Duplicate-and-Share: A Novel Approach to Efficient Vision Transformer Unlearning

Anonymous Author(s)

ABSTRACT

Machine unlearning, which aims to erase the traces of specific training data points from a pre-trained model, has become a crucial task, especially with increasing privacy concerns surrounding large pre-trained models. While substantial research has explored various approaches, fewer studies have focused on the model architectural perspective. In this paper, we introduce the Duplicate-and-Share (DASH) strategy, a novel model architecture effective in machine unlearning. Integrating DASH into different unlearning algorithms consistently enhances performance, achieving state-of-the-art results on the recently proposed unlearning datasets, MUFAC and MUCAC. To our knowledge, this is the first work to modify the Vision Transformer's architecture to enhance machine unlearning performance.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Machine Unlearning, Machine Learning, Model Architecture Engineering, Vision Transformer

ACM Reference Format:
Anonymous Author(s). 2024. Duplicate-and-Share: A Novel Approach to Efficient Vision Transformer Unlearning. In *Proceedings of 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Machine unlearning (MUL) is an arising field in machine learning that focuses on removing the learned knowledge associated with specific training data points from a pre-trained model. The demand for MUL has surged recently, driven by the increasing frequency of privacy-related concerns in large pre-trained models [3, 13, 24]. For example, individuals now have the authority to request the deletion of their personal data from pre-trained models, as established by the Right to Be Forgotten [22, 26].

One straightforward solution to MUL is to retrain the model from scratch, excluding the data points to be forgotten at each deletion request. However, this solution has two major limitations. Firstly, it is computationally prohibitive to train a model for every single request, especially considering the vast volume of recent training data. Additionally, if the data points to be forgotten are substantial, retraining could potentially lead to a model with lower overall performance. To address these limitations, the task of MUL has emerged, aiming to forget specific learned knowledge from a pre-trained model rather than retraining from scratch. MUL can be interpreted as eliminating the overfitted knowledge while preserving the general understanding of the data points to be forgotten.

Despite active research in this field [3, 11, 13, 24], only a limited number of studies have focused on the *model architectural* perspective [1, 28]. In this paper, we investigate effective model architectures for MUL with the aim of addressing the following underexplored research question: "Can we improve MUL performance by modifying the model architecture? If so, how and why should it work?"

We tackle this research question in our paper through extensive ablation studies and ultimately introduce **D**uplicate-**a**nd-**Sh**are (DASH), a simple yet effective Transformer-based model architecture as a viable solution. The core intuition behind DASH is to target the upper layers of ViT models that are known to be responsible for overfitting [10] (further verified in Section 5.1) and apply an architectural regularization that allows rapid removal of overfitted knowledge while maintaining the overall model performance efficiently (refer to Section 3.1 for details). We empirically demonstrate that the DASH architecture consistently enhances MUL performance, achieving state-of-the-art results on both datasets, MUFAC and MUCAC, with both models, ViT-base and ViT-large. Our findings suggest that exploring the model architectural perspective is a promising research direction for advancing MUL. We believe our work represents an important step toward promoting further exploration in this area.

We summarize our main contributions below:

- We introduce Duplicate-and-Share (DASH), a novel model architectural approach to machine unlearning. DASH is an orthogonal approach that can be seamlessly integrated with the majority of existing unlearning algorithms.
- DASH sets new state-of-the-art records on both datasets (MUFAC and MUCAC) with both models (ViT-Base and ViT-Large), verifying its general effectiveness.
- To our knowledge, this is the first work to directly modify the ViT model architecture to faciliate machine unlearning.

2 RELATED WORK

2.1 Vision Transformer

Originally developed for advancements in natural language processing with implementations like BERT [4] and RoBERTa [15] making significant impacts, transformers [25] have now carved a niche in computer vision, marking a shift in model architecture preferences. At the forefront of this transition is the Vision Transformer (ViT) [5], which underpins the latest breakthroughs across a

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

spectrum of computer vision tasks, including but not limited to image classification [27] and captioning [17]. The unique approach of ViT, diverging from conventional methodologies, revolves around leveraging a purely transformer-based architecture. It segments images into equally sized patches or tokens, incorporates learnable position embeddings for these tokens, and subjects them to a series of self-attention mechanisms within transformers. This technique is adept at parallelizing the learning process for connections among segmented parts of the image, thereby enhancing the model's interpretative capabilities. With ViT models, pre-trained on extensive datasets comprising millions of image-context pairs, establishing themselves as pivotal in various computer vision domains, the importance of assessing machine unlearning methods on such ViT-based models is underscored.

2.2 Machine Unlearning

Formal definition and metrics For a fixed train dataset D_{train}, forget set D_{forget}, retain set D_{retain} $(= D_{train} \backslash D_{forget})$, and a deep learning model $\{f_{\theta_0}(x) : x \longrightarrow y \mid (x, y) \in D\}$ where its parameters θ_0 are initially pre-trained on D_{train}, an unlearning algorithm U takes $(f, \theta_0, D_{forget}, D_{retain})$ as input and produces a model:

$$\theta^* = U(f, \theta_0, D_{forget}, D_{retain}) \tag{1}$$

that has "forgotten" D_{forget}. Typical machine unlearning algorithms are evaluated using a two-metric framework: (1) utility and (2) forgetting performance.

Utility score A meaningful unlearning algorithm should not significantly degrade the general performance of the given model. Therefore, utility, measured by the accuracy on a separate test set D_{test}, is crucial for evaluating the overall effectiveness of the unlearning process. Utility is measured as follows:

$$Utility : \frac{\sum_{i=1}^{N_{test}} I_{[\hat{y}_i = y_i]}}{N_{test}}$$

where $\hat{y}_i$ is the model's prediction, y_i is the true label, and I indicates the indicator function.

Forgetting performance A common approach to assessing the forgetting performance of an unlearning algorithm is through the Membership Inference Attack (MIA) framework [3, 11, 24]. The core idea of MIA involves training a binary classifier, typically a logistic regression model, to distinguish loss values from $f_{\theta^*}(D_{forget})$ and $f_{\theta^*}(D_{unseen})$. Here, D_{unseen} represents another held-out set of approximately the same size as D_{forget}, which was not used in training. If the classifier's accuracy converges to 50%, it indicates that the classifier performs no better than random choice, suggesting a high degree of similarity between $f_{\theta^*}(D_{forget})$ and $f_{\theta^*}(D_{unseen})$. This implies that f_{θ^*} behaves similarly on the forget set and the unseen set, demonstrating effective forgetting of D_{forget}.

In this paper, we adopt the MIA framework for measuring the forgetting quality. We first collect the loss values from D_{forget} and D_{unseen} using the unlearned model θ^*:

$$L(f_{\theta^*}(x), y), \ (\{x, y\} \in D_{forget} \cup D_{unseen})$$

where L is the loss function. Then, a logistic regression classifier $\psi(\cdot)$ is trained to distinguish whether the input x is from the forget or unseen dataset based on the loss value:

$$\psi(L(f_{\theta^*}(x), y)) = \begin{cases} 1 & \text{if } \{x, y\} \in D_{forget} \\ 0 & \text{if } \{x, y\} \in D_{unseen} \end{cases}$$

An ideal unlearning algorithm will result in $\psi(\cdot)$'s accuracy of 50%, indicating indistinguishability between forget and unseen samples. Therefore, the forgetting score is calculated as $|M - 0.5|$, where M represents the accuracy of the MIA classifier, with a smaller value indicating better performance.

Normalized Machine Unlearning Score (NoMUS) Introduced by Choi and Na, the NoMUS score combines utility and forgetting scores into a single value within the range [0,1], with higher values indicating better performance. NoMUS is computed as follows:

$$NoMUS = Util \times \lambda + (1 - abs(M - 0.5) \times 2) \times (1 - \lambda)$$

We set the value of λ to 0.5, giving equal importance to both metrics.

2.3 General Approaches to Machine Unlearning

We provide a summary of widely recognized machine unlearning methods that have gained prevalence. These methods can be broadly categorized into four groups: (1) fine-tuning-based, (2) gradient manipulation-based, (3) knowledge manipulation-based, and (4) re-initialization-based approaches.

Fine-tuning The fine-tuning approaches to machine unlearning focus on adapting the model's training process to selectively forget specific data. This approach includes methods like Fine-Tuning [9] and Catastrophically Forgetting-k (CF-k) [8], where the former refines the entire model using only the data intended to be retained and employs a slightly higher learning rate to aid in forgetting the data to be unlearned. The latter, CF-k, leverages the model's inherent tendency for catastrophic forgetting [7] by concentrating the fine-tuning process on the model's last k layers. This strategic focus allows for efficient unlearning of data with minimal impact on the model's overall utility and performance, requiring fewer training epochs to achieve the desired outcomes. Both methods exemplify the principle that targeted adjustments in the training process can significantly enhance a model's ability to forget unwanted data, offering a nuanced approach to machine unlearning that preserves the integrity of the retained information.

Gradient manipulation Gradient manipulation techniques introduce a direct intervention in the model's learning dynamics to erase specific data points. Within this domain, Advanced Negative Gradient (AdvNegGrad) [3] emerges as a pivotal method that extends Negative Gradient (NegGrad) [9] method. NegGrad utilizes gradient ascent on the forget set to increase loss and achieve machine unlearning, whereas AdvNegGrad combines the joint loss of fine-tuning with NegGrad within the same training batches. This approach underscores a strategic manipulation of the model's learning process, aiming to directly counteract the retention of undesired information through calculated adjustments in the model's gradient and loss landscapes.

Knowledge manipulation Knowledge manipulation-based approach uses knowledge distillation techniques [2, 19, 23] to introduce changes on the model parameters and then apply fine-tuning to restore desired performance. In this domain, Unlearning by Selective Impair and Repair (UNSIR) and SCalable Remembering and

Unlearning unBound (SCRUB) methods present innovative strategies. UNSIR, as proposed by [24], initially aimed to help models forget specific classes by introducing disruptive noise that negatively affects the model's weights during the learning phase. This process of corruption is followed by fine-tuning with a retain set to rectify the changes. However, its application has been extended [3] to target the forgetting of individual data points, achieved by creating synthesized noise that maximizes the difference from the data intended to be forgotten, followed by fine-tuning to correct the model. On a related front, SCRUB employs a stochastic initialization model that acts as a student model to remove knowledge of the forget set, utilizing a "bad teacher" concept to mitigate the forget set's impact. This method, introduced by [13], strategically distances the student model from the teacher concerning the forget set while ensuring closeness for the retain set, addressing the potential negative effects on model performance that can arise from maximizing the distance between the student and teacher models for the forget set.

Re-initialization Re-initialization approaches re-calibrate specific models to distinguish between the information to be retained and that which is to be forgotten. These strategies are exemplified by the Attack-and-Reset (ARU) [11] method. ARU identifies and reinitializes model parameters that contribute to overfitting on the forget set. By measuring gradient differences and targeting specific parameters for resetting, followed by subsequent fine-tuning, ARU effectively erases the memory of the forget set from the model. This approach showcases the effectiveness of pinpoint parameter adjustments in removing undesired information while maintaining the integrity of the learned model.

In this paper, we implement the previously mentioned baseline algorithms on Vision Transformer (ViT)-based models and compare their performance with and without DASH, utilizing the MUFAC and MUCAC datasets.

2.4 Model Architectural Approach to MUL

To the best of our knowledge, there is no prominent study that has specifically focused on the model architecture perspective to enhance MUL performance. [1] is, to some extent, the closest example of a model architectural approach to enhance MUL. Bourtoule et al. [1] present SISA (Sharded, Isolated, Sliced, Aggregated), which introduces a strategic sharding of the training data and saving multiple checkpoints during pre-training to selectively use one (or several) of the checkpoints for fast machine unlearning. The main difference with our work is that they do not physically change the model architecture; instead, they save multiple model checkpoints. DASH, on the other hand, directly modifies the model architecture to improve MUL.

2.5 Instance-based Machine Unlearning

Most previous studies on MUL have employed traditional computer vision datasets such as MNIST [14], CIFAR-10 [12], and SVHN [18]. Earlier research primarily concentrated on unlearning specific classes, such as eliminating classes of digit 1 in MNIST. However, recent observations indicate that this approach may not be well-suited for real-world applications [3]. In practical scenarios, there is often a need to forget specific instances (individuals) with potentially diverse labels, rather than a unified single class. Recognizing this, Choi and Na [3] introduced two new machine unlearning benchmark datasets: MUFAC and MUCAC. This paper specifically focuses on instance-based machine unlearning, as it better aligns with real-world scenarios.

MUFAC (Machine Unlearning for Facial Age Classifier) The multi-class age classification dataset, MUFAC, was introduced by Choi and Na [3]. This dataset consists of Asian facial images with annotated labels, categorizing individuals into eight age groups. In contrast to previous machine unlearning tasks that targeted forgetting specific class(es), MUFAC's focus is on unlearning a group of individuals with diverse class labels. The primary goal, similar to previous machine unlearning tasks, is to achieve a decent trade-off between test accuracy and the quality of forgetting.

MUCAC (Machine Unlearning for Celebrity Attribute Classifier) Choi and Na introduced another MUL dataset, named MUCAC, derived from CelebA [16]. This dataset focuses on a multi-label facial classification task with labels covering gender (male/female), age (old/young), and expression (smiling/unsmiling). Comprehensive statistical information about MUFAC and MUCAC is available in Section 4.1.

3 DUPLICATE-AND-SHARE (DASH)

3.1 Motivation

Machine unlearning can be interpreted as the process of erasing overfitted knowledge while retaining a general understanding of the forget images. We focus on the well-known observation that the upper layers are more responsible for overfitting in transformer architectures, given their focus on detailed and intricate features of the input image [10] (further verified in Section 5.1). The core intuition behind Duplicate-and-Share (DASH) is that an efficient model architecture that specifically targets the upper layers could accelerate the removal of overfitted knowledge from those layers. Specifically, we claim and empirically verify that duplicating the upper *n* layers and sharing their parameters, as depicted in Figure 1, is an effective architecture for unlearning.

Intuitively, finetuning the model on the retain set with this augmented structure leads to quicker unlearning, as this augmentation induces an additional regularization effect. In the augmented architecture, each layer now contributes to two layers instead of one in the original design. Consequently, each layer must handle the tasks of two layers, preventing it from overfitting to a singular layer's task. Thus, this regularization automatically promotes more general learning, forcing the upper layers to swiftly remove overfitting. Furthermore, DASH has the advantage of having a greater number of layers, which prevents the introduced regularization effect from significantly degrading performance. This is because a higher number of layers generally leads to improved performance. As a result, DASH enables the efficient removal of overfittedness in the upper layers while simultaneously ensuring similar or even improved utility performance, making it well-suited for machine unlearning.

3.2 Methodology

As explained in Section 3.1, the central idea of DASH is to duplicate the upper n layers and apply parameter sharing, as illustrated in Figure 1. That is, the upper layers now run twice in the forward pass. In the backward pass, the gradients of the upper n layers are calculated twice (once in the upper shared layer and once in the bottom shared layer) and subsequently updated. This methodology can be easily implemented by modifying the forward pass, as illustrated in Figure 2. It is important to note that the backward pass update is automatically done by default in PyTorch according to our intention, requiring no additional modifications. The overall algorithm of DASH can be represented as:

$$\theta^*_{dash} = U(f_{dash}, \theta_0, D_{forget}, D_{retain})$$

where f in equation (1) is modified to f_{dash}, the modified forward pass explained above. One important note is that after the unlearning process is completed, the model is reverted back to its original architecture f. This ensures that the final model maintains the same architecture as the baseline model, enabling a fair comparison.

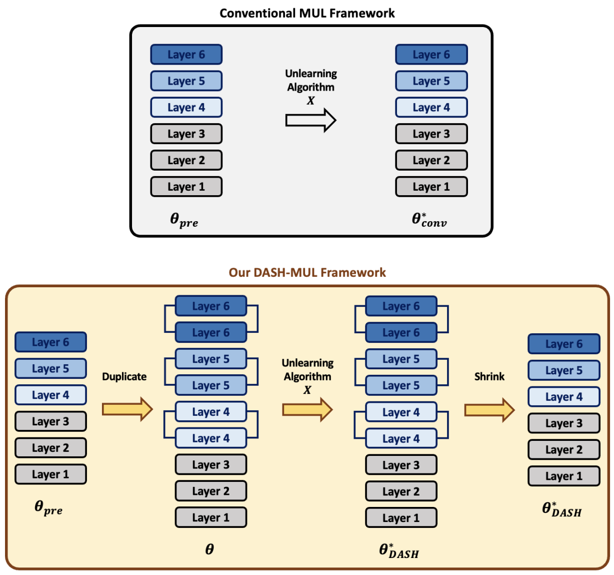
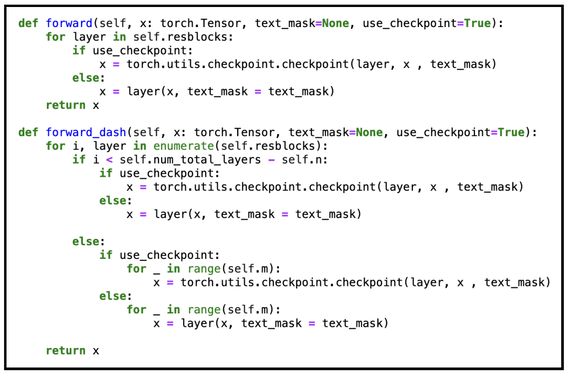


Figure 1: The overall framework of DASH. The example uses a 6-layer transformer model with $(n, m) = (3, 2)$ for display purposes. The main difference between DASH and conventional unlearning frameworks is the modified model architecture (i.e., the forward pass). It is important to note that the total number of parameters remains unchanged, with a slight increase in runtime attributed to the augmented number of layers (see Table 5.5). At the end of the unlearning process, we shrink the model back to its original architecture to ensure fair comparison with conventional approaches.

3.3 Hyperparameters of DASH

DASH introduces two additional hyperparameters. (1) n: the number of layers to duplicate and (2) m: how many duplicates to make for each layer. We find that various combinations of n and m (e.g.,



```
def forward(self, x: torch.Tensor, text_mask=None, use_checkpoint=True):
    for layer in self.resblocks:
        if use_checkpoint:
            x = torch.utils.checkpoint.checkpoint(layer, x , text_mask)
        else:
            x = layer(x, text_mask = text_mask)
    return x

def forward_dash(self, x: torch.Tensor, text_mask=None, use_checkpoint=True):
    for i, layer in enumerate(self.resblocks):
        if i < self.num_total_layers - self.n:
            if use_checkpoint:
                x = torch.utils.checkpoint.checkpoint(layer, x , text_mask)
            else:
                x = layer(x, text_mask = text_mask)

        else:
            if use_checkpoint:
                for _ in range(self.m):
                    x = torch.utils.checkpoint.checkpoint(layer, x , text_mask)
            else:
                for _ in range(self.m):
                    x = layer(x, text_mask = text_mask)

    return x
```

Figure 2: Implementing the forward pass of DASH. No modification is needed for the backward pass. self.n and self.m represent the hyperparameters n and m, respectively.

$(n, m) = (3, 2), (4, 2), \ldots, (6, 3)$ for ViT-Base) are effective in unlearning. Nevertheless, since we do not want to achieve deceptive benefits arising from an extensive hyperparameter search for our method, we set the default values of n to 6 and 12 for ViT-B-16 and ViT-L-14, respectively, which is half of the total number of layers, and m as 2. We empirically found these values to be generally effective.

4 EXPERIMENTS

4.1 Datasets

As detailed in Section 2.5, our work focuses on instance-based unlearning rather than unlearning entire classes, as it aligns better with real-world scenarios. Accordingly, we employ the recently introduced benchmark datasets, MUFAC and MUCAC, proposed by Choi and Na [3]. A summary of the statistics for these datasets is provided in Table 1. All images in MUFAC and MUCAC have a resolution of 128×128, with a focus on the facial region.

	MUFAC	MUCAC
Train dataset	10,025	25,846
Test dataset	1,539	2,053
Forget dataset	1,500	10,135
Retain dataset	8,525	15,711
Unseen dataset	1,504	2,001

Table 1: Overall Statistics of MUFAC and MUCAC benchmark datasets. More details can be found in [3].

4.2 Experimental Settings

We use two most widely utilized ViT models, namely ViT-Base and ViT-Large. For pre-training the models, we use the pre-trained models (ViT-B-16 and ViT-L-14) provided by Radford et al. [21] as initial checkpoints. The default hyperparameter configurations from the official repository [21] are adopted, including the use of the adamW optimizer, a learning rate of 1e-5, a batch size of 64, and a patch size of 16 for ViT-B-16 and 14 for ViT-L-14. To reduce stochasticity, we use 5 random seeds uniformly across all algorithms and present the results as averages along with standard deviations.

It is important to note that DASH requires more time to complete unlearning compared to the baseline architecture due to the increased number of layers. Thus, we ensure a fair runtime comparison for the unlearning algorithm by setting 30 epochs for the baseline architecture and 20 epochs for DASH. It's worth highlighting that after the unlearning process is completed, the inference time remains exactly the same as we revert DASH back to its original architecture (more details in Section 5.5).

4.3 Overall Results

Table 2 provides a comprehensive comparison between baseline ViT models and DASH across various unlearning algorithms. We observe the followings: (1) With the exception of retraining from scratch, all algorithms exhibit improved performance when the DASH architecture is applied, indicating the general effectiveness of DASH and its orthogonality to existing unlearning algorithms. (2) Across all four settings ({MUFAC, MUCAC} x {ViT-B-16, ViT-L-14}), DASH consistently achieves state-of-the-art results. (3) DASH appears to be relatively more effective for MUFAC. We presume this is because MUCAC is a relatively easier task, with a utility of around 95%, and might not require advanced unlearning methods to achieve decent unlearning performance.

As mentioned above, DASH is ineffective in the retraining case, and we delve into this aspect in detail in Section 5.3.

4.4 Significance Tests

Following the suggested significance test specifically designed to compare deep learning models by [6], we executed the provided official code to compare our DASH and the benchmark models for evaluation. This evaluation employs three components as inputs: the results from method A, the results from method B, and a designated confidence level, with the aim to address the following question. "Is A **almost stochastically greater** than B?" The theoretical underpinning of this assessment is the concept of "Almost Stochastic Order" a derivative of the "Stochastic Order" that allows for a more lenient comparison. The analysis concludes by generating a minimal ϵ value, serving as the basis for interpreting the comparative results of the two methods. Result is explained as follows:

- If $\epsilon > 0.5$: We can't say A is almost stochastically greater than B.
- If $\epsilon \leq 0.5$: We can say A is almost stochastically greater than B.
- If $\epsilon = 0$: We can say A is "stochastically greater" than B not just "almost stochastically greater" which is the best result we can get in this test. Mathematically meaning $\inf\{x : t \leq F(x)\} \geq \inf\{x : t \leq G(x)\}$ for $\forall t \in (0, 1)$, where F and G are the (empirical) CDFs of A and B.

Please refer to the paper for further details.

We perform the aforementioned significance test for each unlearning algorithm with and without the application of DASH, with the significance level of 0.05. The results are presented in Table 2, where dark blue signifies $\epsilon = 0$, light blue indicates $0 < \epsilon < 0.5$, and dark red indicates $\epsilon = 0$ for the baseline being stochastically greater than DASH. Observing the table, we can infer that, except for the "retrain" method, all unlearning algorithms exhibit improved performance when DASH is applied, confirming the general effectiveness of DASH.

5 ANALYSIS

We now conduct several ablation studies to justify our design decisions and explore the various advantages arising from the DASH architecture. We use the MUFAC dataset as the representative dataset for ablation studies.

5.1 Verifying the Key Motivation of DASH

In Section 3.1, we mentioned the vulnerability of upper layers to overfitting. While this tendency is well-established in prior works [10, 20], we reconfirm this characteristic empirically through ablation studies using MUL datasets and algorithms. The idea is to apply re-initialization selectively to different layers. Specifically, we compare two variants: ARU-bottom and ARU-top, which apply the re-initialization method from ARU [11] to the bottom half layers and upper half layers, respectively. Intuitively thinking, if ARU-top yields better performance compared to ARU-bottom, it indicates that the upper half layers have more parameters contributing to overfitting, thereby validating our motivation.

The results are summarized in Table 3. We can observe that ARU-top yields significantly better MUL performance in both ViT-Base and ViT-Large, confirming our primary motivation. Additionally, we notice that ARU-top performs even better than the ARU baseline, indicating that selective re-initialization of layers is more effective. Further exploration of this aspect would be an interesting future work.

5.2 Effectiveness of Parameter Sharing in DASH

To justify our design choice of sharing parameters between adjacent layers, we conduct an ablation study to assess the impact of parameter sharing on the effectiveness of DASH. We compare DASH with a variant where we duplicate layers without sharing the parameters. In this variant, the newly introduced layers are initialized with the corresponding adjacent layer but have distinct parameters, allowing them to have different values during unlearning. That is, we aim to determine whether the enhanced performance in DASH is solely attributed to the increased number of layers or if sharing the parameters also contributes to the performance improvement.

We compare these two models on the MUFAC dataset using finetune and SCRUB as the representative unlearning algorithms. Finetune serves as the most basic algorithm used in the majority of unlearning methods, while SCRUB is chosen as it represents the state-of-the-art method on MUFAC for ViT-B-16. The results are summarized in Table 4, showing that applying parameter sharing shows "stochastically greater" performance, verifying the effectiveness of sharing in DASH.

5.3 How Does DASH Maintain Utility in the Original Architecture?

A fundamental question regarding DASH is how it maintains a high utility score without employing the original forward pass. Our hypothesis is that this is achievable because we apply DASH to a set

	MUFAC				MUCAC			
Model	Utility (%, ↑)	Forget (%, ↓)	NoMUS (%, ↑)	ϵ	Utility (%, ↑)	Forget (%, ↓)	NoMUS (%, ↑)	ϵ
1. ViT-B-16								
Pre-trained	66.54	12.56	70.71	—	95.74	8.95	88.92	—
Unlearning Methods								
• Retrain	62.70 (±2.28)	5.94 (±1.20)	75.40 (±1.42)	0	95.05 (±0.10)	1.58 (±0.50)	95.94 (±0.50)	0
• DASH + Retrain	60.36 (±0.92)	5.55 (±0.61)	74.63 (±0.72)		93.33 (±1.20)	2.21 (±0.50)	94.23 (±0.56)	
• Finetune	64.78 (±0.79)	2.03 (±1.05)	80.36 (±0.81)	0	94.72 (±0.30)	3.27 (±0.38)	94.09 (±0.33)	0
• DASH + Finetune	65.25 (±0.71)	0.56 (±0.38)	82.07 (±0.36)		94.68 (±0.37)	2.77 (±0.42)	94.56 (±0.31)	
• CF-k	64.98 (±0.89)	2.37 (±1.27)	80.12 (±0.99)	0	94.72 (±0.21)	3.60 (±0.31)	93.75 (±0.23)	0
• DASH + CF-k	65.88 (±1.03)	0.85 (±0.65)	82.09 (±0.71)		94.65 (±1.02)	2.92 (±0.75)	94.42 (±0.38)	
• AdvNegGrad	63.10 (±1.26)	1.04 (±0.97)	80.51 (±0.85)	0	94.08 (±0.13)	0.42 (±0.43)	96.62 (±0.44)	2e-4
• DASH + AdvNegGrad	64.81 (±0.89)	0.89 (±0.86)	81.51 (±0.61)		**94.11 (±0.13)**	**0.22 (±0.27)**	**96.84 (±0.28)**	
• UNSIR	64.93 (±0.90)	2.26 (±0.76)	80.20 (±0.74)	0	94.78 (±0.18)	2.81 (±0.52)	94.58 (±0.45)	0
• DASH + UNSIR	65.12 (±1.31)	0.74 (±0.59)	81.82 (±0.74)		94.52 (±0.31)	2.23 (±0.41)	95.03 (±0.45)	
, • SCRUB	65.93 (±0.83)	1.45 (±0.76)	81.52 (±0.42)	0	93.97 (±1.35)	3.16 (±1.13)	93.83 (±0.59)	0
• DASH + SCRUB	**65.87 (±0.57)**	**0.22 (±0.18)**	**82.72 (±0.21)**		93.72 (±1.26)	2.25 (±0.86)	94.61 (±0.34)	
• ARU	62.70 (±0.79)	0.73 (±0.89)	80.62 (±0.77)	0	94.61 (±0.47)	2.98 (±0.54)	94.33 (±0.35)	0.13
• DASH + ARU	64.22 (±0.95)	0.62 (±0.56)	81.49 (±0.93)		94.43 (±0.22)	2.69 (±0.64)	94.53 (±0.62)	
	MUFAC				MUCAC			
Model	Utility (%, ↑)	Forget (%, ↓)	NoMUS (%, ↑)	ϵ	Utility (%, ↑)	Forget (%, ↓)	NoMUS (%, ↑)	ϵ
2. ViT-L-14								
Pre-trained	71.35	18.9	66.77	—	95.63	6.58	91.23	—
Unlearning Methods								
• Retrain	67.23 (±1.49)	5.05 (±0.64)	78.56 (±0.95)	0	94.88 (±0.14)	1.25 (±0.27)	96.19 (±0.24)	0
• DASH + Retrain	66.34 (±1.42)	5.58 (±1.41)	77.59 (±2.09)		94.74 (±0.24)	2.00 (±0.22)	95.38 (±0.35)	
• Finetune	67.75 (±1.40)	8.34 (±1.14)	75.53 (±0.56)	0	94.97 (±0.21)	4.22 (±0.27)	93.27 (±0.24)	0
• DASH + Finetune	67.12 (±1.28)	5.92 (±0.86)	77.64 (±0.53)		94.60 (±0.12)	2.78 (±0.38)	94.52 (±0.38)	
• CF-k	68.98 (±1.15)	10.13 (±1.07)	74.36 (±1.11)	0	94.85 (±0.10)	4.41 (±0.27)	93.01 (±0.30)	0
• DASH + CF-k	66.58 (±0.78)	6.47 (±1.03)	76.82 (±0.72)		94.67 (±0.11)	3.48 (±0.27)	93.85 (±0.24)	
• AdvNegGrad	66.33 (±1.84)	2.76 (±1.24)	80.40 (±0.98)	0	94.28 (±0.22)	0.41 (±0.31)	96.72 (±0.27)	0
• DASH + AdvNegGrad	66.08 (±1.97)	0.95 (±0.74)	82.09 (±1.32)		**94.58 (±0.14)**	**0.08 (±0.02)**	**97.21 (±0.06)**	
• UNSIR	66.81 (±1.86)	5.42 (±1.19)	77.98 (±0.62)	0	94.52 (±0.38)	2.42 (±0.29)	94.84 (±0.34)	0.18
• DASH + UNSIR	66.47 (±1.00)	2.58 (±0.71)	80.66 (±0.90)		94.33 (±0.30)	2.08 (±0.70)	95.09 (±0.58)	
, • SCRUB	67.28 (±3.38)	2.92 (±2.06)	80.72 (±1.14)	0	94.42 (±0.89)	3.51 (±0.99)	93.70 (±0.60)	4e-4
• DASH + SCRUB	67.15 (±0.90)	0.95 (±0.83)	82.63 (±1.17)		93.17 (±1.72)	2.41 (±1.28)	94.17 (±0.50)	
• ARU	65.07 (±1.38)	0.53 (±0.46)	82.01 (±0.59)	0	94.98 (±0.14)	2.87 (±0.39)	94.62 (±0.37)	0.01
• DASH + ARU	**66.86 (±0.66)**	**0.29 (±0.26)**	**83.15 (±0.38)**		94.49 (±0.26)	2.49 (±0.22)	94.75 (±0.19)	

Table 2: A comprehensive comparison between baseline model architecture and DASH architecture. Dark blue cells represent cases where applying DASH is "stochastically greater" than the baseline (i.e., $\epsilon = 0$), light blue cells represent cases where DASH is "almost stochastically greater" than the baseline (i.e., $\epsilon < 0.5$). Dark red cells correspond to "retrain" method where applying DASH does not show any significant improvement. The symbols ↑ /↓ indicate that higher/lower values are better for the respective metrics. All scores represent the average of 5 runs. State-of-the-art results based on NoMUS score is represented in bold. We can observe that DASH is consistently effective when applied to any unlearning algorithm except retraining. We investigate the case of retraining in detail in Section 5.3. Best viewed in color.

	MUFAC		
Model	Utility	Forget	NoMUS
1. ViT-B-16			
ARU	62.70 (±0.79)	0.73 (±0.89)	80.62 (±0.77)
ARU-bottom	63.03 (±0.81)	0.43 (±0.41)	81.45 (±0.50)
ARU-top	64.93 (±0.60)	0.26 (±0.10)	82.12 (±0.27)
2. ViT-L-14			
ARU	65.07 (±1.38)	0.53 (±0.46)	82.01 (±0.59)
ARU-bottom	67.93 (±0.32)	2.21 (±0.64)	81.75 (±0.47)
ARU-top	65.53 (±1.70)	0.38 (±0.44)	82.22 (±0.17)

Table 3: Ablation study to reconfirm the well-known tendency that the upper layers in Transformer models are more prone to overfitting compared to bottom layers. From the table we can observe that ARU-top is significantly better than ARU-bottom, verifying that the upper layers have more parameters that contribute to overfitting.

	MUFAC			
Model	Utility	Forget	NoMUS	ϵ
1. ViT-B-16				
DASH + Finetune	65.25 (±0.71)	0.56 (±0.38)	82.07 (±0.36)	0
DASH + w/o sharing + Finetune	64.99 (±1.00)	0.88 (±0.35)	81.27 (±0.82)	
DASH + Scrub	65.87 (±0.57)	0.22 (±0.18)	82.72 (±0.21)	0
DASH + w/o sharing + Scrub	65.32 (±1.14)	0.55 (±0.40)	81.42 (±0.57)	

Table 4: Ablation study to validate the effectiveness of parameter sharing in DASH.

of pre-trained parameters instead of starting the training process from scratch. Intuitively, given that θ_0 is already pre-trained on D_{train} and likely to be close to a local optimum, updating based on f_{dash} would result in parameters that are less likely to deviate significantly from this local optimum. In contrast, if we were to train a model from scratch, the performance difference between f and f_{dash} could vary more significantly. We validate this hypothesis through the following experiment, tracking the changes in the utility of f and f_{dash} using "finetune" and "retrain" as the unlearning algorithms. The results are depicted in Figure 3. As shown in the figure, due to the superior initial starting point θ_0, "finetune"-f maintains high utility even when only f_{dash} is used during unlearning. Conversely, "retrain"-f consistently exhibits lower performance due to the random starting point, providing insight into the reason for the poor performance of "DASH + Retrain".

5.4 Fast Convergence of DASH

The runtime of the unlearning algorithm is an important aspect in MUL, although not the primary focus of this paper, as the majority of MUL methods outperform the retraining baseline. Nevertheless, an unlearning algorithm that achieves decent performance in limited amount of unlearning time is undoubtedly a desirable property. Therefore, we report the results of running unlearning

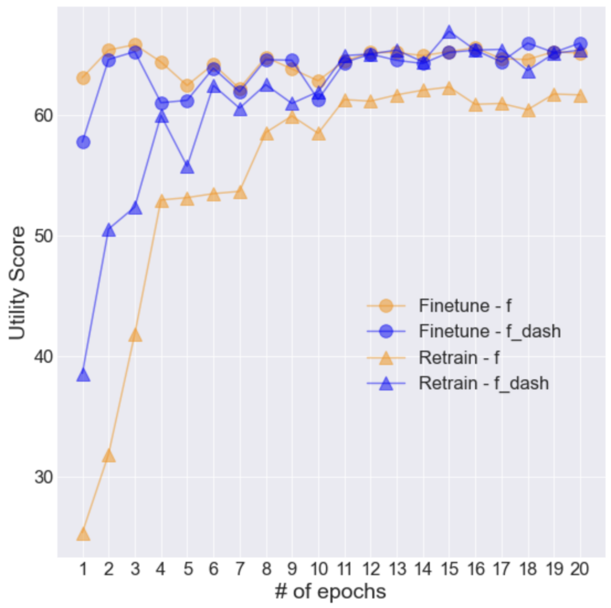


Figure 3: Reporting the changes in utilities for f and f_{dash} when employing "finetune" and "retrain" as the unlearning algorithms. It is evident that due to the superior initial starting point θ_0, "finetune"-f easily sustains a high utility score, even when only f_{dash} is used during unlearning.

algorithms with a maximum of 10 epochs for the baselines and 7 epochs for DASH. The results are reported in Table 5. We observe that DASH still outperforms the majority of the baseline algorithms, highlighting the fast convergence of DASH.

	MUFAC			
Model	Utility	Forget	NoMUS	ϵ
1. ViT-B-16				
Finetune	64.60 (±1.16)	1.75 (±1.11)	80.55 (±0.93)	0
DASH + Finetune	64.79 (±1.09)	0.50 (±0.48)	81.90 (±0.51)	
CF-K	64.74 (±0.81)	2.41 (±1.26)	79.96 (±1.12)	0
DASH + CF-K	65.08 (±0.80)	1.11 (±0.78)	81.43 (±0.79)	
AdvNegGrad	63.10 (±1.26)	1.04 (±0.97)	80.51 (±0.85)	0
DASH + AdvNegGrad	64.81 (±0.89)	0.89 (±0.86)	81.51 (±0.61)	
UNSIR	63.44 (±0.93)	1.94 (±0.61)	79.78 (±1.02)	0
DASH + UNSIR	64.53 (±1.08)	0.75 (±1.05)	81.52 (±1.55)	
SCRUB	62.56 (±3.04)	1.90 (±2.11)	79.38 (±1.13)	0
DASH + SCRUB	65.63 (±0.47)	1.23 (±0.52)	81.58 (±0.32)	
ARU	61.08 (±1.25)	0.97 (±0.51)	79.56 (±0.80)	0
DASH + ARU	62.57 (±0.34)	0.49 (±0.27)	80.80 (±0.40)	

Table 5: Time efficiency of DASH. With limiting the number of unlearning epochs to ten for baseline and seven for DASH, DASH still shows significantly higher performance, verifying the fast convergence of DASH.

5.5 Runtime Comparison

As briefly mentioned in Section 4.2, DASH requires more time to run the unlearning process due to the increased number of layers in the forward pass. A comparison of the runtime between DASH and the baseline architecture is presented in Table 6. The MUFAC dataset is employed to measure both training and inference times. A single A100 GPU is used for each model, and the time required for one epoch of unlearning, as well as the time for inference on the MUFAC test set, is measured. Each run is performed 10 times, and the reported results in the table represent the averages. We used the default hyperparameters (i.e., $(n, m) = (6, 2), (12, 2)$ for ViT-B and ViT-L, respectively). Note that the inference time is equivalent with the baseline since we revert back to the original architecture when unlearning is finished.

Model	Train time (s)	Inference time (s)
ViT-B-16	35.77	2.38
ViT-B-16 + DASH	45.32 (×1.27)	2.38 (×1.00)
ViT-L-14	102.61	6.32
ViT-L-14 + DASH	144.38 (×1.41)	6.32 (×1.00)

Table 6: Comparing the runtime of DASH and the baseline architecture, where hyperparameters n and m are set to their default values (i.e., n equal to half of the number of layers and m equal to 2). It's important to note that the inference time is exactly the same, as we revert back to the original architecture after the unlearning process is complete. Additionally, we adjusted the number of epochs, with 30 epochs for the baseline and 20 for DASH, to match the total unlearning time.

5.6 Limitations

One limitation of DASH is that it is not easily applicable to ResNet-based models, which have different input-output sizes for several layers. The only prerequisite for applying DASH is that the output and input shapes do not change, a condition satisfied by Transformers but not ResNets. Nevertheless, given that ViT has become the predominant model architecture in computer vision, gradually replacing ResNets, we believe our work makes a meaningful contribution.

6 CONCLUSION

In this study, we present a simple yet effective approach to machine unlearning by focusing on modifying the model architecture, marking the first instance of such a strategy being proposed. Our method specifically targets the unlearning in Vision Transformer models through direct architectural alterations. We evaluate the utility and forgetting capability of our model in comparison to contemporary, high-performing machine unlearning methods. Furthermore, we conduct a detailed analysis to explain the underlying mechanisms and design choices that contribute to the effectiveness of our proposed method. To our knowledge, this is the first work to modify the Vision Transformer's architecture to enhance machine unlearning performance.

REFERENCES

[1] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 141–159.

[2] Ikhyun Cho and U Kang. 2022. Pea-KD: Parameter-efficient and accurate Knowledge Distillation on BERT. *Plos one* 17, 2 (2022), e0263592.

[3] Dasol Choi and Dongbin Na. 2023. Towards machine unlearning benchmarks: Forgetting the personal identities in facial recognition systems. *arXiv preprint arXiv:2311.02240* (2023).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2010. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv 2020. *arXiv preprint arXiv:2010.11929* (2010).

[6] Rotem Dror, Segev Shlomov, and Roi Reichart. 2019. Deep dominance-how to properly compare deep neural models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2773–2785.

[7] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[8] Shashwat Goel, Ameya Prabhu, Amartya Sanyal, Ser-Nam Lim, Philip Torr, and Ponnurangam Kumaraguru. 2022. Towards adversarial evaluations for inexact machine unlearning. *arXiv preprint arXiv:2201.06640* (2022).

[9] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9304–9312.

[10] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language?. In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

[11] Yoonhwa Jung, Ikhyun Cho, Shun-Hsiang Hsu, and Julia Hockenmaier. 2024. Attack and Reset for Unlearning: Exploiting Adversarial Noise toward Machine Unlearning through Parameter Re-initialization. *arXiv preprint arXiv:2401.08998* (2024).

[12] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. https://api.semanticscholar.org/CorpusID:18268744

[13] Meghdad Kurmanji, Peter Triantafillou, and Eleni Triantafillou. 2023. Towards Unbounded Machine Unlearning. *arXiv preprint arXiv:2302.09880* (2023).

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

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

[16] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

[17] Ron Mokady, Amir Hertz, and Amit H. Bermano. 2021. ClipCap: CLIP Prefix for Image Captioning. arXiv:2111.09734 [cs.CV]

[18] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[19] Peyman Passban, Yimeng Wu, Mehdi Rezagholizadeh, and Qun Liu. 2021. Alp-kd: Attention-based layer projection for knowledge distillation. In *Proceedings of the AAAI Conference on artificial intelligence*, Vol. 35. 13657–13665.

[20] Tairen Piao, Ikhyun Cho, and U Kang. 2022. SensiMix: Sensitivity-Aware 8-bit index & 1-bit value mixed precision quantization for BERT compression. *PloS one* 17, 4 (2022), e0265621.

[21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[22] State of California Department of Justice. 2018. California Consumer Privacy Act of 2018. https://oag.ca.gov/privacy/ccpa

[23] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355* (2019).

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

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[26] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*

10, 3152676 (2017), 10–5555.

[27] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. arXiv:2203.05482 [cs.LG]

[28] Yinjun Wu, Edgar Dobriban, and Susan Davidson. 2020. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*. PMLR, 10355–10366.