Analysis of SCRUB and NegGrad+ Machine Unlearning Algorithms

Adam Michał Suszczewicz
Department of Computer Science
University of Warwick

29 April 2025

Abstract

The use of Machine Learning has greatly helped move the world forward, but is constantly threatened by regulations such as the Right to Erasure in the GDPR, which currently requires deletion and retraining of used models. Machine Unlearning attempts to aid this by forgetting certain points that have been requested to be deleted while maintaining its accuracy and performance without the need for retraining. This project evaluates the performance of two state-of-the-art Machine Unlearning algorithms being SCRUB and NegGrad+ alongside their RUM variations. The experiments were conducted using model trained on two different Convolutional Neural Network architectures being VGG-16 and ResNet-18 and tested on the CIFAR-10 and CIFAR-100 datasets.
Key Words: Machine Learning, Machine Unlearning, SCRUB, NegGrad+, Convolution Neural Networks

Contents

Chapter 1

Introduction

1.1 Preface

Modern technology is becoming increasingly reliant on Machine Learning and AI models in order to perform various tasks or automate different processes. In 2024 alone, 80% of companies have claimed that the use of machine learning has increased their revenue [1]. However, alongside this come increasingly stringent regulations and privacy issues that must be complied. One major regulation is the EU's GDPR Right to Erasure [12] which states that a company must remove all instances of the users data even if it was used in machine learning models. This causes many issues as the simplest way of solving this is by retraining the entire model with the single data point removed, however, the large amount of time and resources required for this imply there needs to be more efficient methods.
Another critical challenge is data poisoning [24], a type of attack that is specifically directed at the training data of machine learning models, which introduces heavily biased or contradictory data that causes the generated model to perform suboptimally.
These issues being forward the need for Machine Unlearning which is the process of removing the influence of a specific set of data from a fully trained Machine Learning model. There are many different algorithms and methods to do this, however they are not fully explored. One major gap in current knowledge is the effects of sequential unlearning on a single dataset. This involves using the output model of an unlearning step as a base for future unlearning alongside a different forget set.
This project will focus on analysing the performance of two unique machine unlearn-

ing algorithms, being SCRUB and NegGrad+, using both their vanilla version, which is the base algorithm, and their RUM version, which uses an additional refinement step to try to improve performance. These will be tested on two datasets, CIFAR-10 and CIFAR-100, with the models being trained using the VGG-16 and ResNet-18 architectures. The performance will be analysed using both Tug-of-War (ToW) and Tug-of-War-MIA (ToW-MIA) as well as various useful accuracies for all models, such as test accuracy, retain set accuracy and forget set accuracy.

1.2 Objectives

To summarise, for the project to be deemed successful the following should be accomplished:

- Obtain test accuracy scores over 5 unlearning steps for both algorithms (SCRUB and NegGrad+) over all datasets and architectures and compare them to test accuracies of a model that has been retrained from scratch.
- Obtain ToW and ToW-MIA scores for all 5 proxies for the RUM versions of both algorithms over all datasets and architectures.
- Compare algorithms and their performance for each experiment and provide meaningful conclusions.
- Derive use cases for each unlearning algorithm based on their strengths and weaknesses.

Chapter 2

Background

In the following section, $\theta = \mathcal{A}(D)$ is used to denote a model that has had algorithm $\mathcal{A}$ applied to dataset D. Then there exists a forget set D_f such that $D_f \subset D_{train}$ and a retain set D_r such that $D_r = D_{train} \setminus D_f$. $(x, y) \in D$ refers to an example x and its corresponding ground truth y for dataset D. $f(x; w)$ refers to the prediction of a model for example x using a model with weights w.

2.1 Memorization

Machine learning models tend to memorize more unique data points during the learning phase to achieve the highest accuracy. Memorization, as defined by Feldman [6], can be calculated in the following way:

$$\text{mem}(\mathcal{A}, \mathcal{D}, i) = \Pr_{f \sim \mathcal{A}(\mathcal{D})}[f(x_i) = y_i] - \Pr_{f \sim \mathcal{A}(\mathcal{D} \setminus i)}[f(x_i) = y_i] \tag{2.1}$$

This calculates the memorization level for a data point i in a model that has been trained using algorithm $\mathcal{A}$ and dataset $\mathcal{D}$. The first term is the probability of a correct prediction of the original model $\mathcal{A}(\mathcal{D})$ while the second term is the probability of a correct prediction of a model trained without the singular data point i, $A(D \setminus i)$. A higher value of memorization implies that the data point is more strongly memorized, as it has a much greater effect on the class for which it is trained. As a unique model is required for each data point, calculating memorization levels for an entire dataset is very computationally expensive. Knowing the memorization level of a data point is necessary as some data points are much more influential on the performance of the

model compared to other, therefore forgetting those may require additional changes or longer unlearning times.
Although memorization is not used by either SCRUB or NegGrad+, it is important to analyse the effect unlearning has on data points with different levels of memorization in order to determine how generalizable these algorithms are. For simplicity, an unlearning algorithm should work well across all memorization levels so that it can be easily applied to many different datasets. However, from a performance perspective, an unlearning algorithm performing especially well on a certain level of memorization could lead to better overall unlearning performance if it only targets the data points its good at forgetting.

2.2 Memorization Proxies

To combat the long calculation times of memorization, memorization proxies are introduced. These are ways of estimating the memorization levels of data points without requiring another model to be trained. This work will focus on five different proxies.

2.2.1 Learning Event Proxies

Learning event proxies, introduced by Jiang et al. [13] as learning speed based proxies, are a way of utilising certain features during the training process to compute metrics that have strong correlation to the memorization score while remaining trivial to compute. These include confidence, max confidence, entropy and binary accuracy. These are calculated during each epoch and then averaged out to give a final score, as it produces much more stable results that have stronger correlation to memorization.
Confidence is the softmax probability of an example x_i being assigned its correct label y_i, **max confidence** is the maximal softmax probability over all classes, **entropy** is the negative entropy of the softmax confidences and **binary accuracy** is the binary measure of if an example x_i was labelled correctly as y_i.

2.2.2 Holdout Retraining

The other proxy being used is holdout retraining, introduced by Carlini et al. [4]. This measures the typicality or atypicallity of data points by using the symmetric KL divergence between the original model and a model that was fine-tuned on the test

data. High values for this proxy imply that the predicted label y_i for example x_i is very different after fine-tuning meaning that x_i is atypical to the rest of the data therefore more heavily memorized.

2.3 Architectures

2.3.1 VGG-16

VGG-16 [20] is a type of deep Convolutional Neural Network (CNN) that has its layers divided into three sperate groups: convolution, max pooling and fully connected layers. This allows for improvements over other CNNs due to its simplicity, performing convolutions over 3x3 fields, and its performance, with strong performance in both learning [20] and unlearning [22]. The version of VGG-16 used in this work also utilises both batch normalization and the lottery ticket hypothesis. Batch normalization speeds up both the training and unlearning process by normalising the input. This allows for easier and faster calculations due to layers having consistent distributions for each batch. It also uses the lottery ticket hypothesis which states that a small subnetwork of the entire CNN can be trained on to achieve a similar result [8]. This is used during both training and unlearning as there may be no need to backpropagate throughout the entire network to achieve similar accuracies. Both of these help reduce training times and henceforth unlearning times due to the nature of the unlearning algorithms working as additional training steps.

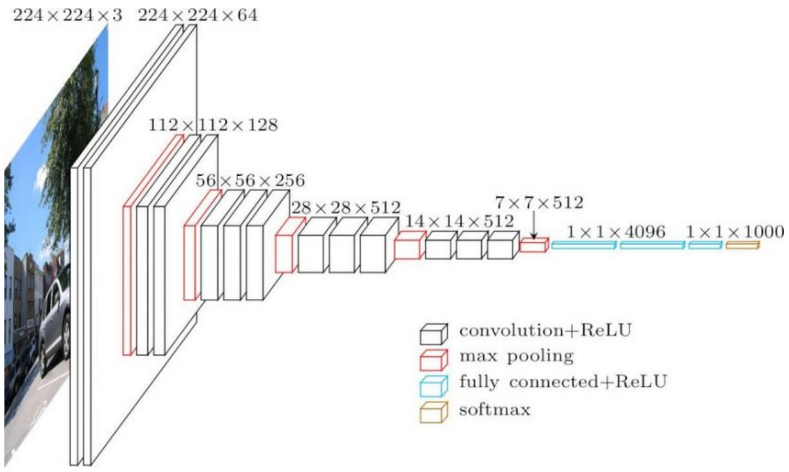

Figure 2.1: Architecture Network of VGG-16 [5]

2.3.2 ResNet-18

ResNet-18 is another type of deep CNN which utilises residuals to improve performance [11]. These residuals link further back to previous nodes as seen in Figure 2.2. These allow for more links to previous nodes which allows for a larger range of gradients during backpropagation which further speeds up training times. ResNet-18 is an 18-layer variation of the ResNet model meaning there are 17 convolutional layers and a single fully connected layer that provides the output. This also utilises batch normalization after each convolutional layer to speed up convergence among other benefits. These features make ResNet-18 a good choice for machine unlearning due to its simplicity and performance. Due to it containing only 18 layers, updating weights is very fast compared to other ResNet versions such as ResNet-50 or ResNet-100 while maintaining good performance, especially with smaller datasets for similar experiments [23] [22].

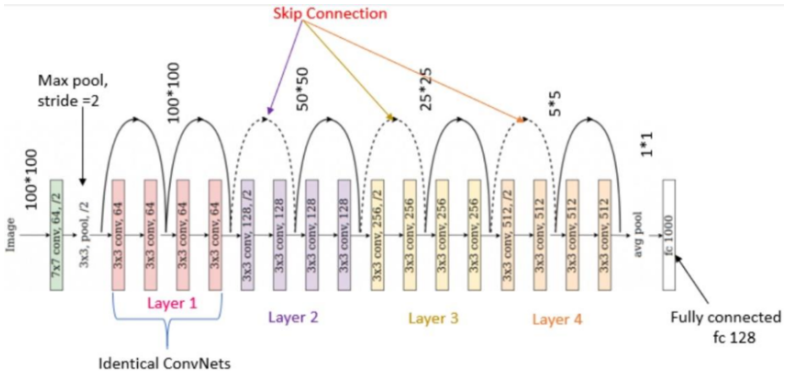

Figure 2.2: Architecture Network of ResNet-18 [18]

2.4 Datasets

The experiments used in this work span across two unique datasets. These are CIFAR-10 and CIFAR-100 [15]. CIFAR-10 consists of 60,000 32x32 coloured images, split across 10 classes, each with 6,000 images. These are then further split into 45,000 training examples, 5,000 validation examples and 10,000 test examples. CIFAR-100 consists of the same amount of images with the same resolution however split across 100 classes with 600 images each. The training, validation and testing split remains the same.
Using these 2 datasets allows for analysis of performance over both a small and large environment. CIFAR-10 is much smaller in terms of classes compare to CIFAR-100

therefore experiments done on both of these will give a good overview on how the number of classes affect these different unlearning algorithms.

2.5 Algorithms

2.5.1 SCRUB

SCRUB, introduced by Kurmanji et al. [16], is a student-teacher based approach for unlearning. It works by using the forget set D_f and retain set D_r to train a student model w^o based on the original model's weights (the teacher model) w^u. The student model w^o will obey and learn about D_r while ignoring data concerning D_f and, therefore, will not learn it. This will result in the new model w^o effectively forgetting the set D_f. This can be done by optimising the following objective function:

$$\min_{w^u} \frac{\alpha}{N_r} \sum_{x_r \in \mathcal{D}_r} d(x_r; w^u) + \frac{\gamma}{N_r} \sum_{(x_r, y_r) \in \mathcal{D}_r} l(f(x_r; w^u), y_r) - \frac{1}{N_f} \sum_{x_f \in \mathcal{D}_f} d(x_f; w^u) \tag{2.2}$$

Here the l function is the cross entropy loss of the student model compared to the ground truth values, the f function is the prediction of values x using the model with weights w^u and the d function is the KL divergence between the student and teacher model as shown below.

$$d(x; w^u) = D_{\mathrm{KL}}(p(f(x; w^o)) \| p(f(x; w_u)))$$

α and γ are hyperparameters that control different parts of the algorithm. Mainly, α controls the impact of the retain set distributions changing, while γ, controls the impact of the model's correctness. Balancing these factors requires careful consideration to ensure that the model remains generally accurate, as indicated by the test accuracy, while effectively removing the influence of the forget set.
This objective function is minimized by alternating between maximize and minimize steps. Each maximize step attempts to optimize the forget set error using the negative KL divergence, while each minimize step attempts to optimize the retain set error using both the KL divergence over the retain set and the cross entropy loss of the current model. The number of maximize steps is lower than the total steps as a series of minimize steps are performed at the end to attempt to preserve the original performance of the retain set.
This algorithm can be formulated as:

Algorithm 1 SCRUB

Input: Original model weights w^o
Input: Total unlearn epochs `UNLEARN_EPOCHS`
Input: Total max steps `MAX_STEP`
$w^u \leftarrow w^o$
$i \leftarrow 0$
while $i <$ `UNLEARN_EPOCHS` **do**
  if $i <$ `MAX_STEP` **then**
    $w^u \leftarrow$ `DO_MAX_STEP`(w^u)
  end if
  $w^u \leftarrow$ `DO_MIN_STEP`(w^u)
  $i \leftarrow i + 1$
end while

As weights are updated twice for each maximize step, this does make SCRUB computationally expensive due to the additional calculations needed. DO_MIN_STEP and DO_MAX_STEP perform the following:

Algorithm 2 `DO_MAX_STEP`

Input: Student model weights w^u
Input: Batch size B
Input: Forget set D_f
Input: Learning rate ϵ
$b \leftarrow$ `NEXT_BATCH`(D_f, B)
while b IS NOT EMPTY **do**
  $w^u \leftarrow w^u + \epsilon \nabla_{w^u} \frac{1}{|b|} \sum_{x_f \in b} d(x_f; w^u)$
  $b \leftarrow$ `NEXT_BATCH`(D_f, B)
end while

Algorithm 3 `DO_MIN_STEP`

Input: Student model weights w^u
Input: Batch size B
Input: Retain set D_r
Input: Learning rate ϵ
$b \leftarrow$ `NEXT_BATCH`(D_r, B)
while b IS NOT EMPTY **do**
    $w^u \leftarrow w^u + \epsilon \nabla_{w^u} \frac{1}{|b|} \sum_{(x_r, y_r) \in b} \alpha d(x_r; w^u) + \gamma l(f(x_r; w^u), y_r)$
    $b \leftarrow$ `NEXT_BATCH`(D_r, B)
end while

As these algorithms use batch unlearning, both the min and max steps update the weights for a batch of data each time, until all batches are complete which is why the `NEXT_BATCH` function is necessary.
Previous work by Kurmanji et al. [16] has shown that SCRUB is excellent at forgetting data, while maintaining good retain set accuracy. Works from Zhao et al. [22] show that SCRUB performs well in unlearning via ToW scores and MIA scores alongside its RUM version, with the RUM version performing noticeably better. SCRUB has also been tested in "corrective machine learning" by Goel et Al. [9] where it is shown to perform well in resolving interclass confusion, where examples from 2 different classes are incorrectly labelled leading to confusion between these specific classes, but does not work well in the case of data poisoning, where both examples and labels are added that are either contradictory to the class or that exploit another feature of the model to make it perform worse.

2.5.2 NegGrad+

NegGrad+, proposed by Kurmanji et al. [16] as an evolution of NegGrad [10], works by fine-tuning the original trained model w^o on both D_r and D_f. This is performed in a way such that is maximises the loss on the forget set while minimising the loss on the retain set. It does this by optimising the following loss function:

$$\mathcal{L}(w) = \beta \times \frac{1}{|D_r|} \sum_{i=1}^{|D_r|} l(f(x_i; w), y_i) - (1 - \beta) \times \frac{1}{|D_f|} \sum_{j=1}^{|D_f|} l(f(x_j; w), y_j) \tag{2.3}$$

β is a tunable hyperparameter that should be kept in the range [0,1] that attempts to control the weight between remembering D_r and forgetting D_f. The difference

between NegGrad and NegGrad+ is the usage of D_r in the loss function which provides an additional focus in retaining performance of the retain set.
This can be formally defined as:

Algorithm 4 NegGrad+

Input: Retain set D_r
Input: Forget set D_f
Input: Original model weights w
Input: Total unlearn epochs `UNLEARN_EPOCHS`
while $i <$ `UNLEARN_EPOCHS` **do**
    $w \leftarrow w + \epsilon \nabla_w \left(\frac{\beta}{|D_r|} \sum_{(x_i, y_i) \in D_r} l(f(x_i; w), y_i) - \frac{1-\beta}{|D_f|} \sum_{(x_j, y_j) \in D_f} l(f(x_j; w), y_j) \right)$
    $i \leftarrow i + 1$
end while

Previous work from Kurmanji et al [16] show NegGrad+ to be a good machine unlearning algorithm for privacy related issues, being only consistently beaten by SCRUB in user privacy, resolving confusion and removing biases tests. Further work by Zhao et al. [23] [22] show that NegGrad+ is a top contender for being applied to RUM especially for larger datasets such as CIFAR-100 where it becomes the best performer for all memorization levels. There is also a start to sequential unlearning using NegGrad+ over 5 unlearning steps, with both the RUM versions and vanilla versions of the algorithm. These results can be further expanded on by utilising a wider variety of architectures, datasets and proxies to decide the RUM refinement process.

2.5.3 RUM

RUM, introduced by Zhao et Al. [22], is a meta unlearning algorithm that consists of two parts. The first is refinement which splits D_f into K disjoint subsets with each $x_i \in D_f$ appearing in exactly 1 subset. These are split in a way such that some unlearning algorithm can exploit any structural or statistical features in that subset to achieve the best unlearning performance, The second stage consists of applying an unlearning algorithm from a pool of state-of-the-art algorithms that would perform best on that specific subset. These are unlearned in sequence such that step $i + 1$ will use the output of step i as its base model for unlearning. However, due to the optimal split of D_f and the optimal algorithm to use for each case being currently unknown, these parameters are currently deinfed by the user. This means that the current "best"

version of RUM utilises splits based on memorization levels, with the 3 levels being low, medium and high memorization based on scores calculated via the proxies, and applies currently known algorithms based on the dataset. Currently, the best RUM ingredients for CIFAR-10 are None, Fine-Tune, and SalUn, while for CIFAR-100, it is NegGrad+ across all splits. None indicates that nothing needs to be done as data points with a low memorization level have such little influence on the model that forgetting them does not change anything [22].

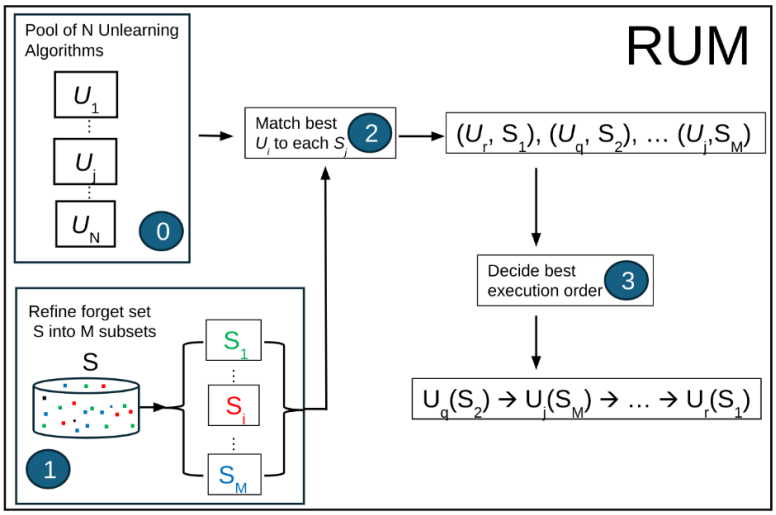

Figure 2.3: Visual representation of RUM [22]

From previous works by Zhao et al. [22], RUM has been shown to perform significantly better when forgetting the same set of data compared to an algorithm on its own with specific highlights being NegGrad+, achieving consistently high ToW scores across many datasets and architectures and SCRUB achieving a very high ToW score of 0.945 ± 0.017 in the single test it was used.
In addition, further works by Zhao et al. [23] begin the exploration of sequential unlearning with limited results for NegGrad+ specifically. This allows for a solid baseline for expectations while opening the doors for further exploration for many different algorithms, datasets and architecture combination to be experimented with.
As RUM is an ever evolving algorithm that relies on accurate and plentiful knowledge of various state-of-the-art unlearning algorithms, further testing of NegGrad+ and SCRUB using a variety of architecture and dataset combinations will give further insight into a more robust and correct refinement process and the generalisability of

these algorithms.

2.6 Analysis Methods

2.6.1 Membership Inference Attack

Membership Inference Attack (MIA) is a security focused algorithm that tests for the presence of specific data inside a dataset. This is further extended for machine learning by testing for the presence of data in the training set of the model through various methods such as using shadow models and binary classifiers to distinguish memberships using the models' output confidence scores [19]. Performing MIA on a set of data will yield a score between 0 and 1, based on the confidence of that set being included. A score of 1 indicates absolute certainty that the data is included while a score of 0 shows the data is definitely not included. The optimal value for MIA is 0.5 where the membership of the data is uncertain as that provides the highest level of privacy for a user's data.
This can be used to analyse the forgetting efficacy of a machine unlearning algorithm by creating a binary classifier to distinguish between the retain and test set and querying it with examples from the forget set to check how well it was unlearned. This provides a good measure of how effective the unlearning algorithm is at actually forgetting the data and being secure in terms of user privacy.

2.6.2 Accuracies

The performance of the algorithms can be analysed based on their test accuracy, forget set accuracy and retain set accuracy. These will all provide valuable information for how good the algorithm is at unlearning. Test accuracy will show how well the algorithm performs on unseen data after unlearning has been done. This value should stay around the same throughout or potentially increase if unlearning can improve the model. Forget set accuracy will focus on how well the model can predict samples from the forget set. This should decrease as it will no longer be trained on these examples therefore a lower performance is expected, however it should stay in line with test accuracy. Retain set performance another important metric which will show how well the model predicts the data from the retain set after unlearning. This should not decrease as that would mean that the algorithm has negatively impacted the retain set performance, harming the model overall.

These will all be compared to a model that has been retrained from scratch that will be used as a baseline for unlearning.

2.6.3 Tug-of-War

Another metric that will be used is Tug-of-War (ToW) and its privacy focused version Tug-of-War-MIA (ToW-MIA). ToW, as introduced by Zhao et al. [22], captures the similarity of an unlearned model against a model that has been retained from scratch using a value between 0 and 1 with 1 indicating identical performance and 0 indicating contradictory performance. This is calculated in the following way:

$$\text{ToW}(\theta_u, \theta_r, D_f, D_r, D_t) = (1 - \Delta_a(\theta_u, \theta_r, D_f)) \cdot (1 - \Delta_a(\theta_u, \theta_r, D_r)) \cdot (1 - \Delta_a(\theta_u, \theta_r, D_t)) \tag{2.4}$$

Here, $\Delta_a(\theta_u, \theta_r, D)$ is the absolute difference in accuracies in the unlearned model θ_u and the retrained model θ_r. This is calculated using:

$$\Delta_a(\theta_u, \theta_r, D) = |\text{a}(\theta^u, \mathcal{D}) - \text{a}(\theta^r, \mathcal{D})|$$

where $\text{a}(\theta, \mathcal{D})$ is $\frac{1}{|D|} \sum_{(x,y)\in\mathcal{D}} (f(x;\theta) = y)$ which is the accuracy of a model θ on dataset $\mathcal{D}$ making the previous equation the absolute difference between them.
ToW-MIA is a privacy focused version of ToW which attempts to compare both the accuracy of the models as well as the effectiveness of for getting. This is done by using MIA on the forget set to see if there is any trace of the data point still present in the retain set. This is calculted using the following:

$$\text{ToW-MIA}(\theta_u, \theta_r, D_f, D_r, D_t) = (1 - \Delta_m(\theta_u, \theta_r, D_f)) \cdot (1 - \Delta_a(\theta_u, \theta_r, D_r)) \cdot (1 - \Delta_a(\theta_u, \theta_r, D_t)) \tag{2.5}$$

The difference between ToW and ToW-MIA is the calculation of difference in MIA for the forget set in ToW-MIA. This allows a balance of privacy and performance in a single metric. This difference in MIA is calculated by training a binary classifier to distinguish between the retain and test set and is queried with examples from D_f. This means that if the binary classifier thinks that the data point belongs in the retain set, then it has not been forgotten effectively as there is still traces of its influence in the retain set. However, if it results in the test set then it was successfully forgotten. The aim is for this value to be 0.5 where there is absolute uncertainty for the membership of the forget set data.
By using both ToW and ToW-MIA, it will give insights into different aspects that the unlearning algorithm excels in. For example if ToW-MIA scores are higher than ToW scores, then the algorithm performs better in terms of privacy compared to accuracy.

Chapter 3

Experiment Setup

3.1 Implementation

The first steps into setting up the experiments was understanding the codebase that was being used. This project uses the RUM GitHub repository [21] as a base for all unlearning methods for the datasets. The codebase mainly uses PyTorch [2] to perform all machine learning and unlearning tasks. This is used for reproducibility of results.

3.1.1 SCRUB

For SCRUB, each epoch firstly loads the forget set and retain set separately and initialises the loss function with the KL divergences and the cross entropy loss through the use of torch.nn library. Then there are 2 models that are used, the constant teacher model that does not get updated through the use of torch.no_grad(), and the student model that is initialised normally.

```
logit_s = model_s(input)
with torch.no_grad():
    logit_t = model_t(input)

criterion_cls = nn.CrossEntropyLoss()
criterion_div = DistillKL(args.kd_T)
criterion_kd = DistillKL(args.kd_T)

criterion_list = nn.ModuleList([])
```

```
criterion_list.append(criterion_cls)
criterion_list.append(criterion_div)
criterion_list.append(criterion_kd)
```

Then it updates the loss differently depending on if the current step is a minimize or maximize step. In the case of maximize it uses the negative KL divergence between the distributions of the models based on forget set and for minimize it uses the KL divergence between the distributions of the models based on the retain set and the cross entropy loss of the student model.

```
if split == "minimize":
    loss = gamma * loss_cls + beta * loss_div
elif split == "maximize":
    loss = -loss_div
```

It then updates the model accordingly through the use of backpropagation and prints relevant information to keep track of what is happening throughout.

3.1.2 NegGrad+

For NegGrad+, it similarly loads two models, one that uses the forget set data and another that uses the retain set data. Then it uses the custom loss function that can be seen in 2.3 and continues as normal.

```
output_clean = model(input)
del_output_clean = model(del_input)
r_loss = criterion(output_clean, target)
del_loss = criterion(del_output_clean, del_target)

loss = args.alpha*r_loss - (1-args.alpha)*del_loss
```

It calculates the cross entropy loss for both the retain set and forget set and their corresponding ground truths and uses those values to update the loss. This is where the hyperparameter α plays a big role as it dictates the weight of each set. Intuitively, the value of α should be closer to 1 as additional training on the retain set will improve its score while slowly phasing out the influence of the forget set, the additional of the second term just speeds up the rate at which the forget set will lose its influence.

3.1.3 RUM

RUM, being a meta algorithm, simply takes in the sequence of unlearning algorithms and performs them sequentially. This is done by having the user input a sequence of unlearning steps and performing them all sequentially, with the input of step i being the output of step $i-1$. The last step that is performed is copied into another file that allows for evaluation of the overall performance of the algorithm. There are also files generated at each step allowing for evaluation of each sequential part of the RUM process.

```
runs = [
        f"python main_forget.py --seed {seed} --no_aug --
           sequential --mem_proxy {proxy} --mem low --unlearn {
           unlearn} --unlearn_step {unlearn_step} ... --mask '${
           original_model_path}'",
        f"python main_forget.py --seed {seed} --no_aug --
           sequential --mem_proxy {proxy} --mem mid --unlearn {
           unlearn} --unlearn_step {unlearn_step } ... --mask '${
           low_model_path}'",
        f"python main_forget.py --seed {seed} --no_aug --
           sequential --mem_proxy {proxy} --mem high --unlearn {
           unlearn} --unlearn_step {unlearn_step} ... --mask '${
           mid_model_path}'",
        # Evaluation
        f"python main_forget.py --seed {seed} --no_aug --unlearn
           seq_mix --mem_proxy {proxy} --mem mix --unlearn_step {
           unlearn_step} ... --mask '${high_model_path}'"
]
```

In this sample code for RUM, a single unlearn algorithm is applied over 3 different sequential steps in the order of low memorization to high memorization. The first unlearning algorithm is applied to either the original model that has just been trained on or the model from the previous unlearn step based on the current unlearn step. The final part uses the `seq_mix` unlearn method which simply copies the final step of the process into a single RUM unlearn step without performing any additional unlearning.
With the current implementation, the refinement process and execution order is decided by the user which allows for more flexibility for experimenting with various refinement methods and their orderings.

3.2 Hyperparameter Tuning

As Machine Learning models are very volatile with many different ways to set them up, hyperparameter tuning was performed before running large tests to ensure that results are accurate and fast to calculate. These will be split into three categories: general unlearning, NegGrad+ and SCRUB.

3.2.1 General Unlearning

The hyperparameters included in this section are ones that are not particular to either algorithm being analysed. There include unlearning epochs for unlearning and batch size for training. All other hyperparameters for this step were kept as either previously found good values, such as keeping α at 0.95 for CIFAR-10 and 0.9999 for CIFAR-100 as shown by Kurmanji et al. [16] or default values found in the RUM repository if no other values could be found. These were tested for a single unlearn step to try to get the best result for the beginning of the sequential unlearning process.
For training, there were 3 batch sizes tested, 128, 256 and 512. These were tested to see which one gave the largest test accuracy on the initial trained model. This was tested over 10 different seeds and the average test accuracies can be seen in Figure 3.2.1.

Architecture	Dataset	Batch Size	Time (s)	Train Acc (%)	Test Acc (%)
VGG-16_bn_1th	CIFAR-10	128	514.80±6.89	99.07±0.02	90.83±0.42
		256	510.84±3.45	99.36±0.00	91.35±0.01
		512	500.21±5.12	99.23±0.06	90.93±0.47
	CIFAR-100	128	1685.79±10.98	99.84±0.01	71.84±0.39
		256	1590.51±10.73	99.93±0.01	70.88±0.54
		512	1550.29±6.59	99.96±0.01	69.65±0.27
ResNet-18	CIFAR-10	128	682.80±15.38	99.20±0.05	90.80±0.25
		256	675.99±35.78	99.13±0.08	91.83±0.58
		512	664.06±6.59	98.53±0.12	90.90±1.13
	CIFAR-100	128	2179.37±8.89	99.97±0.00	74.91±0.38
		256	2132.60±14.69	99.97±0.01	74.82±0.50
		512	2128.50±23.55	99.97±0.00	72.92±0.00

Table 3.1: Performance comparison of training with different batch sizes

The time taken varied differently based on the model and architecture. The largest difference being a difference of about 113s in the case of CIFAR-100 using VGG-16. A batch size of 128 performed best in terms of test accuracy for CIFAR-100 while a batch size of 256 was best for CIFAR-10. All had very high train accuracies, showing that differing batch sizes can help with the overfitting of these models.

In the case of unlearning, the main hyperparameter being tested was amount of unlearn epochs. This was tested over 10 seeds and a single unlearn step using precomputed memorization scores from Feldman et Al.[7] for CIFAR-100 and Zhao et Al. [21] for CIFAR-10 with a mixed memorization level. The ToW and ToW-MIA scores for these can be seen in Figure 3.1.

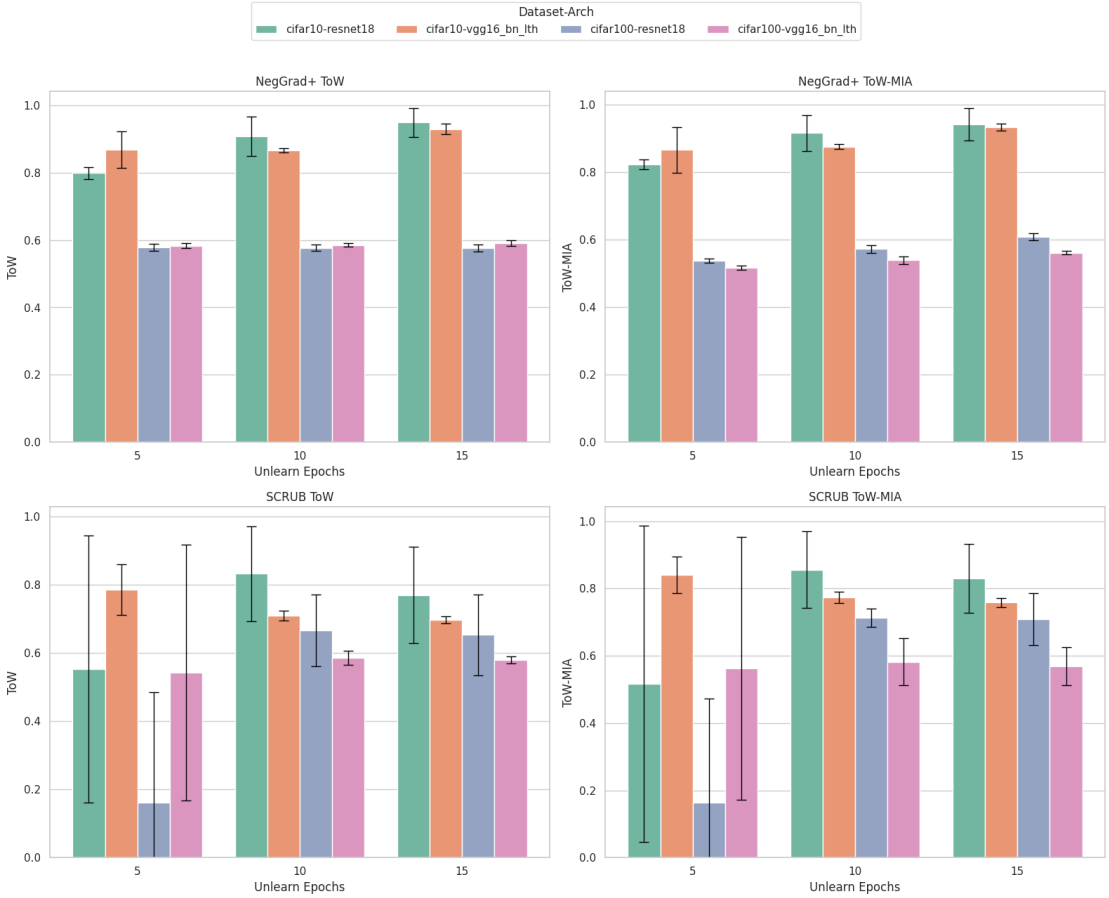

Figure 3.1: ToW and ToW-MIA scores for SCRUB and NegGrad+ across architectures, datasets, and unlearn epochs (5, 10, 15).

In the case of NegGrad+, for CIFAR-10 there is a clear increase in performance with a larger ToW and ToW-MIA value being achieved as unlearn epochs increase with 15 epochs having the best performance across the board. With CIFAR-100, there is very little improvement with ToW as the value stagnates while ToW-MIA increases slightly with unlearn epochs. This signifies that although the accuracy does not change much when increasing the number of unlearn epochs for NegGrad+ with a large dataset, there is an increase in user privacy as with more unlearn epochs, the forget set is harder to distinguish from the retain set. However, the time cost for larger values of unlearn epochs, shown in Table 3.2, is quite significant while the difference in privacy is at most 0.087 for 5 epoch CIFAR-100 VGG-16.

For SCRUB the results tell a slightly different story. Firstly, for 5 unlearn epochs, it

can be seen that the values for ResNet-18 are much lower than the values for VGG-16. However, when unlearn epochs are further increased to 10 and 15, ResNet-18 consistently outperforms VGG-16 in both datasets. The issues caused for 5 unlearn epochs are most likely caused by using 4 max steps which does not allow the algorithm much time to recover the retain set error, making the accuracy of the model relatively low. The large confidence intervals that can be seen in SCRUB further highlight this issue, as without having time to recover the loss, the accuracy is heavily dependant on the actual data that is being forgotten and its influence on the entire dataset. There is also a decrease in performance between 10 and 15 unlearn epochs for both ToW and ToW-MIA, showcasing that focusing too much on recovering the retain set performance negatively affects the accuracy of the model.

The average time taken for each proxy and unlearn step over all architectures and vanilla unlearning algorithms can be seen in Table 3.2. This shows that SCRUB is faster than NegGrad+ in all cases which is unexpected due to it updating weights twice per epoch. The difference in unlearn times between CIFAR-10 and CIFAR-100 is much greater for SCRUB than NegGrad+, showing that SCRUBs performance is impacted much more by the size of the dataset. VGG-16 is much faster than ResNet-18 which is to be expected as it is a smaller architecture, meaning that updating weights is easier as backpropagation is less complicated. Time taken between each epoch amount is also to be expected with 10 epochs taking approximately 2 times longer and 15 epochs taking 3 times longer.

Epoch	Proxy	C10R18		C100R18		C10V16		C100V16	
		NegGrad+	SCRUB	NegGrad+	SCRUB	NegGrad+	SCRUB	NegGrad+	SCRUB
5	max conf	197.576	109.438	195.467	124.316	137.861	85.813	146.043	100.246
	conf	194.784	98.737	194.211	121.090	128.768	82.893	140.584	101.639
	entropy	196.379	99.240	193.624	116.681	126.810	82.036	143.119	99.061
	bi acc	205.123	97.404	193.651	112.477	127.736	80.020	142.491	103.482
	ho ret	198.667	98.410	193.562	117.050	118.187	131.620	144.190	99.802
10	max conf	387.893	183.178	403.924	211.951	250.443	156.521	276.417	182.990
	conf	382.291	182.617	420.865	211.719	249.876	156.651	268.180	205.359
	entropy	375.609	184.471	417.867	217.480	249.687	153.059	267.680	181.961
	bi acc	376.009	184.820	414.553	214.610	252.811	155.093	260.458	182.955
	ho ret	381.117	193.672	412.845	227.805	251.445	157.545	269.282	175.739
15	max conf	572.007	270.526	596.247	312.012	393.822	223.337	419.195	268.060
	conf	593.754	275.325	603.722	327.407	402.448	235.143	422.504	271.250
	entropy	597.986	273.464	603.722	341.134	414.536	234.096	419.866	267.524
	bi acc	585.806	281.328	603.722	339.624	405.160	227.109	401.929	265.998
	ho ret	594.586	279.605	603.722	340.759	409.785	225.784	410.093	255.122

Table 3.2: Unlearning time (in seconds) for each method-architecture combination grouped by proxy and epoch.

3.2.2 NegGrad+

As seen in Equation 2.3, the main hyperparameter that controls NegGrad+ is β. Kurmanji et al. [16], suggest that values in the range $[0.9, 0.999]$ work best for this algorithm, so the main values tested were 0.2, 0.9, 0.95 and 0.9999. The first value is specifically chosen to be out of range to verify the explosive behaviour stated in the literature while the others are used to see what works best from within the range. The results for this can be seen in Figure 3.2

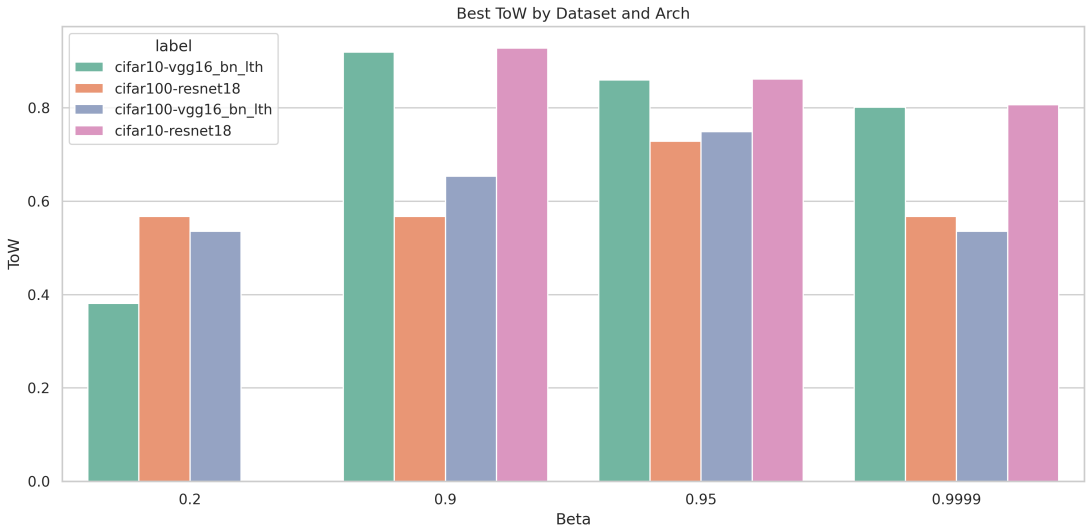

Figure 3.2: ToW scores for various values of α

There are missing values for $\beta = 0.2$ due to small β values causing the loss to explode to negative infinity as seen in Figure 3.3. Aside from that, 0.9 gives best ToW values for CIFAR-10 while 0.95 gives best value for CIFAR-100. Increasing α too much decreases performance, which can be explained by over focusing on the retain set and the forget set having very minimal impact on the actual loss which could lead to a degree of overfitting on the retain set.

```
------------------Get unlearning method: NG------------------
Epoch #0, Learning rate: 0.005
len(r_loader): 329, len(f_loader): 24
Epoch: [0][49/329]      Loss -0.6715 (-0.0822)  Accuracy 96.094 (99.531)       Time 6.84
Epoch: [0][99/329]      Loss -17.2676 (-3.2888) Accuracy 1.562 (63.383) Time 6.11
Epoch: [0][149/329]     Loss -205.0416 (-27.7486)       Accuracy 2.344 (42.766)Time 5.95
Epoch: [0][199/329]     Loss -3894.9058 (-330.5253)     Accuracy 0.781 (32.258)Time 6.07
Epoch: [0][249/329]     Loss -137771.4375 (-7747.3426)  Accuracy 0.000 (26.006)Time 5.96
Epoch: [0][299/329]     Loss -6001138.0000 (-277517.4994)       Accuracy 0.000 (21.841) Time 5.85
train accuracy 20.043
one epoch duration:40.074509143829346
Epoch #1, Learning rate: 0.005
len(r_loader): 329, len(f_loader): 24
Epoch: [1][49/329]      Loss -2542675456.0000 (-669955840.6400) Accuracy 1.562 (0.906)  Time 6.49
Epoch: [1][99/329]      Loss -114626781184.0000 (-15543449316.1600)     Accuracy 2.344 (1.016)  Time 5.75
Epoch: [1][149/329]     Loss -5174152134656.0000 (-470348846614.4000)   Accuracy 0.781 (0.969)  Time 5.64
Epoch: [1][199/329]     Loss -235679149719552.0000 (-16052671019562.4004)       Accuracy 1.562 (1.000)   Time 5.65
Epoch: [1][249/329]     Loss -10567367599849472.0000 (-583261995873977.5000)    Accuracy 0.000 (1.028)   Time 5.78
Epoch: [1][299/329]     Loss -485575396029366272.0000 (-22099088154167748.0000)Accuracy 0.000 (1.026)   Time 5.69
train accuracy 1.019
one epoch duration:38.29421067237854
```

Figure 3.3: Screenshot showcasing NegGrad+ loss go to negative infinity for $\alpha = 0.2$

3.2.3 SCRUB

SCRUB has three main hyperparameters to test for being max steps, α and γ. All of these were tested alongside each other with the following values:

Parameter	Values Tested
Max Step	1, 2, 3, 4, 5, 7
α	0.0005, 0.1, 0.9
γ	0.0005, 0.1, 0.9

Table 3.3: Parameter combinations tested for different values of max step, alpha, and gamma for SCRUB.

The ToW values for these hyperparameters can be seen in Figure 3.4.

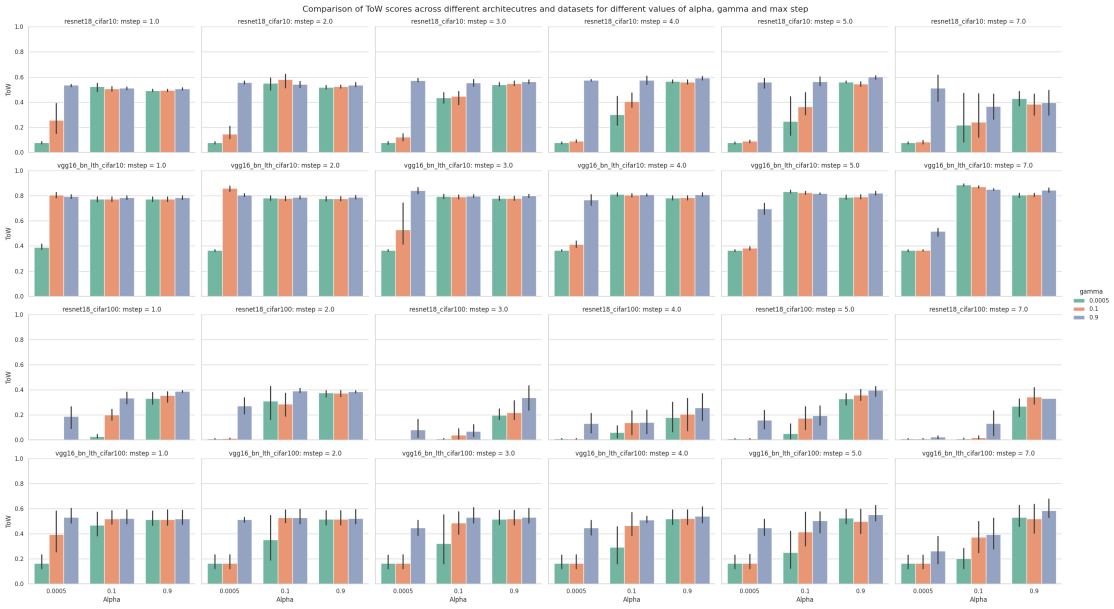

Figure 3.4: ToW values for all variations of max step, alpha and gamma across all datasets and architectures

The first noticeable thing about these results is that the choice of α and γ does affect the unlearning performance, as values that remain at 0.0005 give much lower performance than other values tested, implying that SCRUB is more suitable with larger values of α and γ. This is further supported by Kurmanji et Al. [16] which shows that values between 0.1 and 3 for both alpha and gamma do not experience much difference in

specifically forget error. These results also show that the number of max steps have a varying degree of effect as in the VGG-16, CIFAR-10 example there is little change when both alpha and gamma are high, however in the case of ResNet-18 CIFAR-10, increasing them too much has a very negative impact on performance.
While future results will use the best uncovered values for α and γ, in practice, it would be best to keep both values around 0.9 with 5 max steps. These values consistently perform well across all tests with minimal variation difference from optimal values from all values obtained.
One observation that can be made is that the performance on ResNet-18 and CIFAR-100 is much worse than the results from Figure 3.1. This is due to the hyperparameters not including the exact same values as results from Figure 3.1 used 4 max steps with an α of 0.9 and γ of 1e-5 which was not included in hyperparameter testing for these results. This is an issue of user error as the default value for γ was overlooked when performing hyperparameter testing and time limitations did not allow for the addition of this value. However, this does introduce an issue as performance of SCRUB is unknown for values between 1e-5 and 0.0005, with 1e-5 giving great performance and 0.005 giving very poor performance. This allows for further exploration in this area.

Chapter 4

Experiments and Results

4.1 Experimental setup

Each algorithm and architecture combination was ran for 5 sequential unlearn steps for both their vanilla version and their RUM version, using 5 different seeds. The RUM version consists of using the same algorithm 3 times, once for each memorization level. The other hyperparameters used are ones found via the previous hyperparameter tuning steps and are listed in more detail in Appendix C

4.2 Vanilla

This section will focus on running the base algorithm using a mixed memorization level to emulate forgetting various points in the dataset. As neither algorithm use any memorization in the algorithm itself, the performance of the proxies will not be analysed, but instead the test accuracies of these models will be compared to models that have been retrained from scratch using the same forget set using both explicit test accuracy and ToW and ToW-MIA scores.

4.2.1 NegGrad+

ToW and ToW-MIA values can be seen in Figure 4.1 and Figure 4.2 respectively. Figure 4.1 shows a slightly negative trend between ToW and unlearn step, with the ToW value slowly decreasing as unlearn steps increase. However, there exist instances where the

ToW value increases after an unlearn step has been performed, specifically in the case of VGG-16 and ResNet-18 CIFAR-10 between step 1 and 2. This additional step causes a slight decrese in retain set accuracy as seen in Figure 4.5, however it brings it closer to the retrained model, explaining the increase in ToW. As the steps increase, the difference in retain set accuracy and forget set accuracy between the 2 models increases, resulting in a decreasing ToW score.

Figure 4.1: ToW scores for NegGrad+ over 5 sequential unlearning steps

Figure 4.2: ToW-MIA scores for NegGrad+ over 5 sequential unlearning steps

Figure 4.2 shows a similar trend between unlearn steps and ToW-MIA, having ToW-

MIA slowly decrease as more unlearn steps are performed. This can also be explained by the increasing difference in retain set accuracies between the models. It is also partially due to the MIA scores being vastly different as difference in forget set accuracies will cause a large difference in MIA results.
The 95% confidence intervals, indicated by the black error bars, are quite tight across all experiments, indicating a good level of stability with these algorithms over different seeds. This means that for a mixed level of memorization a similar level of performance can be expected with very little deviation.
There is quite a large difference between the ToW and ToW-MIA scores, with the ToW-MIA values much lower for CIFAR-100 and slightly lower for almost all CIFAR-10 results. This implies that NegGrad+ performs better with staying accurate rather than forgetting user data well. This is further supported by Figure 4.5 and Figure 4.4 which shows that NegGrad+ has very similar performance on the retain set, even outperforming the retrained model for CIFAR-10, however, it fails in effectively forgetting examples from D_f. In the case of CIFAR-100 ResNet-18 it has an almost perfect accuracy of 99.913 ± 0.069 showing that almost no forgetting has occurred. This is most likely a result of choosing a high value for β, which causes the algorithm to focus too much on retain set accuracy, leading to almost no forgetting for a lower amount of unlearn epochs. This may perform better with an increase in unlearn epochs or reduction in β however further testing is required to come to more concrete conclusions.
In terms of test accuracy, Figure 4.3 shows that NegGrad+ remains very close to the retrained model, maintaining a slightly higher test accuracy throughout. This makes vanilla NegGrad+ a good choice for maintaining good performance after forgetting at the cost of the foretting not being very thorough.

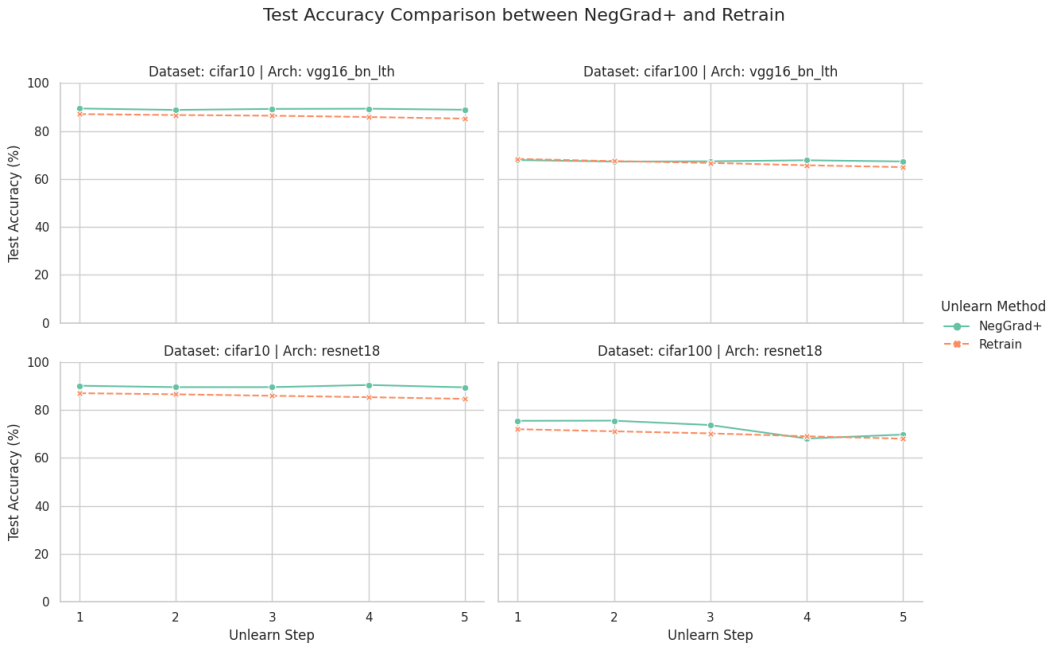

Figure 4.3: Test Accuracy comparison between NegGrad+ and Retrain

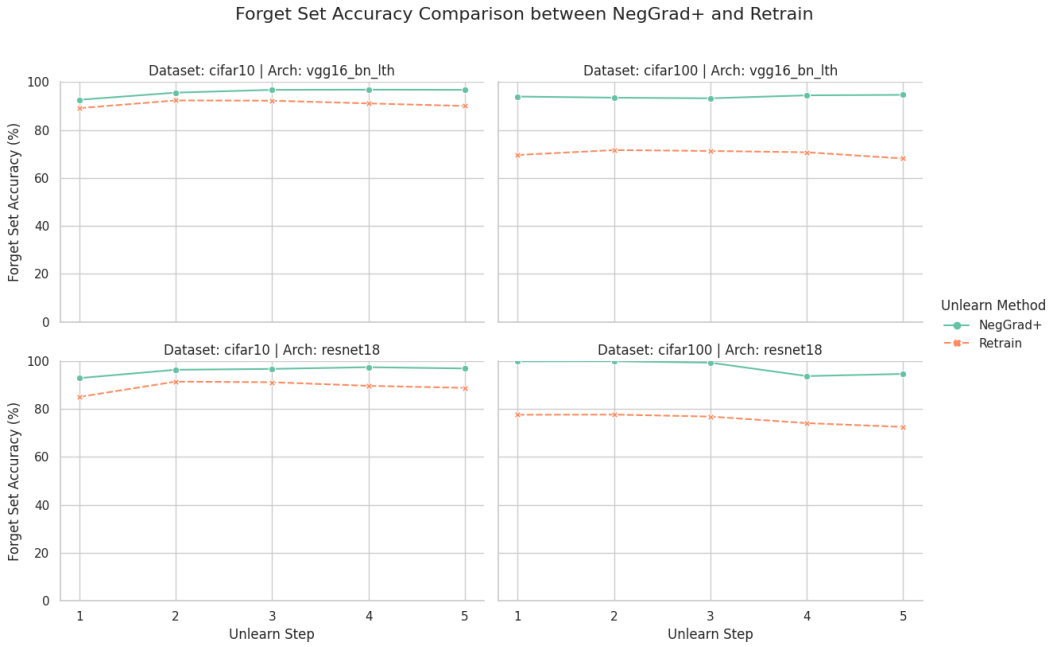

Figure 4.4: Forget Set Accuracy comparison between NegGrad+ and Retrain

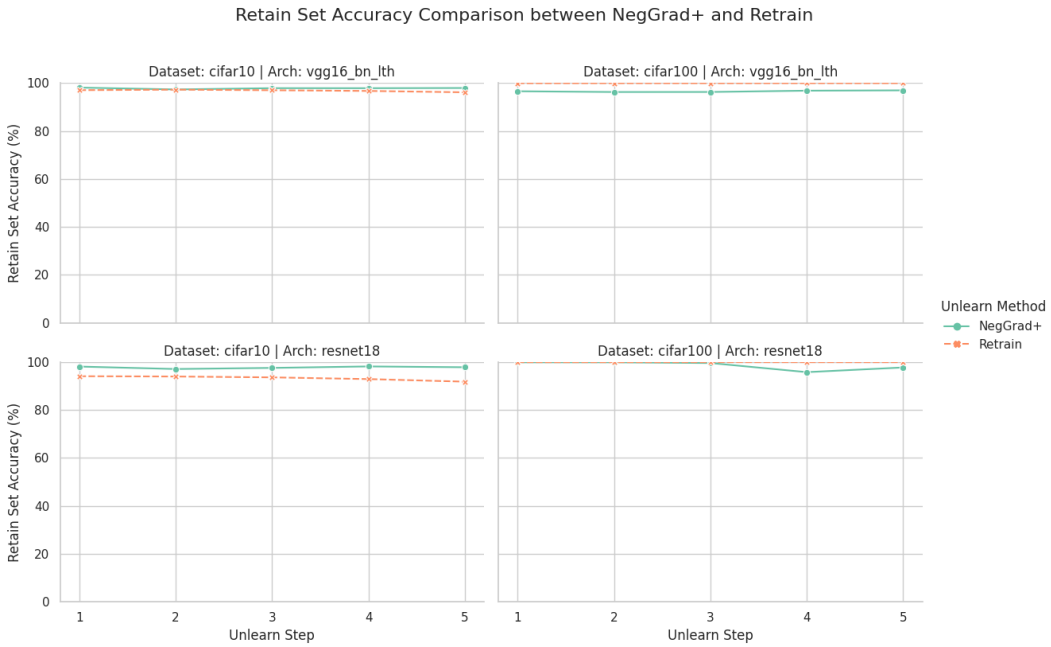

Figure 4.5: Retain Set Accuracy comparison between NegGrad+ and Retrain

4.2.2 SCRUB

There is very little difference across unlearning steps for SCRUB. All ToW and ToW-MIA values stay consistent being, very similar to eachother with a maximal difference in ToW of 0.007 between step 1 and 2 for ResNet-18 CIFAR-10. For ToW specifically, ResNet-18 outperforms VGG-16 as expected from Figure 3.1, while the values experience much more consistency and higher peaks due to the additional hyperparameter optimisation.
ToW-MIA is slightly different, having much better performance for models trained on CIFAR-100 compared to CIFAR-10, indicating that there is a privacy benefit when unlearning using SCRUB for models trained on larger.

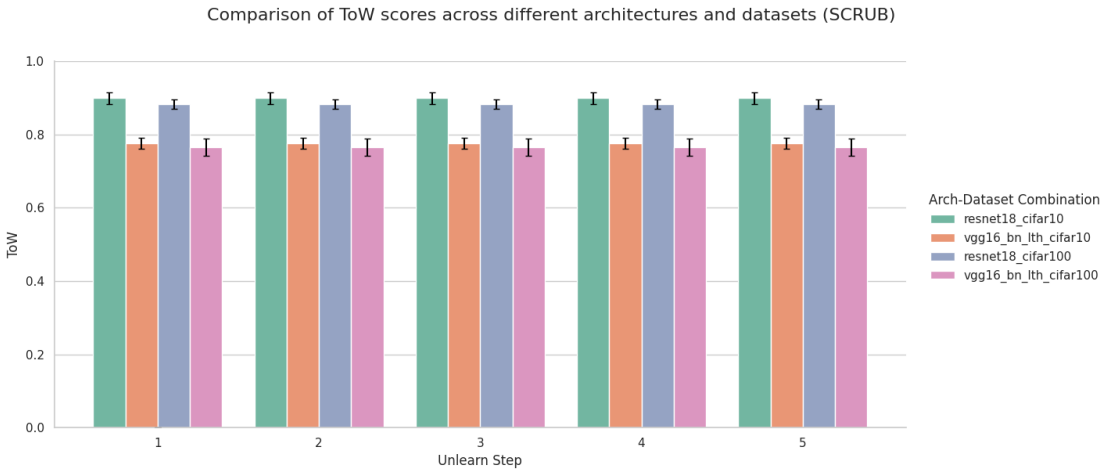

Figure 4.6: ToW scores for SCRUB over 5 sequential unlearning steps

Figure 4.7: ToW-MIA scores for SCRUB over 5 sequential unlearning steps

The test, retain set and forget set accuracies provide a much clearer picture of SCRUB's success. Figure 4.8 shows that test accuracies for SCRUB are consistently above the test accuracies of the model that was retrained from scratch. Combining this with the retain set accuracies ,seen in Figure 4.9, being either better or equal to retraining and the much lower forget set accuracies, seen in Figure 4.10, SCRUB seems like a major improvement over traditional retraining. These differences could be explained by differing levels of overfitting caused by retraining and SCRUB. The retrained model maintains a retain set accuracy between 95% and 100% based on the dataset, however

SCRUB is much closer to 100% over all datasets and architectures. This could suggest that SCRUB is overfitting the retain set while underfitting the forget set to achieve results that are better than traditional retraining. This result can be classed as "benign overfitting" [3], where certain conditions make overfitting training data not necessarily detrimental to test performance. This may work on these datasets, however it is not something that should be relied on.

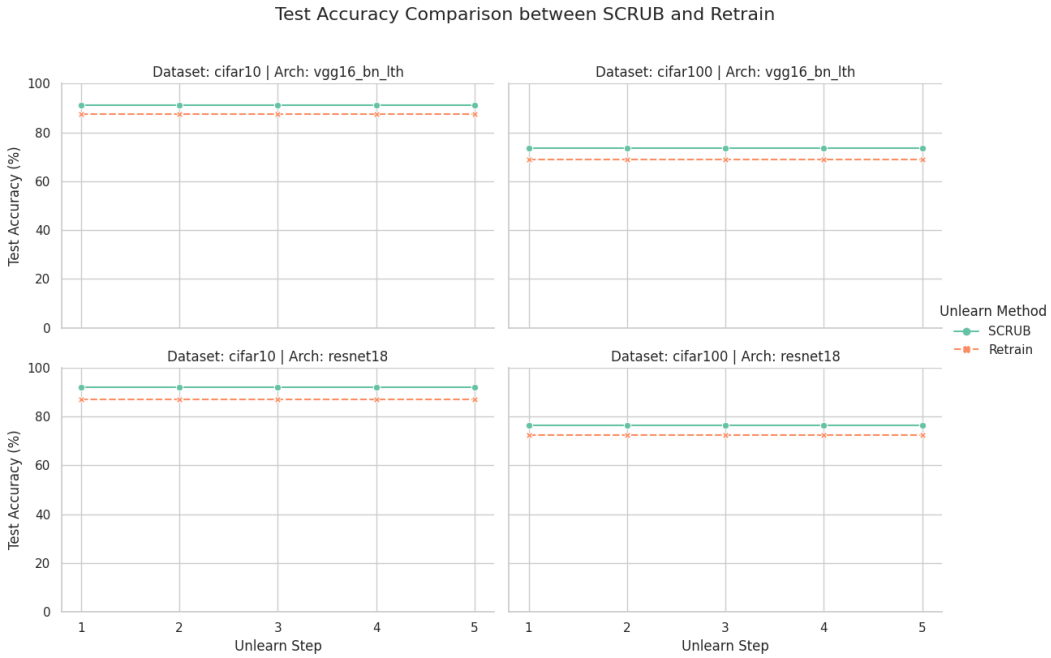

Figure 4.8: Test Accuracy comparison between SCRUB and Retrain

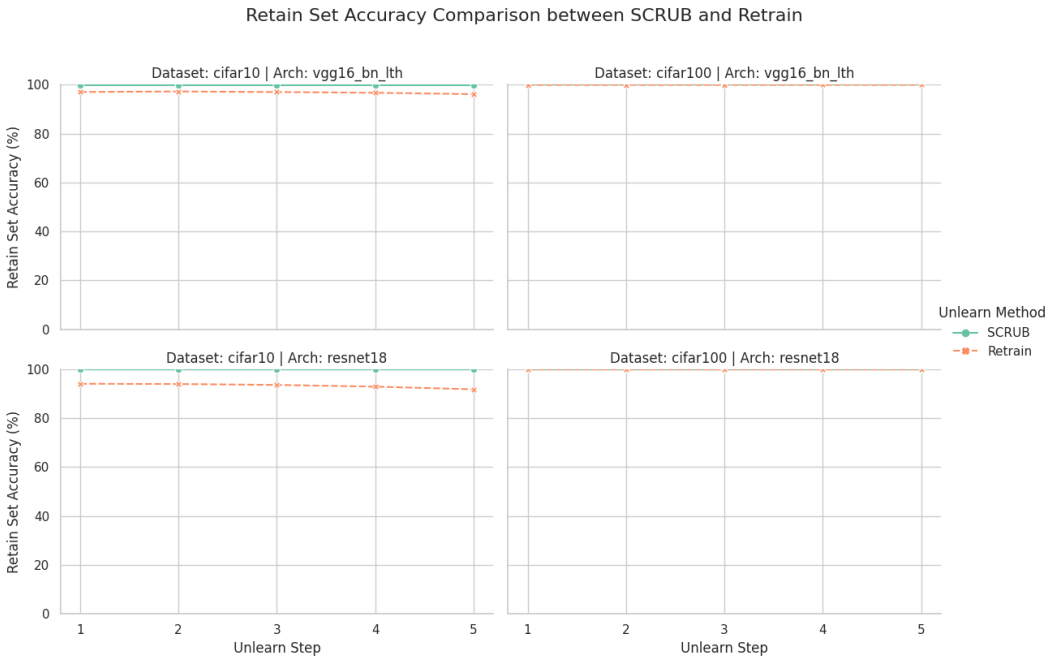

Figure 4.9: Retain Set Accuracy comparison between SCRUB and Retrain

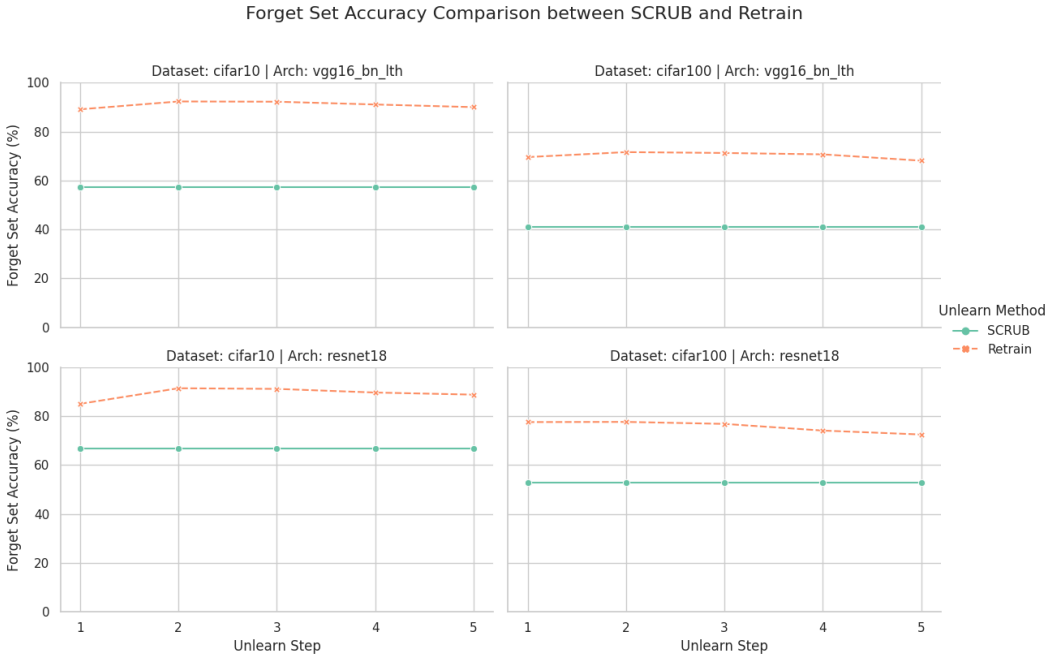

Figure 4.10: Forget Set Accuracy comparison between SCRUB and Retrain

4.3 RUM

This section will focus on running the RUM version of the algorithms. This will consist of splitting the forget set into 3 subsets of size 1000, separated by memorization levels of low, medium and high which are determined by the various memorization proxies mentioned previously. They will be ran in the order of increasing memorization with each proxy being evaluated for stability and the algorithms themselves for performance. RUM NegGrad+ will refer to a version of RUM that uses NegGrad+ for all 3 subsets, while RUM SCRUB will refer to a version of RUM that uses SCRUB for all 3 subsets.

4.3.1 NegGrad+

Figure 4.11 shows the ToW values over all 5 sequential unlearning steps for both CIFAR-10 and CIFAR-100 using ResNet-18 and VGG-16. Step 1 for both VGG-16 models has decent performance with most ToW values being around 0.8 for CIFAR-10 and 0.65 for CIFAR-100 with holdout retraining performing noticeably better. As the learning steps increase, the ToW values decreases sharply for almost all proxies for both CIFAR-100 and CIFAR-10, with the only exception being binary accuracy for CIFAR-10. However, the step 5 binary accuracy value has a very large confidence interval meaning it is not a stable nor reliable result.
For ResNet-18, max confidence, confidence and entropy all remain as poorly performing proxies, with step 5 of CIFAR-100 being an exception with entropy outperforming the other proxies. In the Case of CIFAR-10, binary accuracy again, remains about the same, with holdout retraining declining consistently as steps increase.
ToW-MIA values from Figure 4.12 show a similar trend with one main difference being the large increase in ToW-MIA values for the learning event proxies in the first step.
The results of CIFAR-10 ResNet-18 can be compared to the results obtained by Zhao et Al. [23] where the RUM verison of NegGrad+, using the same refinement method and execution order, was analysed for 5 unlearn steps. These values can be seen in table A.1. The values for ToW and ToW-MIA are higher than the ones obtained here however, not too far away from the holdout retraining scores. The main difference being confidence intervals are much larger for this work. However, this could have been an issue of hyperparameter choices or initial model differences.

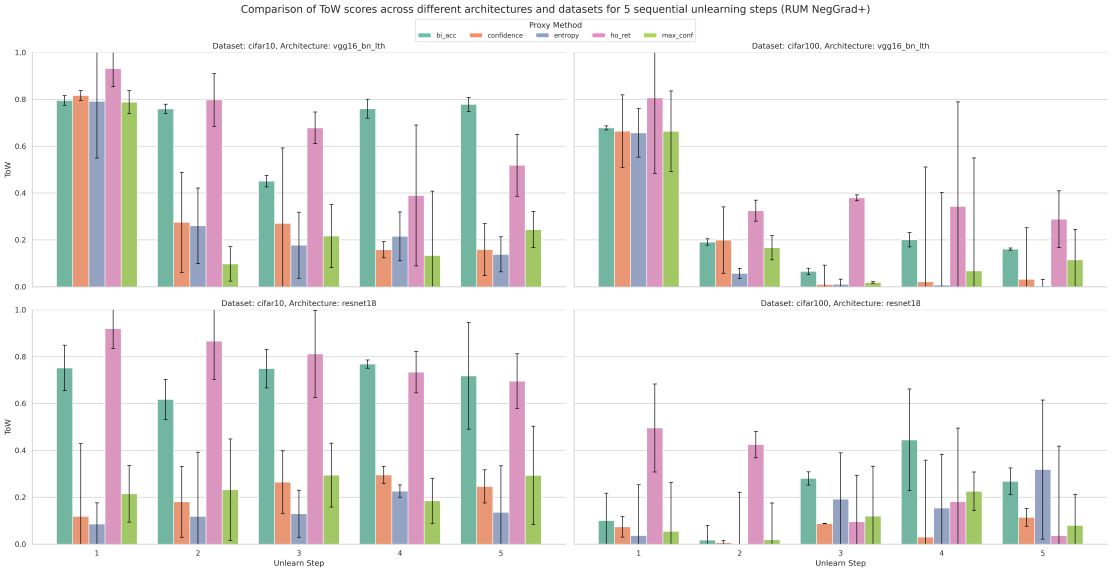

Figure 4.11: ToW scores for RUM NegGrad+ over 5 sequential unlearning steps for 5 memorization proxies

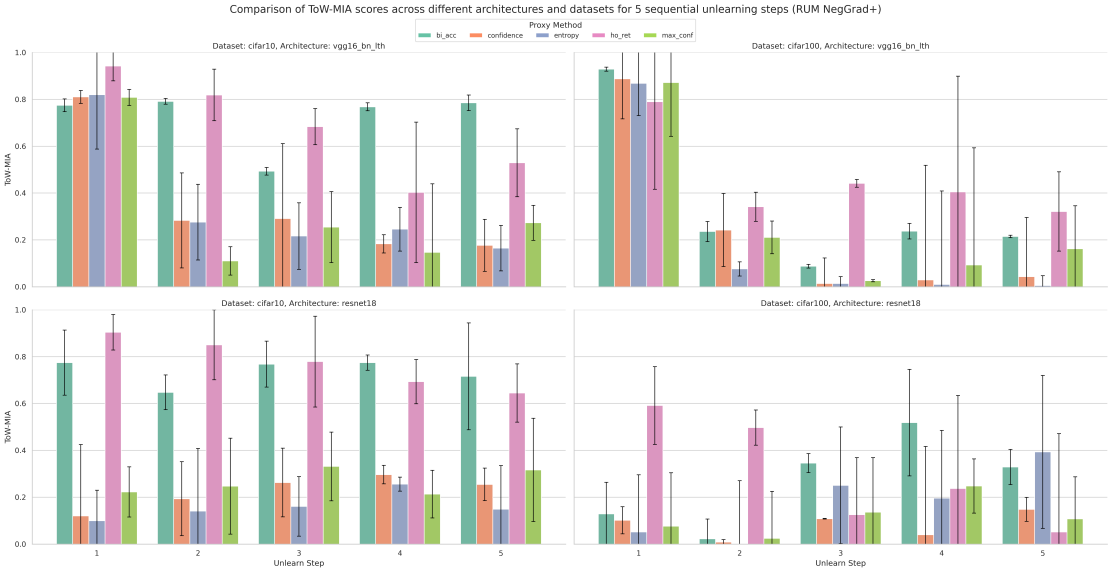

Figure 4.12: ToW-MIA scores for RUM NegGrad+ over 5 sequential unlearning steps for 5 memorization proxies

Looking at test accuracies in Figure 4.13, VGG-16 experiences a drop in accuracy when comparing the first step to the last with some proxies experiencing small improve-

ments for certain steps. For ResNet-18, the accuracies remain similar for the CIFAR-10 dataset while increasing and decreasing, seemingly at random for the CIFAR-100 dataset. These patterns can also be seen in the retain set accuracies, Figure 4.15, and forget set accuracies, Figure 4.4. These indicate some level of worsening performance when using RUM NegGrad+ alongside these proxies as there is rarely any increase in performance after the first unlearn step.

These sharp declines could be caused by the loss growing or decreasing very quickly as it is common for the loss of NegGrad+ to explode when given suboptimal values of β. Another reason could be the low unlearn epochs, having only 5 unlearn epochs per substep of RUM which made the efficacy of the algorithm much worse.

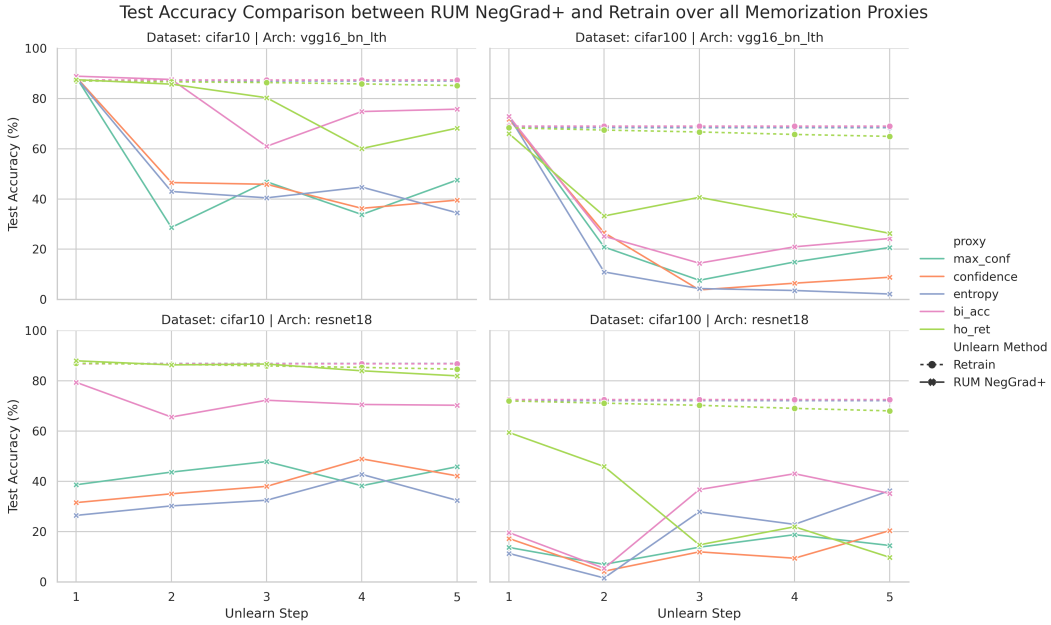

Figure 4.13: Test accuracy comparison for all proxies between RUM NegGrad+ and Retrain

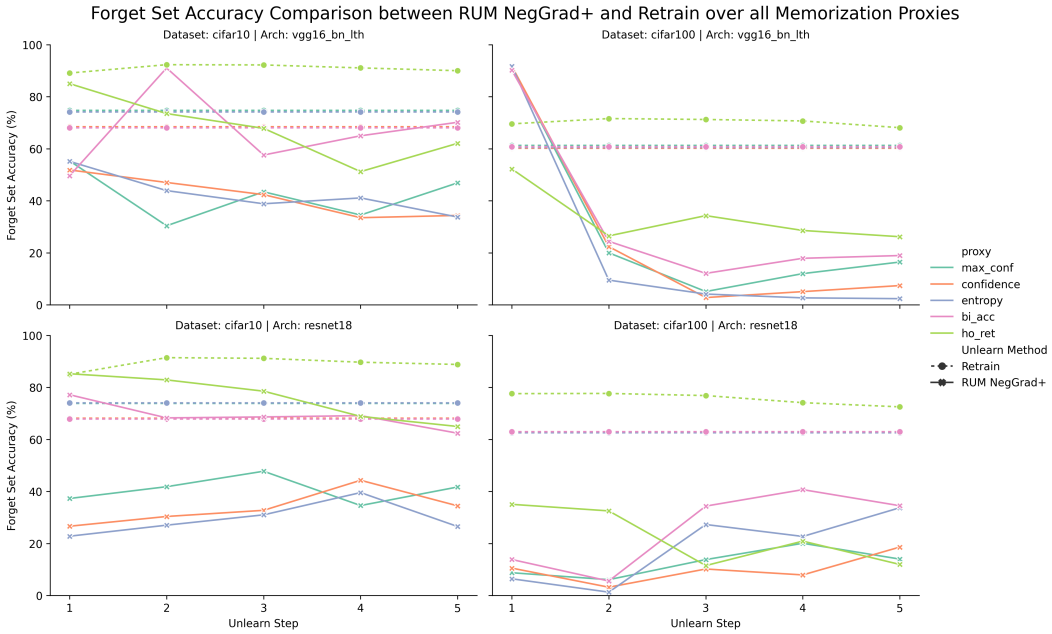

Figure 4.14: Forget Set accuracy comparison for all proxies between RUM NegGrad+ and Retrain

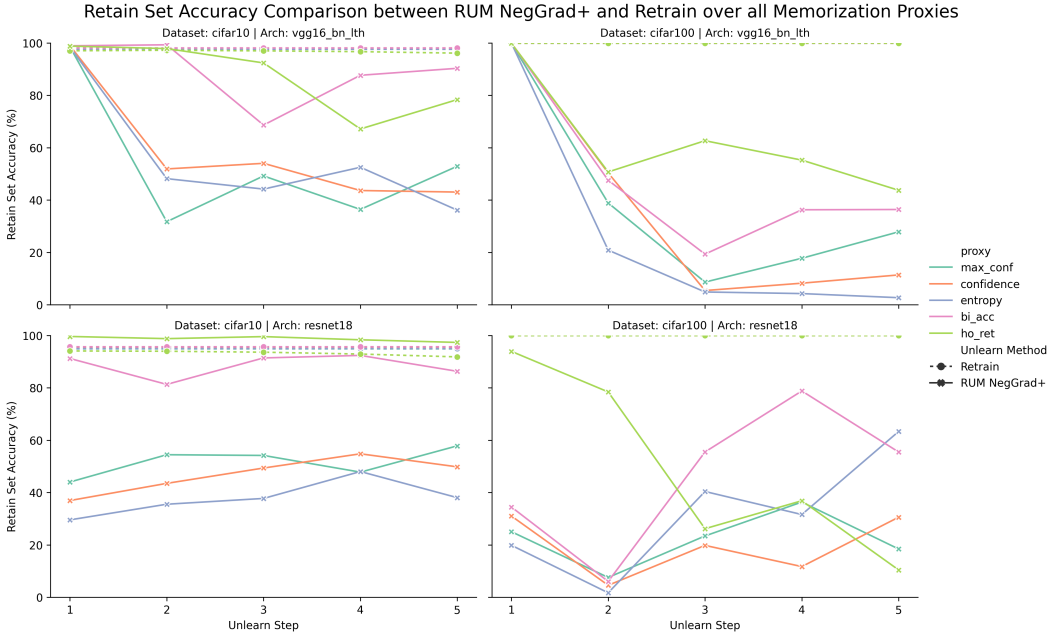

Figure 4.15: Retain Set accuracy comparison for all proxies between RUM NegGrad+ and Retrain

4.3.2 SCRUB

Values for ToW and ToW-MIA across the 5 unlearning steps can be seen in Figure 4.16 and Figure 4.17 respectively. For ToW, holdout retraining is the best performing proxy, scoring the highest ToW score with the only exception being VGG-16 CIFAR-10. Although holdout retraining has the largest ToW value across the other 3 experiments, its confidence intervals are often very large indicating some level of instability or inconsistency. On the other hand, max confidence, while not having the largest ToW value, consistently has tight confidence intervals indicating a strong level of stability and consistency.

The ToW scores slowly increase with unlearn steps with the only exception being VGG-16 CIFAR-10 which is the only case where ToW decreases, and it does so at a very fast rate. This could be an issue of using incorrect hyperparameters for this single test as the trend is not repeated elsewhere.

For ToW-MIA, holdout retraining loses it's spot as the best performing proxy, with the other 4 performing well throughout. This indicates that using proxies other than holdout retraining will have a greater benefit for privacy. Max confidence again, has the tightest confidence intervals making it the most consistent.

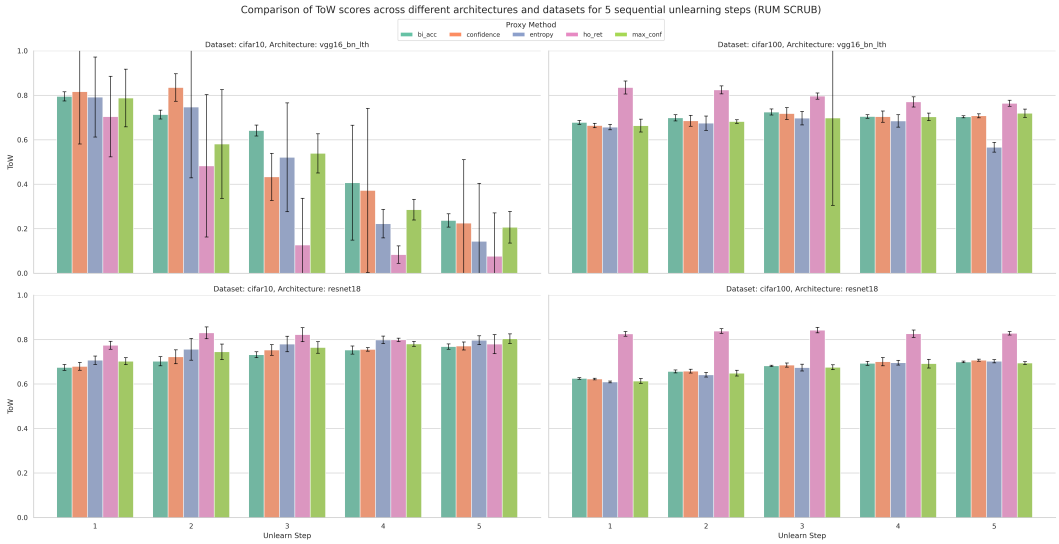

Figure 4.16: ToW scores for RUM SCRUB over 5 sequential unlearning steps for 5 memorization proxies

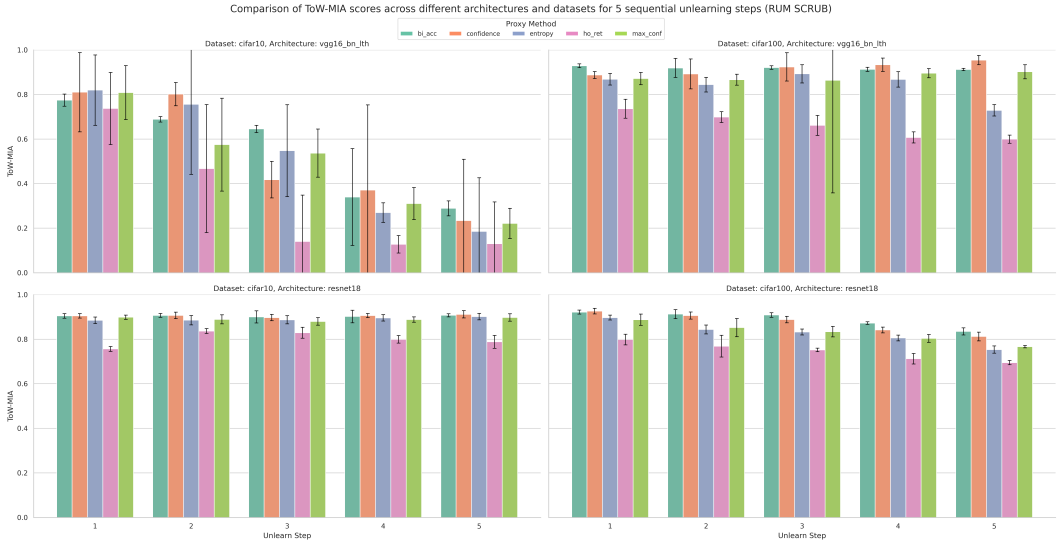

Figure 4.17: ToW scores for RUM SCRUB over 5 sequential unlearning steps for 5 memorization proxies

Test accuracy, seen in Figure 4.18, follows a similar trend to the vanilla version of SCRUB, mostly staying around or above the retrained model. The only exception, again, being VGG-16 CIFAR-10 which has the test accuracy decline over unlearn steps. Forget set accuracy however, seen in Figure 4.19, is larger than the retrained model, indicating that the RUM version is less effective at forgetting compared to its vanilla counterpart. The retain set performance, seen in Figure 4.20, is either the same or slightly better than the retrained model with VGG-16 CIFAR-10 being the exception.

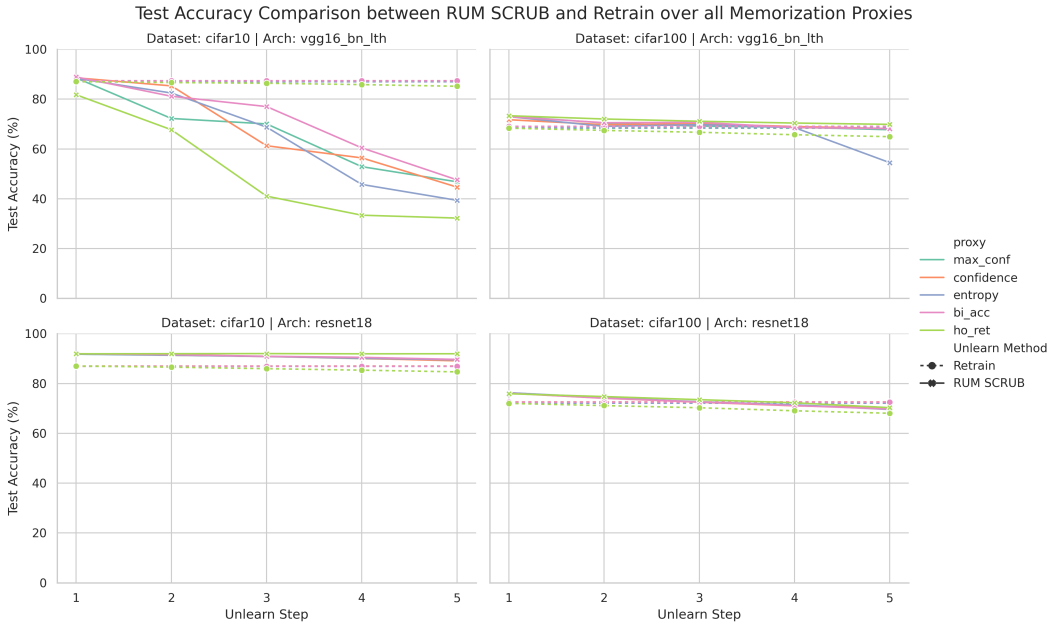

Figure 4.18: Test accuracy comparison for all proxies between RUM SCRUB and Retrain

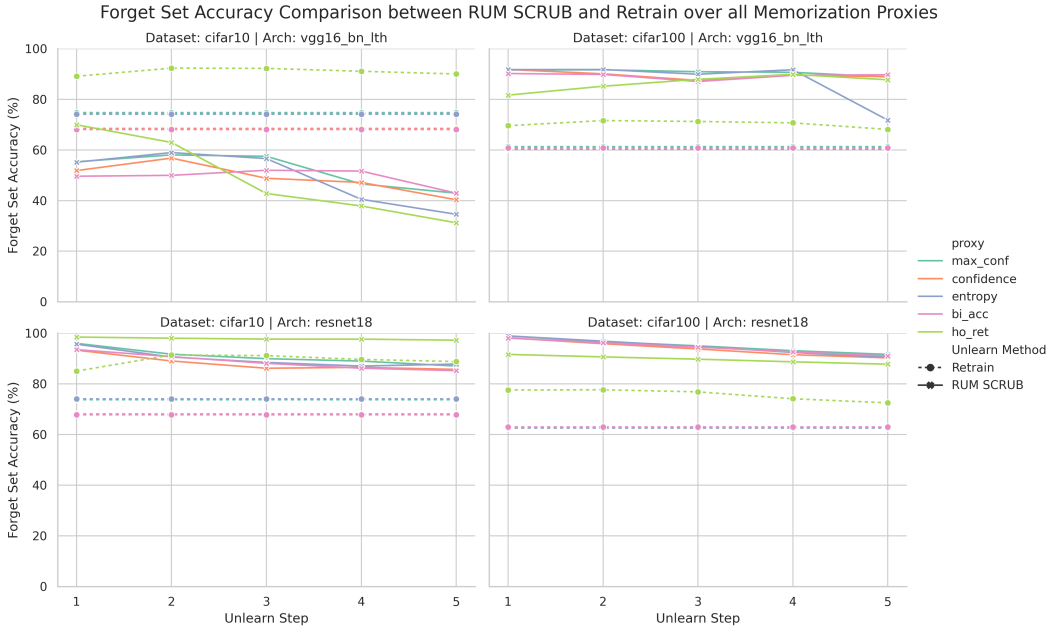

Figure 4.19: Forget Set accuracy comparison for all proxies between RUM SCRUB and Retrain

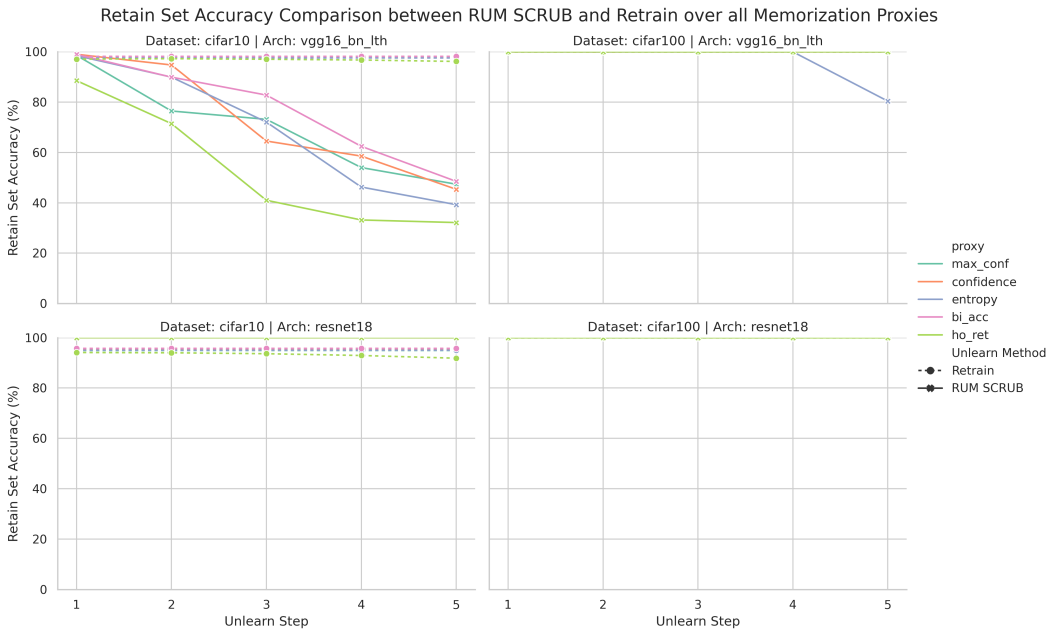

Figure 4.20: Retain Set accuracy comparison for all proxies between RUM SCRUB and Retrain

4.4 Algorithm Comparison

To compare performance between these algorithms, they have all been plotted against each other for all previously used metrics, being, ToW (Figure 4.21, ToW-MIA (Figure 4.22), test accuracy (Figure 4.23), retain accuracy (Figure 4.24) and forget accuracy (Figure 4.25). For simplicity and readability, only the best performing proxy was used for both RUM algorithms being binary accuracy in the case of NegGrad+ and max confidence in the case of SCRUB. Retrain is used as a baseline for comparisons with its ToW and ToW-MIA scores being 1 due to the nature of ToW calculations.

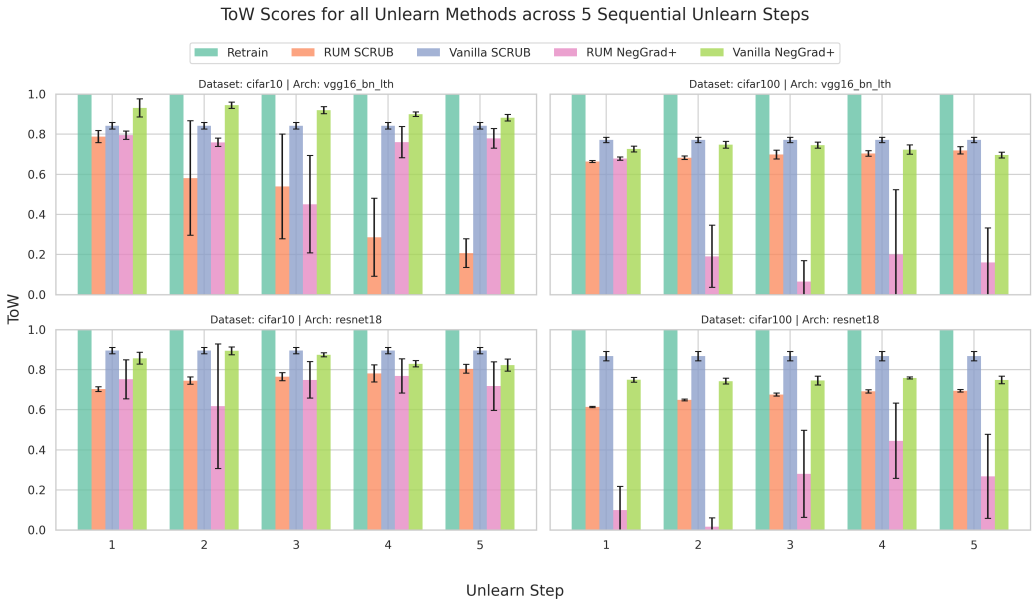

Figure 4.21: ToW Scores for all tested unlearning methods over 5 sequential unlearning steps

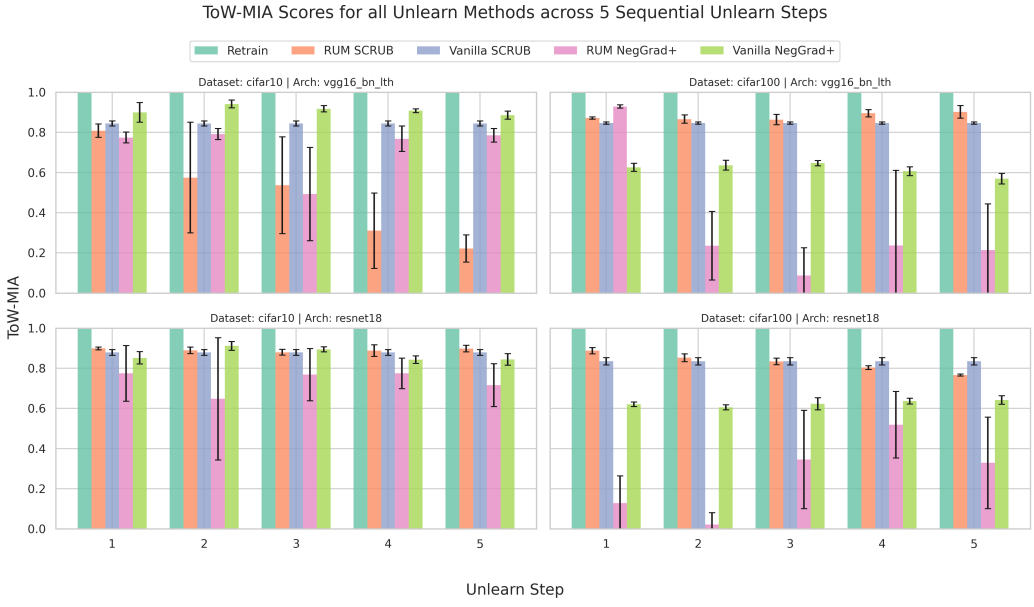

Figure 4.22: ToW-MIA Scores for all tested unlearning methods over 5 sequential unlearning steps

Comparing ToW scores between all methods, the vanilla version of both algorithms outperform their RUM versions. This was not expected as previous work from Zhao et Al. [23] show that RUM versions often outperform their vanilla versions. This could have been caused by incorrect configuration of RUM or hyperparameter issues. The vanilla results however are still very strong and show that the algorithms on their own can provide great value in unlearning.
ToW-MIA values are similar to ToW throughout with vanilla NegGrad+ performing the best for CIFAR-10 VGG-16 and both versions of SCRUB performing well in all other experiments. Vanilla SCRUB appears to be the most consistent in terms of high ToW-MIA scores making it a great choice for privacy focused unlearning compared to NegGrad+.

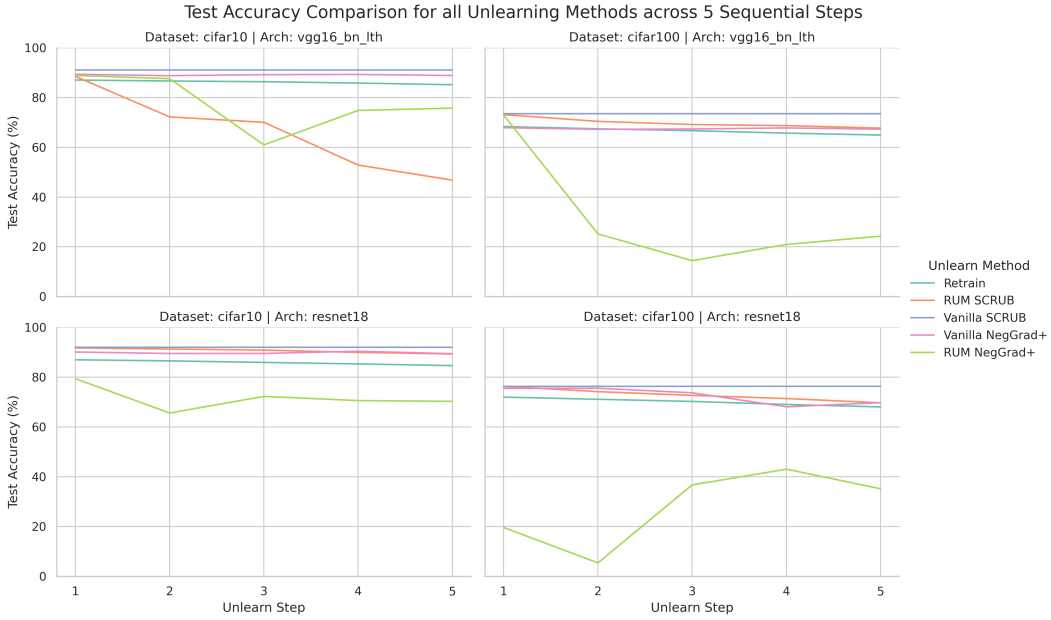

Figure 4.23: Test accuracy comparison between all unlearning methods

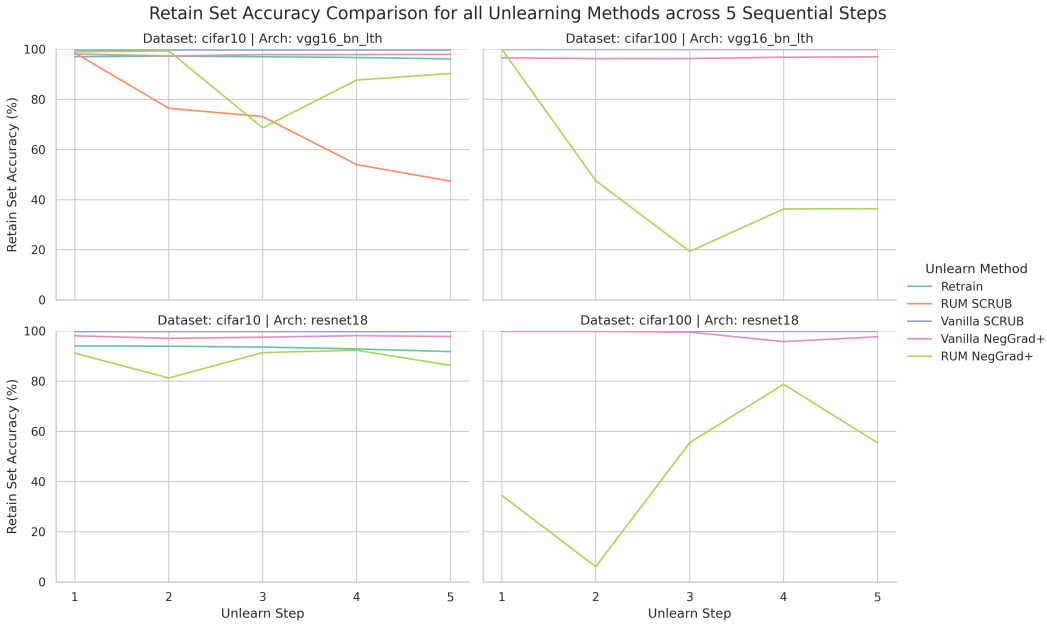

Figure 4.24: Retain set accuracy comparison between all unlearning methods

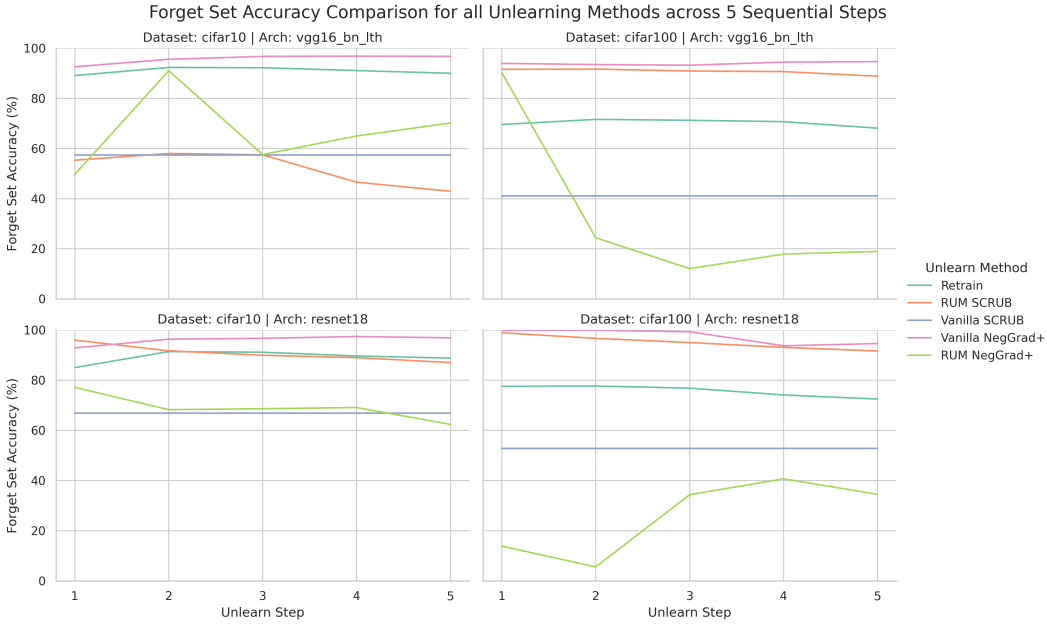

Figure 4.25: Forget set accuracy comparison between all unlearning methods

In terms of test accuracy (Figure 4.23), most algorithms perform similarly with little

deviation with the major outlier being RUM NegGrad+, most likley caused by previously mentioned issues. Retain set accuracy (Figure 4.24) shows a similar trend with RUM SCRUB being outperformed by all other algorithms for CIFAR-10 VGG-16. For the forget set (Figure 4.25), the only algorithm outperforming retrain is vanilla SCRUB while the others retain relatively high forget set accuracies.
These all point to SCRUB being a very strong algorithm from both a privacy and performance perspective, performing very well across all 5 unlearn steps and outperforming NegGrad+ in almost every test.

Chapter 5

Project Management and Evaluation

5.1 Project Management

Due to this project being heavily research focused, Gantt charts were used to give a rough outline of the amount of work that should be completed at a given time. These were often updated as the understanding and scope of the project developed. The initial Gantt chart in Figure B.1 was a good starting point, with much time spent on researching machine unlearning concepts and trying to understand the workings of the algorithms. The time dedicated to development shifted into being focused on understanding the RUM codebase [21] as it became available after the project specification deadline. Due to the initial limited knowledge about the project, it did not include any details about how to move forward.
The second Gantt chart that was being used during the end of Term 1 can be seen in Figure B.2. This was much more detailed with clearer understandings of how to move forward and what should be done at a given time. Most times, the results were overestimated to account for any errors or setbacks that could happen if any errors with the results were found. As the project progressed smoothly at this point and was ahead of schedule, additional objectives were added in Term 2 to increase the scope of the project. These included the addition of obtaining results for the RUM versions of each algorithm, obtaining results for the Tiny ImageNet dataset [17] and comparing sequential unlearning results to single batch unlearning, where the entire sequence is forgotten in one step. Due to time constraints, results could not be obtained for Tiny ImageNet and single batch unlearning.

5.2 Legal, Social, Ethical and Professional Issues

The code, alongside the datasets used are all open source with proper citations used to avoid any professional and ethical issues. The datasets used do not contain any personal or identifiable information. An ethical consideration that was made is the energy consumption caused by the long running machine unlearning algorithms. To gather this amount of data, the batch system was almost constantly in use since the beginning of Term 2, however obtaining these results can help further the field of machine unlearning and remove the need for retraining from scratch which takes much longer and ends up consuming more energy. There are no legal or social issues with this project.

5.3 Evaluation

Based on the objectives outlined in the introduction, the project has been successful. Meaningful results have been obtained for both SCRUB and NegGrad+ alongside their RUM versions. Although some results stray from expectation such as the RUM version of NegGrad+, the results still provide great insights into machine unlearning and its uses in large datasets.
From the additional objectives added in Term 2, results for Tiny Imagenet were unfortunately not possible to obtain given the time constraints of the project and the large time investment needed to train the models and apply the unlearning algorithms.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

Overall both NegGrad+ and SCRUB are great unlearning algorithms that balance effective unlearning and retaining accuracy. From these experiments, SCRUB stands above NegGrad+, consistently outperforming it. The RUM versions of both algorithms seem very promising with RUM SCRUB performing very well. The main issue currently with RUM is the difficulty in setting it up correctly so unlearning can be performed successfully. This leads to some less than ideal results like the ones from this work, however, it has been shown to work well.
ToW and ToW-MIA are great metrics to use for evaluating unlearning performance but care must be taken as they only compare to a model retrained from scratch, however, it is possible for an unlearned model to outperform a retrained model, especially if the retraining is not done optimally. This means that other metrics need to be used alongside them to have a complete picture of unlearning performance.
For someone trying to use these algorithms, many things must be taken into account. If there is a lot of time available and the best unlearning performance is being sought after, RUM versions of both NegGrad+ and SCRUB can be considered due to their high peaks but a lot of work needs to be put in beforehand to ensure that it works well on a specific dataset. If there is very limited time, vanilla NegGrad+ is a good option, having only 1 main hyperparameter that controls performance. Vanilla SCRUB is a nice ground having very strong performance under the right conditions, for which some time has to be dedicated to achieve.

6.2 Limitations

The main limitation for this project was the strict time constraint. This greatly limited the amount of experiments that could be ran and the amount of results that could be obtained. This time limitation also affected the ability to get the most out of the unlearning algorithms. Hyperparameter optimization was done in a suboptimal way, only testing some hyperparameters with eachother and also ignoring a lot of others.
Another limitation was the availability of the DCS batch compute system which the project heavily relied on. There were times when all CUDA capable nodes were in use, and no progress could be made.
The time required for each experiment can be seen in Table D.1

6.3 Future work

This work can be further expanded upon by testing these algorithms on many more unique datasets and architectures. Currently all work that has been done has mainly used CIFAR-10, CIFAR-100 and Tiny Imagenet [23] [22] [16] [9]. However these are datasets where the class sizes are initially all the same so it would be useful to test for datasets where the class sizes are not the same. Another interesting area that could be explored is using these algorithms on non-image datasets. This work has recently been started for large language models [14] with promising results.
One interesting application of these unlearning algorithms is in the case of class removal. They could be used to remove an entire class of data from a dataset and the impact could be analysed. This can be done on classes of different size and memorization to see how it affects the model.
Going forward with sequential unlearning, the performance of unlearning sequentially can be tested against unlearning everything in one go. This will give valuable insights into how often data should be unlearned in order to maintain optimal performance of a model. If unlearning singular data points becomes fast and accurate, it becomes much easier to comply with privacy regulations such as GDPR as Right to Erasure requests can be handled as soon as they are received.
Although both algorithms are good, they can definitely be improved upon, especially SCRUB, with it having 2 weight updates for a single epoch. Finding a way to do both updates simultaneously could heavily improve runtime and potentially performance of the algorithm. For NegGrad+, it is much slower than SCRUB even though only 1

weight update is needed. In this case maybe changing the loss function or even splitting the work into 2 weight updates like SCRUB has could make it run slightly faster. With more knowledge about the possibilities and limitations of these algorithms, they creep closer to widespread usage. These have the chance to greatly increase efficiency across all machine learning models as well as greatly aid user privacy with quick and efficient deletion making this an area that will hopefully continue to grow further.

Chapter 7

Acknowledgements

I would like to thank Peter Triantafillou for introducing me to the topic of Machine Unlearning and helping me with understanding concepts throughout the year. I would also like to thank Kairan Zhao for providing me with the RUM codebase and helping me understand how to use it effectively.

Bibliography

[1] Machine learning statistics 2024. Accessed: 17/11/2024.

[2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024.

[3] Yuan Cao, Zixiang Chen, Mikhail Belkin, and Quanquan Gu. Benign overfitting in two-layer convolutional neural networks, 2022.

[4] Nicholas Carlini, Ãlfar Erlingsson, and Nicolas Papernot. Distribution density, tails, and outliers in machine learning: Metrics and applications, 2019.

[5] Imen Chebbi. *VGG16: Visual Generation of Relevant Natural Language Questions from Radiology Images*. 12 2021.

[6] Vitaly Feldman. Does learning require memorization? a short tale about a long tail, 2021.

[7] Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation, 2020.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[9] Shashwat Goel, Ameya Prabhu, Philip Torr, Ponnurangam Kumaraguru, and Amartya Sanyal. Corrective machine unlearning, 2024.

[10] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks, 2020.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[12] GDPR Info. Article 17 gdpr - right to erasure (âright to be forgottenâ), 2024. Accessed: 17/11/2024.

[13] Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C. Mozer. Characterizing structural regularities of labeled data in overparameterized models, 2021.

[14] Dominik Kowieski. Large-scale machine unlearning in transformer-based pretrained language models: Evaluation and adaptation of existing machine unlearning frameworks kga and scrub. 2024.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[16] Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning, 2023.

[17] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015.

[18] Poorya Mohammadinasab. Pneumonia detection using deep convolutional neural networks, 09 2023.

[19] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models, 2017.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[21] Kairan Zhao. Rum: A repository for unlearning, 2024. Accessed: 24/11/2024.

[22] Kairan Zhao, Meghdad Kurmanji, George-Octavian BÄrbulescu, Eleni Triantafillou, and Peter Triantafillou. What makes unlearning hard and what to do about it, 2024.

[23] Kairan Zhao and Peter Triantafillou. Scalability of memorization-based machine unlearning, 2024.

[24] Pinlong Zhao, Weiyao Zhu, Pengfei Jiao, Di Gao, and Ou Wu. Data poisoning in deep learning: A survey, 2025.

Appendix A

Additional Results

This section contains additional results, including results obtained from other papers that were used for comparisons as well as previous results that performed much worse.

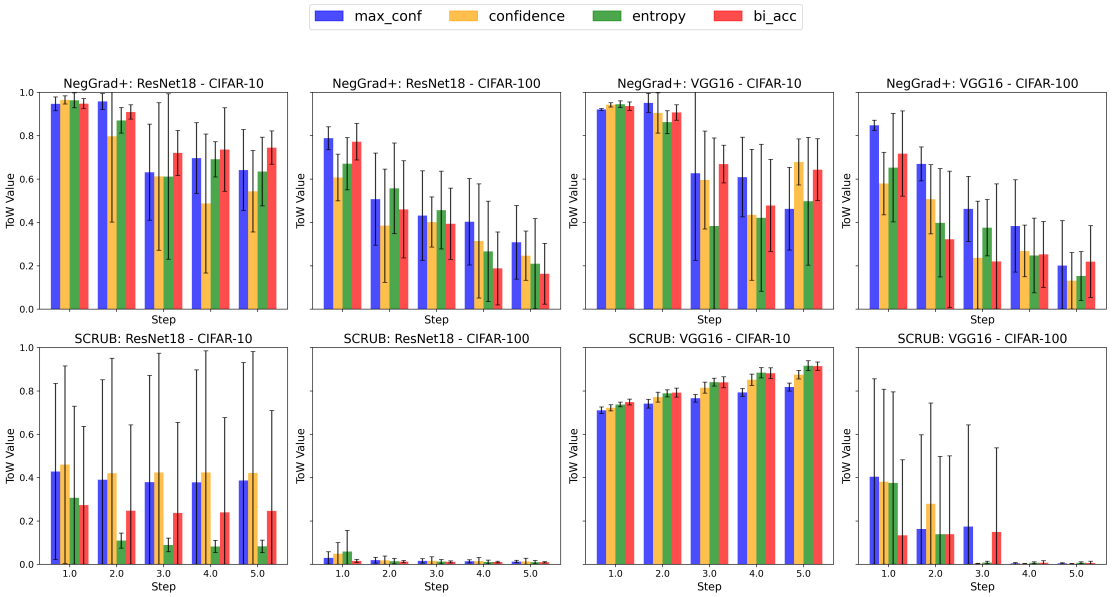

Figure A.1: ToW scores over learning event proxies for both vanilla SCRUB and vanilla NegGrad+

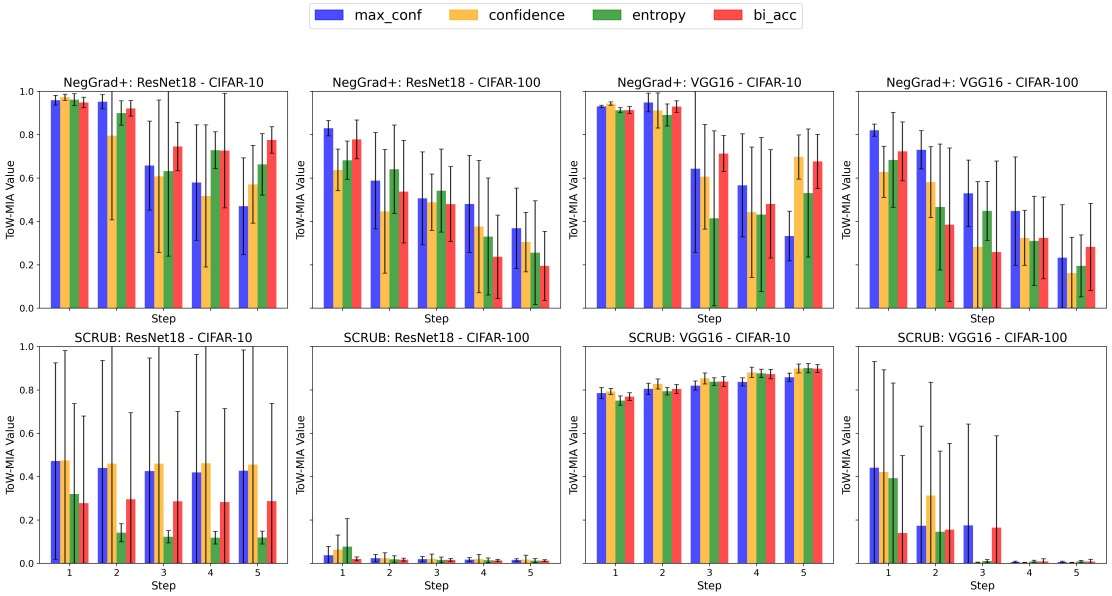

Figure A.2: ToW-MIA scores over learning event proxies for both vanilla SCRUB and vanilla NegGrad+

	NegGrad+ RUM$^{\mathcal{F}}$			NegGrad+ vanilla			Retrain		
	ToW (↑)	ToW-MIA (↑)	Runtime (s)	ToW (↑)	ToW-MIA (↑)	Runtime (s)	ToW (↑)	ToW-MIA (↑)	Runtime (s)
Step 1	0.901 ± 0.035	0.791 ± 0.046	372.202	0.771 ± 0.079	0.661 ± 0.047	140.775	1.000 ± 0.000	1.000 ± 0.000	378.786
Step 2	0.883 ± 0.029	0.737 ± 0.082	341.856	0.825 ± 0.031	0.691 ± 0.073	124.151	1.000 ± 0.000	1.000 ± 0.000	374.820
Step 3	0.888 ± 0.059	0.771 ± 0.070	311.569	0.880 ± 0.047	0.743 ± 0.070	116.526	1.000 ± 0.000	1.000 ± 0.000	366.586
Step 4	0.887 ± 0.047	0.740 ± 0.047	291.224	0.880 ± 0.041	0.729 ± 0.033	102.648	1.000 ± 0.000	1.000 ± 0.000	348.416
Step 5	0.893 ± 0.081	0.748 ± 0.051	270.743	0.890 ± 0.025	0.743 ± 0.043	98.810	1.000 ± 0.000	1.000 ± 0.000	317.038

Table A.1: ToW and ToW-MIA scores for NegGrad+ over 5 unlearning steps obtained by Zhao et Al. [23]

Appendix B

Gantt Charts

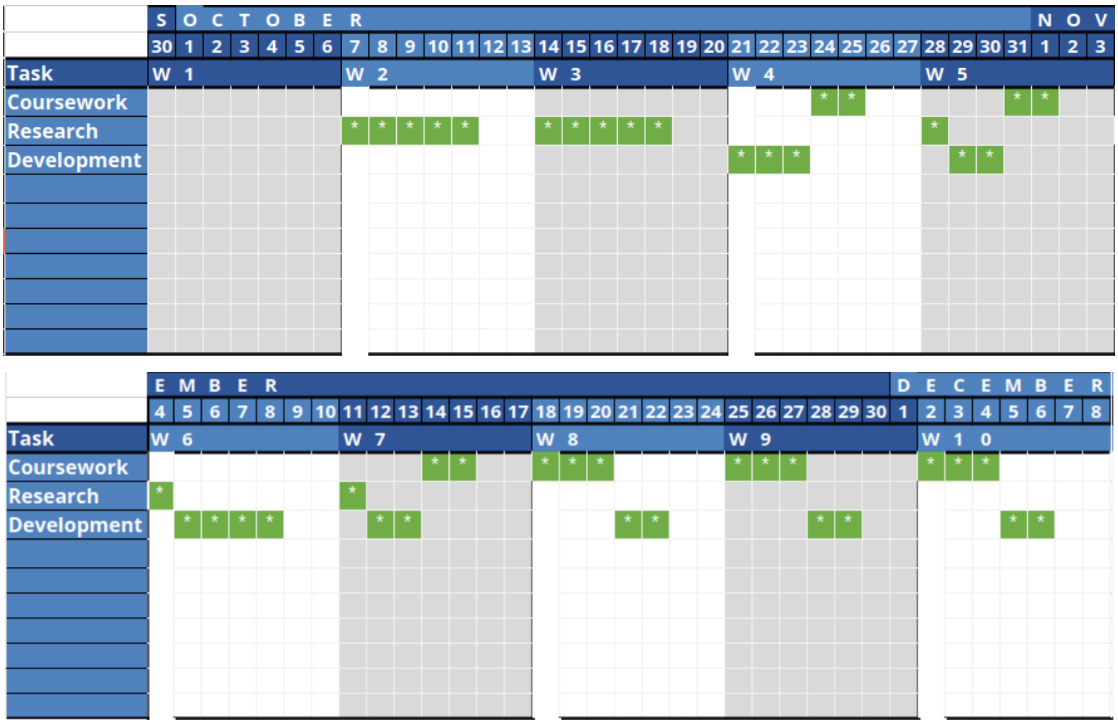

Figure B.1: Gantt Chart for Term 1

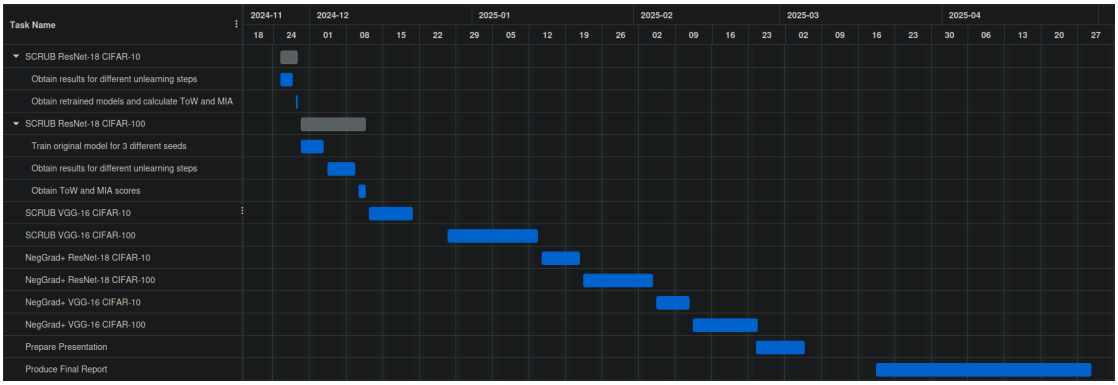

Figure B.2: Gantt Chart for Term 2

Appendix C

Hyperparameters

Hyperparameters that are not mentioned use the default value used in the RUM codebase [21]. All results were obtained by running code on the gecko node of the DCS batch compute system which uses an AMD EPYC 7443 CPU and a Nvidia A10 GPU.

Dataset	Epochs	Batch Size	Learning Rate
CIFAR-10	30	256	0.1
CIFAR-100	100	128	0.1

Table C.1: Training Hyperparameters

Model	Unlearn Epochs	Unlearn Rate	Class to Replace	Num Indexes	Alpha	Gamma	Max Steps
All Models	10	0.01	-1	3000			
C10-R18	10	0.01	-1	3000	0.9	0.9	4
C10-V16	10	0.01	-1	3000	0.1	0.005	7
C100-R18	10	0.01	-1	3000	0.9,	0.9	5
C100-V16	10	0.01	-1	3000	0.9	0.9	7

Table C.2: SCRUB Unlearning Hyperparameters

Dataset	Unlearn Epochs	Unlearn Rate	Class to Replace	Num Indexes	Beta Value
All Models	10	0.01	All	3000	-
CIFAR-10	10	0.01	All	3000	0.9
CIFAR-100	10	0.01	All	3000	0.95

Table C.3: NG Unlearning Hyperparameters

Model	Unlearn Epochs	Execution Order
SCRUB	$10 \rightarrow 10 \rightarrow 10$	low $\rightarrow$ mid $\rightarrow$ high
NG	$5 \rightarrow 5 \rightarrow 10$	low $\rightarrow$ mid $\rightarrow$ high

Table C.4: RUM Unlearning Hyperparameters

Appendix D

Runtimes

Unlearn Method	Total Runtime(h)
Retrain	175.4
NegGrad+	69.1
SCRUB	62.9
RUM NegGrad+	101.4
RUM SCRUB	92.8

Table D.1: Total Runtime for experiments for 5 seeds, VGG-16 and ResNet-18 architectures, CIFAR-10 and CIFAR-100 datasets.