2: Storage cost of S^{3}T under different training budgets. We show how the deletion rate increases with an increased storage cost. The overall storage cost of PEFT layers is minimal and is equivalent to storing an additional model.

Budget	Model (GB)	PEFT (GB)	Del. Rate
$B = 1$	1.2	0.21	66.4
$B = 2$	1.2	0.42	86.9
$B = 3$	1.2	0.63	100.8
$B = 4$	1.2	0.84	107.8
$B = 5$	1.2	1.05	110.1
$B = 6$	1.2	1.26	124.1

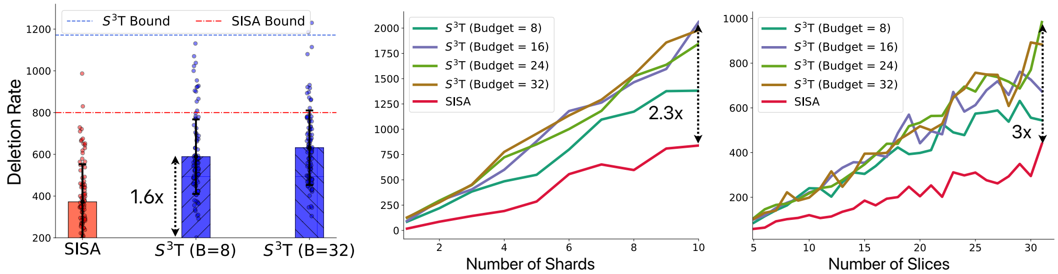

Figure 14: We report the deletion rate of S^{3}T and compare it with baselines. (*Left*) We compare the deletion rates of S^{3}T under varying budgets with SISA and observe significant gains. (*Center*) We report the deletion rates with an increasing number of shards and (*Right*) an increasing number of slices. In both scenarios, we observe that S^{3}T's deletion rate grows significantly faster than SISA.

Model	MMLU	HellaSwag	PIQA	Winogrande	ARC-c	ARC-e	TruthfulQA	OBQA
Llama2-7B	41.3$_{(0.4)}$	76.0$_{(0.4)}$	79.1$_{(1.0)}$	69.1$_{(1.3)}$	46.3$_{(1.5)}$	74.6$_{(0.9)}$	39.0$_{(1.4)}$	44.2$_{(2.2)}$
Llama2-7B (FT)	43.3$_{(0.4)}$	77.4$_{(0.4)}$	**79.9**$_{(0.9)}$	69.6$_{(1.3)}$	50.4$_{(1.5)}$	**74.7**$_{(0.9)}$	**46.3**$_{(1.6)}$	**46.8**$_{(2.2)}$
Llama2-7B (S^{3}T)	**43.6**$_{(0.4)}$	**77.7**$_{(0.4)}$	<u>79.7</u>$_{(0.9)}$	**70.5**$_{(1.3)}$	**51.4**$_{(1.5)}$	<u>74.1</u>$_{(0.9)}$	43.0$_{(1.5)}$	<u>46.4</u>$_{(2.2)}$
Llama2-13B	50.5$_{(0.4)}$	79.4$_{(0.4)}$	80.5$_{(0.9)}$	72.2$_{(1.3)}$	49.0$_{(1.5)}$	77.5$_{(0.9)}$	36.9$_{(1.4)}$	45.2$_{(2.2)}$
Llama2-13B (FT)	**51.8**$_{(0.4)}$	**81.3**$_{(0.4)}$	**82.0**$_{(0.9)}$	72.2$_{(1.3)}$	**55.0**$_{(1.5)}$	79.3$_{(0.8)}$	38.8$_{(1.5)}$	**46.6**$_{(2.2)}$
Llama2-13B (S^{3}T)	<u>51.6</u>$_{(0.4)}$	<u>80.9</u>$_{(0.4)}$	<u>81.8</u>$_{(0.9)}$	**72.8**$_{(1.3)}$	54.5$_{(1.5)}$	**80.6**$_{(0.8)}$	**39.6**$_{(1.5)}$	<u>46.2</u>$_{(2.2)}$
Llama3-8B	62.2$_{(0.4)}$	79.1$_{(0.4)}$	80.6$_{(0.9)}$	73.6$_{(1.2)}$	53.2$_{(1.5)}$	77.6$_{(0.9)}$	43.9$_{(1.4)}$	45.0$_{(2.2)}$
Llama3-8B (FT)	59.2$_{(0.4)}$	**80.8**$_{(0.4)}$	80.9$_{(0.9)}$	72.2$_{(1.3)}$	**59.0**$_{(1.4)}$	79.6$_{(0.8)}$	46.6$_{(1.6)}$	46.8$_{(2.2)}$
Llama3-8B (S^{3}T)	**59.6**$_{(0.4)}$	<u>80.2</u>$_{(0.4)}$	**82.4**$_{(0.9)}$	**73.8**$_{(1.2)}$	57.0$_{(1.5)}$	**80.5**$_{(0.8)}$	**48.5**$_{(1.6)}$	**47.0**$_{(2.2)}$

Table 3: Performance comparison of LLMs before and after instruction tuning using Alpaca dataset. We report the performance of Llama2-7B, Llama2-13B, and Llama3-8B models for the following settings: pre-trained model, full training (FT), and slice-wise training (S^{3}T). We observe S^{3}T achieves comparable performance to FT, even outperforming FT's performance in several datasets.

storage cost of PEFT layers (with LoRA rank=16) is considerably less compared to the full model size. As the budget and storage cost increases there is an improvement in the deletion rate. However, the rate of improvement of the deletion rate slows down with an increased budget, indicating there is a lesser return on increasing the budget.

Deletion Ablations. We evaluate the deletion capabilities of S^{3}T and compare with the theoretical bounds. In Figure 14 (left), we report the deletion rate of S^{3}T and SISA for the setup (with $m = 5$ shards, $L = 32$ slices). We observe that with a small budget $B = 8$, S^{3}T can achieve 1.6x gains in the number of deletion requests handled. We also plot the theoretical bounds derived in Section 3.4 and show that they hold in practice.

In Figure 14 (center & right), we report the model performance with varying shard and slice counts. As expected, we observe that both increasing the shards and slicing the data more helps in improving the deletion performance. However, the rate of growth in deletion rate is more significant for S^{3}T resulting in up to 2.3x and 3x gains over SISA for the same number of shards and slices respectively.

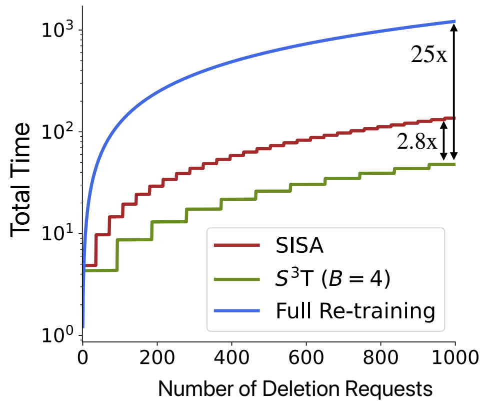

Figure 15: We compare the deletion time as the number of deletion requests increases. S^3T requires the lowest deletion time compared to SISA and full re-training.

Deletion Time. Unlearning in S^3T is cost-effective; it primarily involves selecting the best checkpoint and swapping with the current one. The main cost is incurred when a large number of deletion requests necessitate re-training of the system from scratch. In Figure 15, we report the total deletion time as the number of deletion requests increases. In this experiment, we compare the deletion time of S^3T with SISA and full re-training on CIFAR10 dataset. Full re-training retrains the model after each deletion request. For SISA and S^3T, the step jumps indicate that the system requires retraining. Overall, S^3T achieves the lowest deletion time. S^3T reduces the total deletion time (after 1000 requests) by 2.8x compared to SISA and 25x compared to full retraining. This experiment shows the efficacy of S^3T in reducing the overall deletion time during exact unlearning.

D LIMITATIONS

In this paper, we present a novel exact unlearning framework, S^3T. S^3T improves the deletion rate of existing unlearning systems at the cost of additional offline training. The training process can be time-consuming if we are fine-tuning larger models and have a higher performance requirement (thereby high budget B). However, this is an inherent tradeoff between offline training and losing revenue due to re-training costs. The developer should adjust their budget according to the tradeoff in their specific application.

E BROADER IMPACT

We present a scalable approach to perform exact unlearning in production systems. We hope that this framework will be adopted by organizations and enable a cost-effective way to ensure the privacy requests of various users. In general, unlearning systems are susceptible to attacks where the adversary may design deletion requests in a way to modify the model behaviour according to their needs. Therefore, it is important to ensure that the deletion requests executed on the unlearning system are not malicious. However, this is a limitation of unlearning systems in general and not specific to our proposed framework, S^3T. Future works can focus on the identification of malicious unlearning requests.