Machine Unlearning for Class Removal through SISA-based Deep Neural Network Architectures

by

Siam Ferdous
21201526
Nabid Hasan Omi
23241105
Ishrak Hamim Mahi
24241312
Md Habibun Nabi Hemel
22241042
Md Sakib Sadman Badhon
22341082

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
October 2025

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Siam Ferdous

21201526

Nabid Hasan Omi

23241105

Ishrak Hamim Mahi

24241312

Md Habibun Nabi Hemel

22241042

Md Sakib Sadman Badhon

22341082

Approval

The thesis/project titled "Machine Unlearning for Class Removal through SISA-based Deep Neural Network Architectures" submitted by

1. Siam Ferdous (21201526)
2. Nabid Hasan Omi (23241105)
3. Ishrak Hamim Mahi (24241312)
4. Md Habibun Nabi Hemel (22241042)
5. Md Sakib Sadman Badhon (22341082)

of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on 10 October 2025.

Examining Committee:

Supervisor:
(Member)

Dr. Farig Yousuf Sadeque

Associate Professor
Department of Computer Science & Engineering
Brac University

Secondary Supervisor:
(Member)

Md. Tanzim Reza

Senior Lecturer
Department of Computer Science & Engineering
Brac University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science & Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Associate Professor & Chairperson
Department of Computer Science & Engineering
Brac University

Abstract

As the adoption of image generation models and other AI systems accelerates, concerns around user data privacy and consent have become increasingly critical. With the slowdown in growth of publicly available data, major tech companies are anticipated to rely more on proprietary and private user data for training their models. This raises ethical and legal questions, particularly when users request data deletion after it has already influenced a trained model—a process that is both technically challenging and resource-intensive. The emerging field of machine unlearning addresses this issue by developing techniques to remove specific data from trained models. Among these, the SISA (Sharded, Isolated, Sliced, and Aggregated) framework has shown promise as a scalable and privacy-aware unlearning solution. In this research, we focus on leveraging a modified SISA framework to enable class-level data removal in Convolutional Neural Network (CNN) architectures. We evaluate how well the modified SISA framework supports effective class unlearning without requiring full model retraining, and analyze the trade-offs in model performance, accuracy, and privacy. Through experiments across various image datasets and CNN architectures, we aim to demonstrate the practical viability of SISA-based class unlearning for real-world applications, offering insights into its strengths, limitations, and potential for deployment in privacy-sensitive AI systems. The code for this research is publicly available at https://github.com/SiamFS/sisa-class-unlearning.

Keywords: Machine Unlearning, Privacy, CNN, AI, SISA, Class Unlearning, Reinforced Replay Mechanism, Gating Network

Acknowledgement

First and foremost, praise to Allah for His blessings and guidance throughout our journey. We also express our gratitude to our supervisors, Dr. Farig Sadeque sir and Md Tanzim Reza sir, for their advice and support throughout our work. They have always been there to guide us on the right path. We are deeply grateful for the support of our family and everyone who has contributed to our work directly or indirectly.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

Aggregation Function: Method to combine predictions from multiple models. We have used argmax for this aggregation

Checkpointing and Rollback: A mechanism that saves the model's state after each slice training, allowing it to revert to an earlier checkpoint when a class is unlearned, thereby enabling exact unlearning and minimizing retraining overhead

Class Unlearning: The loss of a model's ability to correctly recognize or classify data belonging to a particular class

Gating Network: A specialized neural network that functions as an intelligent router, directing input samples to the most appropriate shard model for prediction

Replay Buffer: A memory component that stores selected samples from previous slices, along with their class and slice information, to reinforce earlier knowledge during incremental training

Shards (S): Independent, non-overlapping partitions of the dataset, each used to train a separate model instance, ensuring isolated learning and localized retraining

Slices (L): Sequential subdivisions within each shard that enable incremental model training and facilitate efficient rollback and retraining through checkpointing

Chapter 1

Introduction

1.1 Background

AI is not new in the history of mankind. Its presence can be noticed as far back as ancient Greece, where inventors like Hero of Alexandria (1st century AD) invented automata, or self-operating machines, such as mechanical birds and water-powered robots. Although it cannot be called intelligent by the present definition of AI, however, it was the start. Over the centuries, many theories and formulas were invented to automate human reasoning. As a result, the mechanical computer and programming were invented, which laid the groundwork for today's concept of AI.

The first use of actual AI and machine learning (ML) can be noticed in the Checkers Program, which was built by Arthur Samuel in 1951. This program could play checkers and improve its performance through learning [1]. In the same year, Alan Turing published a paper designing a system that can play a full game of chess. Even though those inventions were groundbreaking, they failed to catch the attention of the public. That changed in 1997 with the chess match between a supercomputer named Deep Blue and the reigning world chess champion, Garry Kasparov. Although Deep Blue lost the initial match, it won the rematch, marking a major milestone in the history of AI. Through this event, the general public noticed the power of the AI, which sparked a wave of interest among the general users about the field [1]. Over the following years, researchers continued to explore different learning techniques, particularly neural networks, which mimic the structure of the human brain using layers of interconnected nodes or "neurons." Neural networks became the foundation for many modern AI systems. Early versions, such as perceptrons and multilayer feedforward networks, paved the way for more complex architectures [3]

As time passed and research progressed, more advanced variants of neural networks were introduced. Convolutional Neural Networks (CNNs) were designed to process spatial data by applying filters that detect hierarchical features across images [12]. However, they often struggled with computational efficiency when processing high-resolution images and required large amounts of training data. To address these limitations, techniques such as batch normalization and residual connections were developed, which enabled the training of deeper networks and improved feature learning [19]. Advanced architectures like ResNet followed, introducing skip con-

nections that allowed gradients to flow more effectively through very deep networks. Eventually, a major leap occurred with the development of Generative Adversarial Networks (GANs) and later diffusion models [12]. Unlike traditional CNNs used for classification, these architectures employ generative mechanisms to create realistic images from noise or text prompts, enabling unprecedented creative capabilities. This evolution revolutionized computer vision, enabling the creation of powerful models for image synthesis and manipulation. These models are used for generating and editing visual content, from photorealistic images to videos, and have become foundational tools in computer vision and creative AI applications.

Using these models, OpenAI and other companies have developed generative AI systems capable of creating visual content that rivals human creativity. Among them, OpenAI's SORA and other image generation models have gained widespread attention and popularity. With systems like SORA, the world has noticed the power of generative AI and image generation models.

1.2 Motivation

One significant issue with training AI models like GPT is the need for large amounts of data. Companies often source these data from the internet, where most of the contents are publicly accessible but not necessarily meant for unrestricted use. While humans can understand consent, AI cannot, because of that it may incorporate restricted or sensitive data. Moreover, due to the advancement of user data regulations, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), the concept of "the right to be forgotten" has gained prominence. Because of that, if a user requests to remove their data from an AI model, companies are obligated to comply for both ethical and legal reasons [2]. For unlearning, one of the most well-known frameworks is the SISA framework, While it was originally demonstrated on classification tasks using simpler architectures like CNN, this paper tries to implement the same principles of SISA for class unlearning in image processing models like CNNs, with the goal of eventually extending this approach to more complex architectures such as Transformers.

1.3 Problem Statement

Removing data from a database is relatively straightforward. A company just deletes the data from the storage or the cloud server. But in the case of trained AI models, data removal is hard because of a few reasons [5]:

- **Distributed Representation of Knowledge** - Knowledge is "distributed" across many parameters rather than being stored in a specific, localized way. Removing one piece of information (e.g., a fact, text, or behavior) requires identifying and altering all the parameters contributing to it, which is extremely complex.

- **No Explicit Memory** - Unlike a database where specific entries can be deleted, a neural network does not have an explicit memory where individual pieces of information are stored.
- **Hierarchical Feature Learning in CNNs** - CNNs learn hierarchical representations where early layers capture low-level features (edges, textures) while deeper layers learn high-level, class-specific features. When attempting to unlearn an entire class, these learned features are entangled across multiple layers, making it challenging to selectively remove class-specific knowledge without affecting shared features used by other classes.
- **Lack of Testing Method** - the existing testing methods are ambiguous to ensure that the data has been deleted from the model.
- **Shared Convolutional Filters** - In CNNs, convolutional filters are shared across all classes during training. A single filter may activate for features present in multiple classes. Removing knowledge of one class requires careful modification of these shared filters to avoid degrading the model's performance on remaining classes.
- **Class Interdependency** - During training, CNNs learn to distinguish between classes by identifying discriminative features. The decision boundaries between classes are codependent, meaning the removal of one class can alter the decision boundaries for other classes, potentially causing misclassification or reduced accuracy on the remaining classes.
- **Black Box** - It is difficult to trace how specific data contributes to a model's behavior. Pinpointing the exact data and removing it without altering other data points is trivial and has the power to alter the whole model.

Because of that, the industry method has been to retrain the data from scratch, but for large-scale image generation models and complex CNN architectures, retraining would take massive computational power and energy, making this approach impractical.

1.4 Research Objectives

Researchers have developed multiple methods for machine unlearning. These methods produce different results based on what needs to be removed, such as specific data points, features, or entire classes. Currently, the SISA framework is one of the most practical and widely used approaches for machine unlearning. The original SISA framework was designed to delete specific data points from CNN models. However, this paper focuses on a different challenge: removing entire classes from a trained CNN model using a modified version of the SISA framework. This approach of class unlearning is more relevant to real-world scenarios where organizations need to remove all data related to a particular category rather than individual data points. The goal of this research is to demonstrate that the modified SISA framework can effectively unlearn entire classes from CNN models without requiring complete retraining, making it a practical solution for data privacy compliance.

Chapter 2

Background and Literature Review

2.1 Reasons for Machine Unlearning

Machine Unlearning has emerged as a critical capability in modern day AI systems in order to ensure legal, ethical, security, environmental and operational imperatives. Below, we outline these factors and the trade-offs related to them with model performance.

2.1.1 Legal Compliance

Regulations like European Union's GDPR and the California's CCPA require models to remove specific user data if requested in order to enforce "right to be forgotten" popularly referred as RTBF [2]. These regulations grant the users the right to request to delete their personal data which includes any kind of residual impact it might leave on the trained models. For example, healthcare models which are trained on critical patient data must delete the data if the patient requests. However, its removal can inadvertently erase statistically critical patterns leading to accuracy loss. [14] Demonstrated in their work that compliance-driven unlearning in facial recognition systems reduces precision by 12-15% due to fragmented feature representations. Apart from that, conventional model retraining is often impractical due to high computational costs which makes unlearning techniques a legally viable solution [4].

2.1.2 Ethical Imperative

Data misuse, algorithmic bias and fairness raises the ethical concern for machine unlearning. Models which are trained on biased datasets can be subject to discrimination, leading to unfair decision-making. Furthermore, without proper consent from the user, training AI models on personal data raises ethical concerns. Machine unlearning provides a solution to address these concerns by enabling models to "forget" specific data points [14].

2.1.3 Security

Machine unlearning serves as a very useful tool for addressing security concerns in AI systems by enabling proactive mitigation of threats while keeping a balance with performance trade-offs. Unlearning allows targeted removal of poisoned data injected during training. [14] demonstrated that unlearning adversarial samples can reduce attack success rates by 35-50% in image classifiers with a 4-8% accuracy drop on clean data. [31] showed that unlearning can erase backdoor triggers in federated leaning models, restoring model reliability by 20-30% while preserving overall accuracy.

2.1.4 Environmental Cost

In order to reduce the validation loss of the large models, training them with a huge corpus of data for a longer period of time has been proven very useful. Training large models has significant environmental costs. For instance, training GPT-3 consumed 1,287 MWh of energy and emitted 552 tons of CO_2 [7]. Retraining such heavy models for minor updates just increases this carbon footprint. [16] explores how optimizing training compute can reduce environmental impact. Unlearning offers a eco-friendly alternative by avoiding full retraining, but its approximate methods (e.g., gradient updates or shrading) often sacrifice accuracy for efficiency. [8] shows while unlearning reduces CO_2 emissions by 60-80% compared to retraining, it accelerates accuracy decay by 3-5% per removal cycle.

2.1.5 Operational Scalability Costs

In real world deployments traditional retraining strategies involve significant downtime and resource expenditure, making them infeasible for large scale applications. Machine unlearning addresses this challenge by enabling efficient updates without full retraining, ensuring continuous model improvement and real-time adaptability [6]. Also, as models grow in size and complexity, full retraining becomes prohibitively expensive. [11] shows that doubling model size requires exponentially more compute and data to maintain performance.

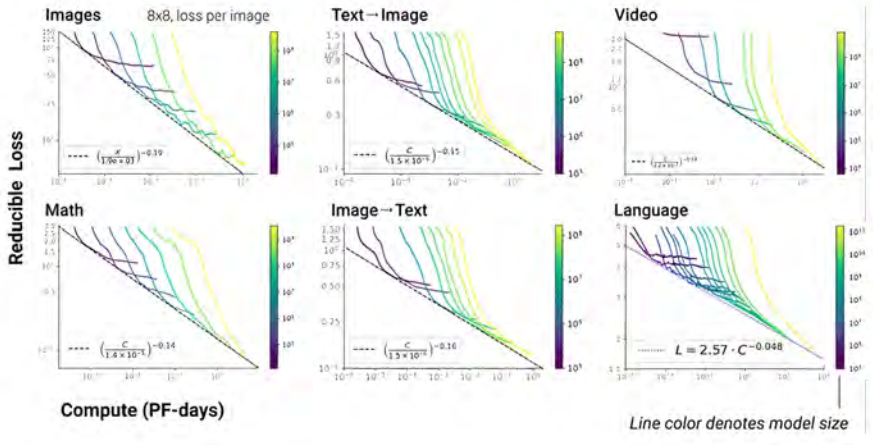

Figure 2.1: Model Expense

As the models will evolve to keep up with the increasing demand to be more accurate, retraining will become obsolete and machine unlearning will be the only optimum solution to address these issues.

2.2 Challenges in Machine Unlearning

Machine unlearning is a complex process of making a machine learning model forget certain information more specifically data, which is important for privacy laws like the GDPR [14]. As the models keep getting complex with time, they appear like black boxes which makes the data removal process very difficult as we struggle to navigate which parameters should be tuned to remove the targeted data point. For instance, a person requests to delete their data from a face recognition model the model should no longer recognize them. However, achieving this is very difficult because models do not store data like a simple database; instead, they learn patterns from the data and spread that information across their internal structure.

2.2.1 Hard to Track Influence of Data

Data is processed in small portions, such as random batches to train machine learning models. Consider training a baby to identify animals: it is impossible to determine which image helped them identify a cat if the baby are shown different images of cats and dogs at various times. Similarly, in a machine learning model, removing a particular data point does not simply remove its influence because its effect is distributed across the entire model [13].
Mathematically, if a model learns from a dataset X and we remove a specific part x, we expect the new model $A(X - x)$ to behave similarly to the old model trained on the full dataset. However, this is not always the case because training is stochastic (randomized). This can be expressed as:

$$\text{New model } A(X - x) \neq \text{Old model } A(X) - \text{Effect of } x$$

Instead of simply subtracting the data, the entire model structure needs to be adjusted.

2.2.2 Incrementality of Training

In the training of a model, a single data point affects not only the current step but also future learning. This indicates that eliminating a single data point has an impact on how the model understands future data in addition to eliminating its influence. If we update the model over time using:

$$\text{New Machine Learning Model} = \text{Old Machine Learning Model} - \text{Minor Adjustment}$$

then removing a single data point would require undoing multiple past adjustments, which will make the unlearning process too complex [9].

Mathematically, a model updates its parameters θ over time:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t, x_i) \tag{2.1}$$

where:

- η is the learning rate (how fast the model learns)
- $\nabla L(\theta_t, x_i)$ is the change caused by the data x_i.

2.2.3 Catastrophic Unlearning

Cleaning too much data at a time can damage a model and lose its performance in terms of accuracy, precision, and F_1 value. Moreover, removing lots of data can be the reason that the model loses its knowledge completely and its prediction prediction becomes less accurate. [18]. If the model is trained on data X and remove a single portion a, the new model should ideally behave like a model trained only on $(X - a)$ but in reality it does not behave the same as that. For example, in an auto-driving car, if all the data related to stop signs gets removed, the car might no longer recognize any kind of sign, leading it to dangerous situation. That means Forgetting Too Much can ruin the Model. In mathematical terms, the model should satisfy:

$$\mathbb{E}\left[\|A(D - D_f) - A(D)\|\right] \leq \epsilon \tag{2.2}$$

where ϵ is a small number that ensures the model remains accurate. But if we remove too much data, ϵ becomes large, meaning the model behaves very differently.

2.2.4 Trade-off Between Accuracy and Efficiency

The easiest way to remove data is to train the model again from the beginning, but it takes too much time and computational power. Despite doing that, we can use unlearning, where we make specific changes to the model to forget the data. However, this often leads to small errors and drop in the overall accuracy of the model as the data can have an influence on more data that is used in past or future training. So, we need to maintain a baseline to balance both accuracy and not using heavy operations to get a desired model. A good unlearning algorithm should ensure that the new model behaves almost the same as a retrained one while being much faster [39].
To measure how well a model has forgotten, we use distance metrics like:

$$D_{KL}(P(A(D - D_f)) \parallel P(U(D, D_f, A(D)))) \leq \delta \tag{2.3}$$

where D_{KL} is the Kullback-Leibler divergence, measuring how much the new model differs from the retrained one. If δ is small, the model has successfully forgotten the data [9]

2.3 Unlearning Framework

A machine unlearning framework is a structured and methodological approach which enables the machine learning models to selectively remove its learned information (i.e: specific data points, subsets of data and learned patterns) while retaining its overall performance. This is essential to meet privacy and security compliance issues like GDPR's RTBF(Right To Be Forgotten). Here are some famous machine unlearning frameworks are discussed below:

2.3.1 SISA Framework

One of the earliest frameworks for machine unlearning is the SISA (Sharded, Isolated, Sliced and Aggregated) training framework. Introduced by [14] this framework aims to achieve the unlearning process by strategically limiting the influence of individual data point during model training. The framework involves this simple approach of partitioning the dataset into multiple shards. These shards are further divided into slices. Models are trained on these isolated slices and their outputs are aggregated to form the final model. Upon receiving a data deletion request, only the specific slice which contains the data needs to be retrained. This approach drastically reduces the computing overhead.

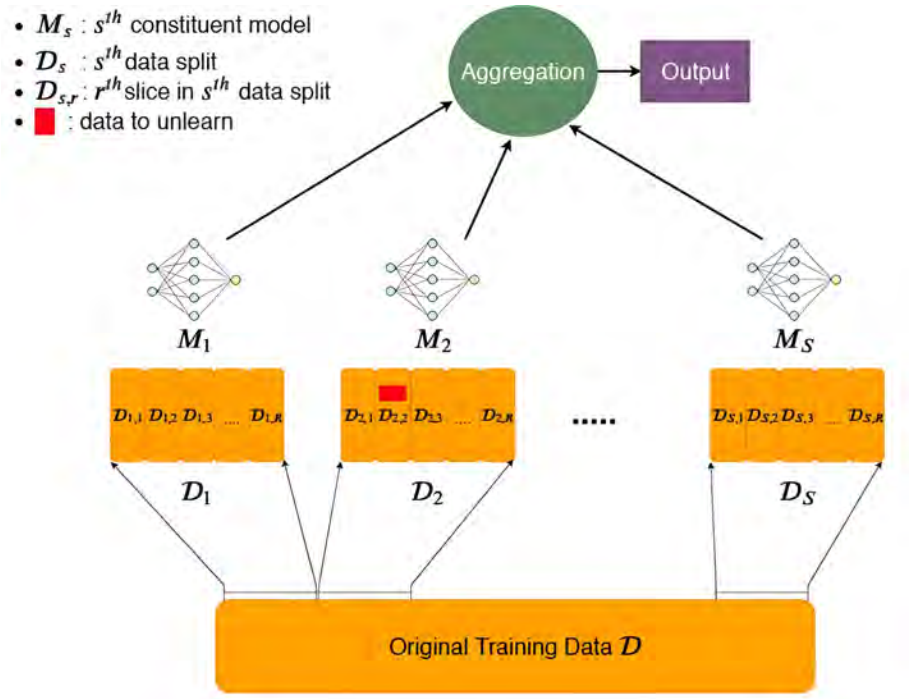

Figure 2.2: SISA Framework

2.3.2 EMN Framework

The EMN Framework is a novel machine unlearning framework proposed by [35]. The EMN stands for Error Maximizing Noise. This framework works by deliberately injecting noise which maximizes error for the class or data which is to be forgotten while retaining the overall performance of the model. In this approach an error maximizing noise matrix is learned for the class(es) which is to be "forgotten" from the original model. This noise is then used to tweak the model's parameters associated with the particular class(es) which is to be forgotten. Consequently a repair setup is implemented to regain the overall model accuracy without reintroducing the forgotten class(es). This method allows for unlearning multiple classes simultaneously in an efficient manner with minimal computational cost.

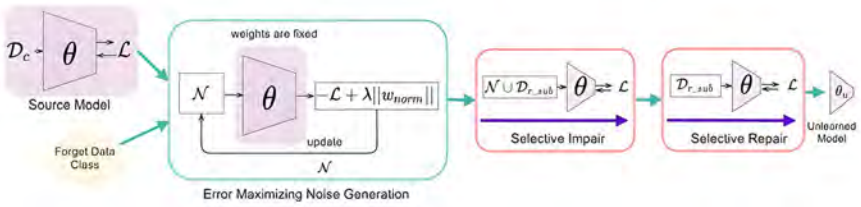

Figure 2.3: EMN Framework

2.3.3 SIBU Framework

SIBU stands for Statistical Inference-Based Unlearning which was introduced by [2]. This approach uses statistical tools like hypothesis testing or confidence intervals to find how much influence does a data point (which is to be forgotten) have on model decisions. After finding the influence over the model decisions , the parameters of the model are adjusted to undo the effect of the forgotten data. This method treats unlearning as an optimization problem.

2.3.4 KGA Framework

[25] propose Knowledge Gap Alignment (KGA), a novel unlearning framework for NLP that is particularly effective for classification, machine translation, and response generation that efficiently removes data from a model while maintaining its performance. The authors demonstrate that KGA is a more efficient alternative to SISA. As SISA becomes ineffective while handling a large number of data deletions, KGA maintains its performance. Unlike SISA, which stores and trains sharded small subsamples of data, KGA does not need to retain the original dataset making it more scalable and storage efficient. Moreover, KGA uses low distributional drift after the unlearning process while maintaining the model accuracy. According to the paper [25], KGA works by training two assistant models where one works on the forgotten data (A_f) and another on unrelated external data (A_n). After that, the main model (A) receives adjustments by making distributions from the knowledge gap match between those two models. Instead of directly modifying the learned weights, KGA minimizes this gap through a specific objective function.

$$\mathcal{L}_a = \sum_{(y,z)\in(D_f,D_n)} KL[Pr_{(A^*)}(y)||Pr_{(A_f)}(y)] - KL[Pr_{(A_D)}(z)||P_{(A_n)}(z)] \tag{2.4}$$

Where KL divergence tells how different the distribution is. Furthermore, KGA keeps the model performance consistent by minimizing another KL divergence loss over retained data (D_r).

$$\mathcal{L}_r = \sum_{x\in D_r} KL[Pr_{(A^*)}(x)||Pr_{(A_D)}(x)] \tag{2.5}$$

Finally, both unlearning and retention are optimized to the final objective function ($\mathcal{L}$) to ensure effective unlearning of the data and retaining accuracy.

$$\mathcal{L} = \mathcal{L}_a + \alpha \cdot \mathcal{L}_r \tag{2.6}$$

Thus, continuous optimization of this loss function helps the main model to act as if it never saw the Forget set and still predicts the accuracy in retaining data.

2.3.5 Continual Learning and Unlearning

[17] introduces the CLPU-DER++ framework for Continual Learning and Private Unlearning that allows models to sequentially learn and selectively unlearn specific tasks. This method creates isolated temporary networks for forgettable tasks and a main model for retained knowledge, inspired by SISA training. To support unlearning, the temporary network is discarded upon requests of unlearning, keeping the integrity of other learned tasks and achieving exact unlearning without retraining. If retention is demanded, knowledge from the temporary network is distilled into the main through knowledge distillation, maintaining adaptability and stability. Thus, the optimization combines Dark Experience Replay++ (DER++) knowledge distillation, which enhances knowledge retention by replaying stored data and aligning raw model outputs with new task learning, along with cross-entropy loss.

2.3.6 Mechanisms of Certified Data Removal

Certified Data Removal (CDR) ensures that a machine learning model forgets specific data, guaranteeing it no longer influences predictions. The most direct method is retraining the model from scratch, excluding the removed data, but this can be costly and impractical for large models. A more efficient approach is using Sufficient Summary Models (SSMs), which only update key statistics like means and covariances, avoiding full retraining. Influence functions estimate how parameters would change if a data point were removed, allowing for efficient updates without retraining. Additionally, privacy-preserving methods like differential privacy introduce noise to model parameters, making it hard to tell whether a particular data point was used in training. These methods balance efficiency with strong privacy guarantees [10]

2.3.7 Gradient-Based Approximate Unlearning

Gradient-Based Approximate Unlearning (GBAU) is a strategy for rapidly eliminating the influence of certain data points from a trained machine learning model without having to start from scratch [36]. This strategy is especially beneficial when exact unlearning is computationally expensive.

Gradient-Based Approximate Unlearning offers an efficient alternative to exact unlearning by leveraging gradient information. While it provides a balance between computational efficiency and accuracy, it remains an open challenge to ensure complete data removal while maintaining model performance.
To remove the influence of a subset $D_r \subset D$ from a trained model f_θ on a dataset D, follow these steps. The aim is to update the model parameters θ to approximate a model trained without D_r, without completely retraining from scratch.
Here is the original training process:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(D, \theta^{(t)}) \tag{2.7}$$

And, If we had trained without D_r, the ideal update would be:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(D \setminus D_r, \theta^{(t)}) \tag{2.8}$$

Since retraining from scratch is impractical, gradient-based approximate unlearning estimates the effect of removing D_r using gradient information.

2.4 Unlearning Methods

Machine unlearning is a way by which it can remove certain data points or patterns from a trained machine-learning model and they no longer influence predictions. This method deals with complying with privacy laws, reducing bias, and improving model security. There are two main types of unlearning approaches. Those are exact and approximation methods.

2.4.1 Problem Formulation

Machine unlearning efficiently deletes specific data points or patterns from a trained model while preserving its overall functionality and accuracy. Unlearning is required in situations such as data privacy requirements (e.g., GDPR's right to be forgotten) and security issues in which incorrect or sensitive data must be removed from a machine-learning system [2]. A major problem in machine unlearning is to ensure that the model no longer has any impact from the removed data while minimizing computing overhead [14]. Unlearning tries to create a new model M' from an updated trained model M using dataset D, as if the specific subset $D_\mathrm{r} \subset D$ was never established.

2.4.2 Exact Unlearning

Exact unlearning assures that the model's parameters are completely restored to a statistically indistinguishable condition as if the removed data never existed. This can be accomplished by utilizing strategies such as re-training from scratch without D_r or efficient checkpointing systems [6]. The fundamental disadvantage of accurate unlearning is its high computational cost, particularly for large-scale models where retraining is not feasible. Despite this, exact unlearning ensures strict compliance with privacy standards while minimizing data remanence in the model [10].
Exact unlearning tries to achieve an updated model M' which is indistinguishable from a model trained from scratch on $D - D_\mathrm{r}$. Here, D represents the dataset, and D_r refers to the data points that have to be removed. This can be formulated as:

$$M'(D - D_\mathrm{r}) = M(D - D_\mathrm{r})$$

where M' is the updated model after unlearning, and M is the fresh model trained on the reduced dataset.

2.4.3 Approximate Unlearning

To overcome the inefficiency of exact unlearning, approximate unlearning strategies seek to eliminate the effects of D_r without requiring total retraining. These methods include influence function-based updates, statistical perturbations, and gradient-based approaches that change model parameters locally [9]. While approximate unlearning is computationally more efficient, it results in trade-offs between

efficiency, privacy, and verifiability. The main problem is determining how much residual information from D_r still influences the model and whether it passes privacy requirements [13].

If θ represents the model parameters, the updated model θ' is given by:

$$\theta' = \theta - \eta \nabla_\theta L(D_\mathrm{r})$$

where:

- θ': The updated model parameters after applying gradient descent.
- θ: The current model parameters.
- η: The learning rate (step size).
- $\nabla_\theta L(D_\mathrm{r})$: The gradient of the loss function L computed on the dataset D_r.

This describes how parameters are updated incrementally by calculating the loss function for data point removal.

2.4.4 Indistinguishability Metrics

Indistinguishability metrics tell us the effectiveness of machine unlearning by determining how closely an unlearned model reaches its ideal state. Common metrics have some parameter distance metrics. It measures changes in weight distributions before and after unlearning [33], prediction consistency and compares model outputs on test data before and after unlearning to ensure minimal functional drift [14], and membership inference resistance, which determines whether an adversary can still infer the presence of in the model post-unlearning [9]. These measurements are a quantitative foundation for benchmarking unlearning approaches and ensuring compliance with privacy and security objectives.

2.5 Unlearning Scenarios

Unlearning scenarios explain how the machine learning model will forget particular data, it can be a small portion of the full dataset and it can also be a big portion of the main dataset by maintaining the model's performance. These techniques are important because they address bias correction, protect security, and follow laws such as those established by the GDPR and the CCPA, which is the California Consumer Privacy Act [14]. The effectiveness, complexities, feasibility, and computational cost of each scenario vary, depending on the system's capacity to handle access to the training data [21].

2.5.1 Zero-glance Unlearning

Zero-glance unlearning enables the system to forget information without going over it again. When data deletion laws prevent organizations from keeping a duplicate of their data, this predicament arises [21].It is usual practice to introduce noise into the model, which makes it more challenging to find patterns related to missing

information [21]. The remaining data is then used to refine the model and restore its accuracy [13]. Adding error-maximizing noise N is a popular technique to reduce the model's previous performance on D_f, which is defined as follows:

$$\mathcal{N} = \arg\max_{\mathcal{N}} \mathcal{L}(\theta, D_f + \mathcal{N})$$

2.5.2 Zero-shot Unlearning

The zero-shot unlearning method is a controlled process where the machine loses some information and it will have no access to the training data [9]. Moreover, the old model is adjusted to do it behave like it did not train using the forgotten data [21]. A common tactic is knowledge distillation, whereby a new "student" model gets information from the "teacher" models without collecting unwanted information [33]. This method avoids direct access to previous information while maintaining model performance. It works well in situations where training information is not appropriately accessible but it is difficult to verify whether the model forgot the specific part or not.

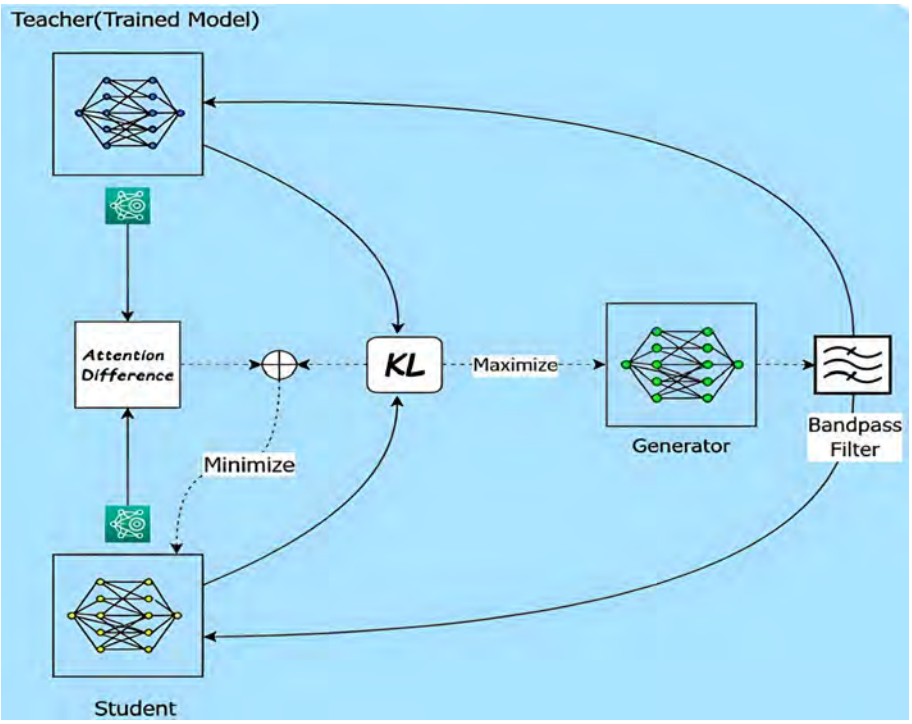

Figure 2.4: Zero-shot Unlearning.

2.5.3 Few-shot Unlearning

The few-shot unlearning method allows the use of a very small percentage of the erased data to help during the method of unlearning [26]. This is useful, for example, when partial access to deleted information is permitted, to remove biased samples while maintaining model accuracy [33]. It is possible to intentionally "untrain" the models from specific biases or sensitive features by using gradient reversal [32]. This method takes care of Fairness and accuracy in a better way but it is very hard to

tune it.

Selecting the perfect unlearning method depends on the situation why we cant specify that this method is good or that method is good, such as resource constraints, accuracy requirements, and privacy laws these things, we need to consider before selecting the unlearning method [21].

2.6 Unlearning Algorithms

Unlearning algorithms play a crucial role in machine unlearning by enabling the removal of specific data or learned representations from models while maintaining efficiency and accuracy. These algorithms are essential for privacy compliance, security, and ensuring models do not retain unwanted or outdated information. Unlearning approaches can be broadly categorized into three main types: model-agnostic, model-intrinsic, and data-driven approaches. Each of these methods varies in complexity, computational cost, and applicability depending on the use case.

2.6.1 Model-Agnostic Approaches

Model-agnostic unlearning techniques focus on removing learned information without requiring modifications to the underlying model architecture. These approaches often involve post hoc adjustments that erase or mitigate the influence of specific data points, making them versatile across different machine learning models. They are particularly useful when full retraining is impractical due to high computational costs. Common methods in this category include:

- **Gradient-Based Reversal** - This method reverses the learning process by applying negative gradients to reduce the influence of specific data points. By backpropagating a correction signal, the model's memory of particular information can be weakened [36]. When the training model uses gradient descent for learning, it uses gradient reversal for unlearning approximately. The unlearning formula for each parameter is:

$$\theta' = \theta + \gamma \nabla_\theta L(D_r, \theta) \tag{2.9}$$

  Here, θ' is the updated parameter and γ is the unlearning step size.

- **Knowledge Distillation-Based Unlearning** - This approach transfers knowledge from a teacher model (which includes undesired data) to a student model trained only on the remaining dataset. By selectively transferring knowledge, the student model effectively forgets the unwanted data while preserving performance [34] Here initially teacher model learns for each data point by the formula:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{2.10}$$

  Where

  z_i is the teacher model's output logits.

z_j is the raw output of the teacher model for class j before applying softmax.

T is the temperature parameter (a higher T makes soft labels smoother).

p_i represents the softened probability distribution over classes.

Then, training the student model without D_r;

$$L = \lambda L_{CE}(y, M'(D - D_r)) + (1 - \lambda)L_{KD}(p, M'(D - D_r)) \tag{2.11}$$

Where,

L_{CE} is the standard cross-entropy loss using hard labels.

L_{KD} is the distillation loss using soft labels.

λ controls the balance between the two losses.

- **Sharded Training and Ensemble Methods** - Instead of retraining an entire model, these techniques divide models into smaller components. When unlearning is needed, only the relevant parts are modified or retrained, reducing computational overhead [22]. The workflow of this model is: At first, we have to partition training data into k shards and then train independent models on each shard. Here shards means smaller, disjoint subsets of the training dataset and we use this for scalability. Then if we need to unlearn something have to find in which shard that data was trained and retrain that specific shard. Finally, model prediction is obtained by averaging or using a weighted ensemble of those updated data points.

Model-agnostic techniques offer flexibility, allowing practitioners to remove data without requiring full access to a model's architecture or internal parameters.

2.6.2 Model-Intrinsic Approaches

Model-intrinsic unlearning techniques are designed to integrate unlearning mechanisms directly into the model's architecture or training process. These methods ensure that forgetting is embedded into the training dynamics, enabling efficient removal of specific data points without external interventions. They are particularly useful in settings where continuous learning and updates are required. Examples include:

- **Retraining with Forgetting Constraints** - By incorporating constraints into the optimization process, this method selectively removes the influence of certain data points without having full retraining. It ensures that the unlearning process does not largely degrade model performance [30]. In this process, model tries to minimize the difference between the new model M' and the original model M. Formula is:

$$\min_{\theta'} L(D - D_r, \theta') + \lambda \|\theta' - \theta_{D-D_r}\|^2 \tag{2.12}$$

Where,

θ' are the parameters of the new model M'

θ_{D-D_r} are the parameters of a model trained only on $D - D_r$

$L(D - D_r, \theta')$ is the loss function.

λ controls how much the new model should stay close to a reference model.

Then for the optimization, it uses Lagrange regularization:

$L = L(D - D_r, \theta') + \lambda \|M' - M_{D-D_r}\|^2$

$$\mathcal{L} = L(D - D_r, \theta') + \lambda \|M' - M_{D-D_r}\|^2 \tag{2.13}$$

If λ is large, the retrained model will be very close to the reference model, minimizing forgetting.

If λ is small, more flexibility is allowed, which may help with generalization.

- **Regularization-Based Forgetting** - Regularization techniques can be applied during training to encourage models to forget specific information over time. This method introduces additional loss terms to push the model away from reliance on forgotten data [27]. In this process, it uses

$$\mathcal{L} = L(D - D_r, \theta) + \lambda \|\theta - \theta^*\|^2 \tag{2.14}$$

Where,

$L(D - D_r, \theta)$ is the loss function on the retained dataset.

θ^* are the original model weights before training on D_r

λ controls the strength of forgetting.

- **Pruning and Weight Modification** - By selectively modifying or pruning model parameters associated with unwanted data, the model can forget targeted information. This approach is effective in neural networks, where certain neurons or weights contribute significantly to specific learned patterns [15]. It works by tracing the neuron that activates for Dr and after tracing the neuron, it fine-tunes that for forgetting. For tracing it uses the formula:

$$A_j = \sum_{x \in D_r} |f_j(x)| \tag{2.15}$$

where $f_j(x)$ is the activation of neuron j for sample x.

Now, after computing the activation score of each neuron for D_r, it prunes the higher influenced neurons. Then fine-tune the remaining network on $D - D_r$ to restore performance.

Model-intrinsic approaches offer structured and systematic unlearning capabilities, making them ideal for scenarios where unlearning is a frequent and necessary operation.

2.6.3 Data-Driven Approaches

Data-driven unlearning techniques focus on modifying the dataset before or during training to ensure that models do not learn undesired information in the first place. These approaches are particularly useful in privacy-sensitive applications, where proactively managing training data can prevent the need for post hoc unlearning. Methods in this category include:

- **Data Augmentation for Forgetting** - Instead of directly training on raw data, modified versions of the dataset can be used to ensure that certain patterns are not learned. This helps prevent models from internalizing sensitive or unnecessary information [37]. In this section, noise injection is an augmentation technique and in this process, we have to identify which data point we want to remove or forget. Then we have to inject some random noise to those data points with gaussian noise:

$$x' = x + \mathcal{N}(0, \sigma^2) \tag{2.16}$$

  where

  x' – The new version of x after adding noise

  x – The original data point.

  $\mathcal{N}(0, \sigma^2)$ – A Gaussian (normal) noise with:

  Mean 0 (i.e., centered around 0).

  Variance σ^2 (or standard deviation σ).

  Then we have to retrain or finetune that model with this augmented dataset so that the model can forget by less focusing on the original input data points.

- **Differential Privacy Mechanisms** - Differential privacy techniques introduce noise into the data, making it difficult for models to memorize specific details. This approach is widely used in privacy-preserving machine learning to ensure that learned representations remain untraceable to individual data points [23]. In this section, we add some noise such as Laplace's noise so that no data point can be identified with certainty by using the formula:

$$\mathcal{L}(D) \to \mathcal{L}(D) + \mathcal{N}\left(0, \frac{\Delta f}{\epsilon}\right)$$

  where $\mathcal{N}\left(0, \frac{\Delta f}{\epsilon}\right)$ is Laplace noise, and Δf is the sensitivity of the function.

- **Instance-Based Forgetting** - This method identifies and selectively removes or modifies data instances that contribute most to learned representations. By using statistical measures or influence functions, the impact of certain data points can be minimized [40]. In this section we first compute the influence function I(x) of data point x which is going to make change in model parameters. Then using the influence function, we determine how to undo the effect of the data point. Influence function is:

$$I(x) = -H^{-1} \nabla L(x) \tag{2.17}$$

where H is the Hessian matrix (second-order derivative) and $\nabla L(x)$ is the gradient of the loss with respect to the model's parameters.

After this, we remove the influence of x by adjusting the model's weight based on the computed influence.

Data-driven approaches provide a proactive solution to unlearning, ensuring that models do not internalize problematic information from the beginning.

2.6.4 Differences Between Approaches

- **Scope of Modification** - Model-agnostic methods work externally and do not require changes to the model's architecture, while model-intrinsic approaches modify the training process itself. Data-driven methods focus on altering the dataset rather than the model.
- **Computational Efficiency** - Model-agnostic methods tend to be computationally efficient as they avoid full retraining, while model-intrinsic approaches may involve complex adjustments during training. Data-driven methods are efficient at the data preprocessing stage but may require additional steps during training.
- **Flexibility** - Model-agnostic approaches offer broad applicability across different models, while model-intrinsic techniques are tailored to specific architectures. Data-driven methods provide flexibility in dataset modifications but may not always be applicable post-training.
- **Privacy and Compliance** - Data-driven approaches are effective in ensuring privacy compliance from the start, whereas model-intrinsic and model-agnostic methods provide post hoc solutions to privacy concerns.

Each unlearning algorithm category offers distinct advantages and trade-offs, depending on computational constraints, privacy requirements, and model accessibility. Model-agnostic approaches provide flexibility without requiring changes to the core architecture, model-intrinsic techniques integrate forgetting into the learning process, and data-driven approaches focus on dataset modifications to enable efficient unlearning. Understanding these approaches helps in designing robust and adaptable machine learning systems that can comply with evolving data privacy and regulatory standards.

2.7 Unlearning Applications

Unlearning applications in machine learning refers to the ways and areas where unlearning techniques are used. It is applied in areas like federated unlearning where user data has to be forgotten, old data or biased information needs to be removed from lifelong learning or need to remove harmful, inappropriate misleading content from large language models and generative AI. Unlearning applications play an important role in protecting privacy, ensuring fairness, and complying with legal regulations which make the models more reliable and ethical.

2.7.1 Unlearning in Federated Learning

Federated Learning allows multiple clients the ability to train one universal machine learning model through a central server-based approach by keeping data on each device locally while sharing only model parameter changes to the server towards encrypted processing. The method protects personal data privacy in critical areas such as medical services and financial businesses. However, Client contribution to the global model is a necessity according to data protection rules such as GDPR and CCPA which mandate the RTBF. Such challenges make federated unlearning particularly challenging to implement. Normally, when removing a client's contribution in most cases would require retraining the model from scratch, which is computationally expensive and time-consuming [20].

To tackle this challenge [20] proposes a federated unlearning framework that effectively removes a client's contributions with little impact on the performance of the model. Their approach consisted of measuring the historical parameter updates of a particular client and then recovering the model's accuracy using knowledge distillation to remove the parameter updates from the global model. In the case of knowledge distillation, the pre-unlearning model was treated as a teacher which informed the unlearned student model. They propose a server-side approach that does not require client participation or labeled data. The framework is evaluated using three public datasets. MNIST, CIFAR-10 and GTSRB using models of convolutional neural networks VGG11 and AlexNet. To check unlearning effectiveness they introduced backdoor attack scenarios in which malicious clients induce harmful patterns into the model. The model successfully reduced the success rates of backdoor attacks to almost zero while still ensuring the model's high accuracy. The unlearned model reached an accuracy of 98.5% on the MNIST dataset comparable to that of the original model. However, the method has some limitations its reliance on external unlabeled data for knowledge distillation and potential inefficiency in highly dynamic or large-scale FL systems.

After the unlearning process, it is essential to verify whether the data has been deleted or not. However, verifying unlearning in a federated approach is challenging as traditional methods require training the model from scratch. Additionally, existing methods like uDP add noises to the model which heavily impacts model accuracy. Furthermore, gradient subtraction methods such as uCGS, uGGS, uIGS require multi-round operations, which is time-consuming and computationally expensive [29]. To solve this issue, [29] introduced a framework VERFI which allowed the users the right to verify where users can check the unlearning effect using marking and checking techniques through the uS2U method. This method improves efficiency by gradient scaling where the leaving user data contribution is downscaled and other data is upscaled to ensure the model maintains its accuracy. The model mark using vFM and vEM sample which track the contribution of leaving users. Then model checked and evaluated marking data using loss, KL divergence, accuracy, and influence function. If the model's response accuracy for marked data is below the predefined threshold then it is confirmed that the data is effectively unlearned. The method achieved the best accuracy of 98.5% on the MNIST dataset. Though it has some limitations in large FL systems, adversarial robustness and evolving data handling, where future work can be done for optimization unlearning

for dynamic data.

2.7.2 Unlearning in Lifelong Learning

Lifelong unlearning refers to the ability of a continuous learning system to selectively forget specific knowledge while maintaining overall performance. This ability is essential where organizations need to remove learned information to meet privacy, security, fairness or regulatory requirements [14].

[17] propose Continual Learning and Private Unlearning (CLPU) which aimed to work on these challenges by developing a method that can remove specific tasks based on request without leaving a trace while maintaining the model performance. Such methods are crucial for applications such as AI assistants, autonomous systems and recommendation engines where organizations or users may want to remove sensitive interaction, remove biases or comply with data protection regulations. Furthermore, [17] developes the CLPU-DER++ method that creates temporary models for tasks that might require future deletion. These models are simply removed when unlearning is required instead of changing the whole network making the process faster, more efficient and ensuring privacy. These models were implemented in a dynamic architecture which helped the model to adaptively create and remove temporary subnetworks for selecting unlearning. Core technologies like Deep Neural Networks were used for strong learning capabilities, DER++ for remembering past knowledge while learning new tasks which prevents the model from forgetting important information. Lastly, Knowledge Distillation was used for task merging and efficient unlearning.
The tests were done in publicly available datasets Rotation MNIST, Permutation MNIST, Split CIFAR-10 and Split CIFAR-100 which were commonly used for continual learning. The test outperformed traditional continual learning methods on the Permutation MNIST dataset with 93.48% accuracy while successfully removing design tasks with minimal interference with other knowledge. However, the major drawback of this model is high memory conjunction due to storing temporary methods that may not be scalable for large-scale applications. Future research should focus on reducing memory overhead while ensuring privacy-preserving unlearning.

2.7.3 Unlearning in Large Language Models

Large Language Models (LLMs) have come a long way in using large datasets to perform tasks like text generation and translation and summarization quickly and efficiently. While they depend on high amounts of data, they pose ethical issues in particular when trained on highly private, personal or copyrighted content, which carries data ownership, privacy and misuse challenges [41].

In [41] they present a new methodology that enables Large Language Models to selectively forget certain pieces of learned information without the need to retrain the entire model. Their proposed method included gradient ascent methodology which intentionally amplifies model errors on targeted information to weaken its influence, fine-tuning with randomized labels which disrupts learned patterns by randomly reassigning training labels and adversarial unlearning, which uses specially designed

examples to lower the model's confidence in certain data points. To validate their framework, they used the Yi-6B LLM on a variety of datasets including arXiv, GitHub and literary works. They showed their results provided substantial computational efficiency while maintaining the model functionality and analyzed their performance by using perplexity and accuracy metrics Evaluation of the unlearned model demonstrated an accurate 78.79% performance with general data comparable to vanilla model levels performance while successfully forgetting targeted data. However, the model depends on external data for knowledge transfer and faces scalability challenges for larger models. The authors emphasized that hyperparameter adjustment plays a crucial role in maintaining model performance with successful unlearning.

Following the key challenges raised [41], [24] identified three distinct unlearning approaches for LLMs which include parameter optimization alongside parameter merging and in-context learning. They used these to show that standard unlearning methods cannot be applied to these systems due to their massive size. The authors handled these specific challenges through a combination of gradient ascent techniques with parameter merging to remove data without compromising overall performance levels. A performance evaluation revealed only minor degradation while attaining 78.5% accuracy rates for the retained data and significantly lower perplexity levels for the forgotten set. [38] also addressed catastrophic unlearning which happens when a system forgets necessary information through the optimization process which can be controlled during optimization steps. Their study proposed decentralized unlearning systems like federated unlearning as effective ways to extend these methods while preserving privacy and model effectiveness.

2.7.4 Unlearning in Generative Models

Generative models are designed to create new content given existing data sets by learning from patterns. But in some cases can hallucinate by adding objects and details that were not present in the input data, which can introduce misinformation and lower the reliability of modeled content in important applications including healthcare and safety systems [38]. Furthermore, these models generate inappropriate content that includes explicit and violent material revealing major ethical concerns along with potential risks.

To prevent hallucination of Generative models, [38] developed an Efficient Fine-Grained Unlearning Framework (EFUF) that avoids hallucination through CLIP-based image text matching without the need for paired datasets and at the reduced computational expense.CLIP classifies incorrect descriptions as hallucinated negatives and correct descriptions as non-hallucinated positives, helping refine the unlearning process. EFUF uses a gradient ascent with specialized loss functions to selectively unlearn hallucinated data less while preserving the natural sentence generation. They used the MSCOCO dataset using MiniGPT4 and LLaVA to show that EFUF leads to a 15% reduction in hallucinations and a 4% improvement in BLEU-1 score compared to baseline models. However, EFUF uses CLIP heavily for identifying images with text and has limitations of only detecting related mistakes but can not detect errors in attributes like color or object placement.

In another study, [28] proposed a Saliency-based Unlearning (SalUn) framework that effectively removes harmful and unwanted data without relying on CLIP. This framework uses gradient-based saliency models that select the most relevant features for modification which helps the model to make precise and efficient unlearning. The framework focuses on influential model weights rather than affecting the whole model weight and can remove harmful issues while producing reasonable results when applied to advanced generative models such as the Stable Diffusion model. The paper [28] showed that it outperformed baseline models including Erased Stable Diffusion and Forget-Me-Not for unlearning data both in terms of performance and unlearnability. SalUn maintained 94.56% accuracy in CIFAR-10 image classification, closely matching the original model with effective unlearning. However, this approach depends on fixed thresholds which might not be optimal for generating complex data.

2.8 Background Analysis on SISA Framework

After reviewing various unlearning methods, we found that the SISA framework is widely used and possible to do research on our current capibilites of hardware provided by the university. Therefore, we decided to analyze the SISA framework in detail to understand its working mechanism and effectiveness. To do that, we tried to replicate the results of the SISA paper using the MNIST dataset (as it was used in the original paper) and compared our results with the original paper.

2.8.1 MNIST Dataset

The MNIST dataset consists of 70,000 small grayscale images of handwritten digits from 0 to 9. Among those images there are 60,000 for training and 10,000 for testing. Each image is of 28x28 pixels in size, where each pixel has only a grayscale channel consisting of values ranging from 0 to 255. Analyzing previous research works, it can be concluded that MNIST Dataset is ideal for Support Vector Machines (SVMs), Multilayer Perceptrons (MLPs), and early Convolutional Neural Networks (CNNs). The MNIST Dataset requires negligible preprocessing work. The images can be normalized or standardized based on the dataset's average and standard deviation as per the model requirement. The digits are already centered and well-cropped. However, it is necessary to mention that as it's a very simple dataset, the model can get overfitted. Below is a sample of the dataset :

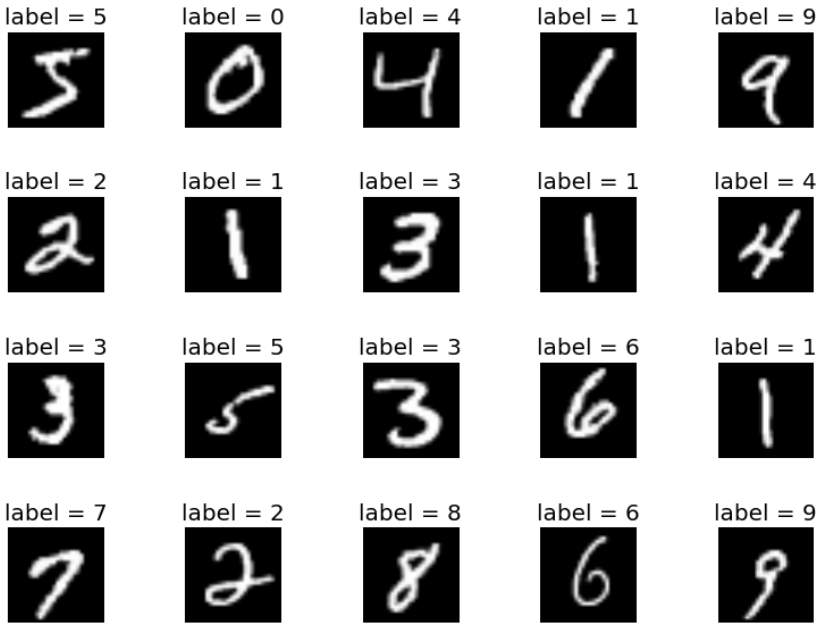

Figure 2.5: MNIST Dataset Sample

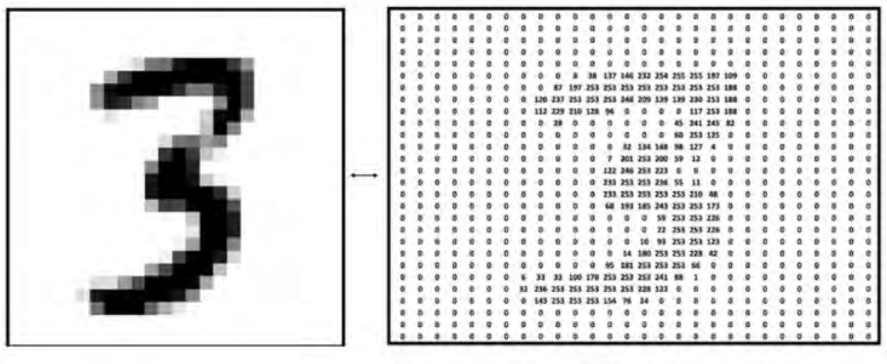

Figure 2.6: MNIST Dataset Pixel Sample

2.8.2 Data Preprocessing

After loading, the full dataset is divided into multiple parts in a process called sharding. The dataset is divided into three shards, each one will contain approximately 23,333 samples, and the last shard may include one extra sample to accommodate the full 70,000. As a result, the first shard will include indices from 0 to 23,332, the second from 23,333 to 46,665, and the third from 46,666 to 70,000. One limitation of this method is that it uses sequential order without class balancing. Therefore, early shards may have a different distribution of digit labels than later ones. For instance, if the beginning of MNIST has more samples of the digit 5, those will be more concentrated in the first shard.

- Shard 0: Indices 0–23,332 (23,333 samples)
- Shard 1: Indices 23,333–46,665 (23,333 samples)

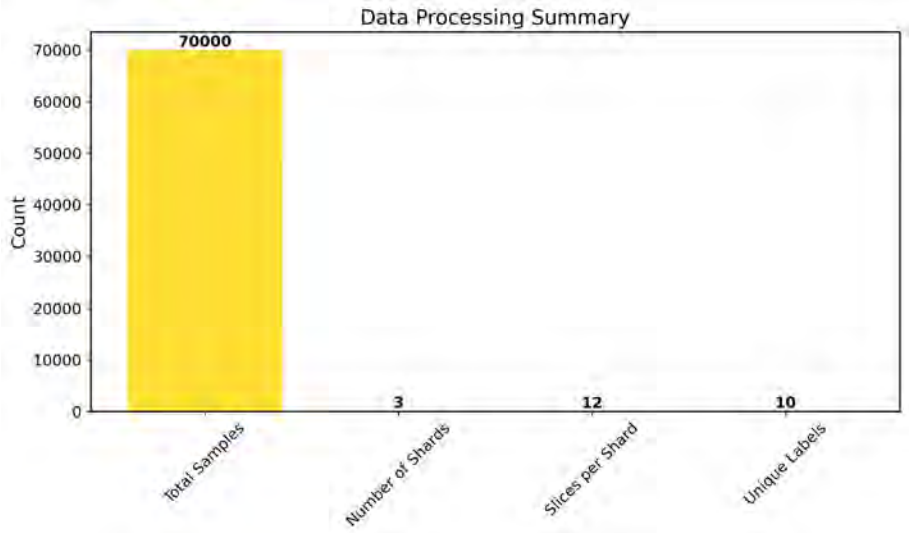

Figure 2.7: MNIST Processing Summary

- Shard 2: Indices 46,666–70,000 (23,334 samples)

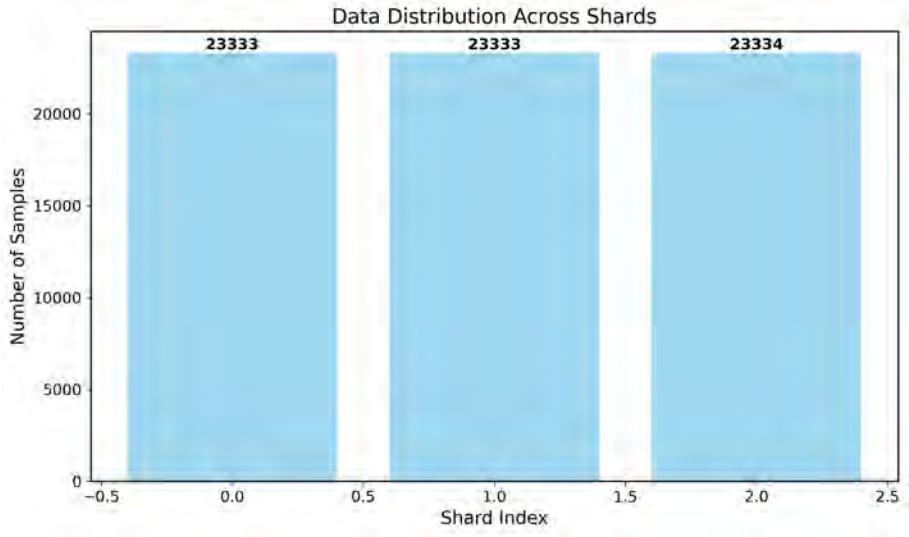

Figure 2.8: Shard Distribution

Each shard is then further divided into smaller groups called slices. These slices allow for incremental training. Each shard is divided into twelve slices; then, each slice would contain around 1,944 samples. Since 23,333 is not perfectly divisible by twelve, the last slice will include a few extra samples to account for the remainder, making it slightly larger than the others. This method ensures that training can proceed in small, manageable increments while maintaining the overall structure of the data.

Once sharding and slicing are complete, the processed data and its associated metadata are saved in two main file formats to support traceability and reproducibility. The first file is an NPZ file, which serves as a container for the actual data. This includes the image slices themselves. And on the other hand The second file is a JSON file that contains metadata about the dataset. This includes information

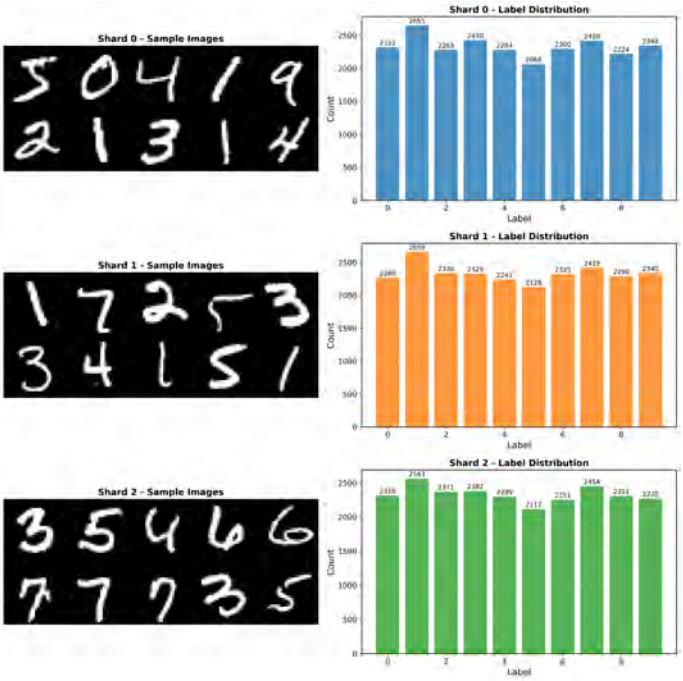

Figure 2.9: MNIST Shard Details

such as the total number of samples, the number of shards and slices, the shape of each image, and the distribution of labels across the dataset. Together, these two files provide a comprehensive snapshot of the processed MNIST dataset, ready for training and evaluation in machine learning tasks.

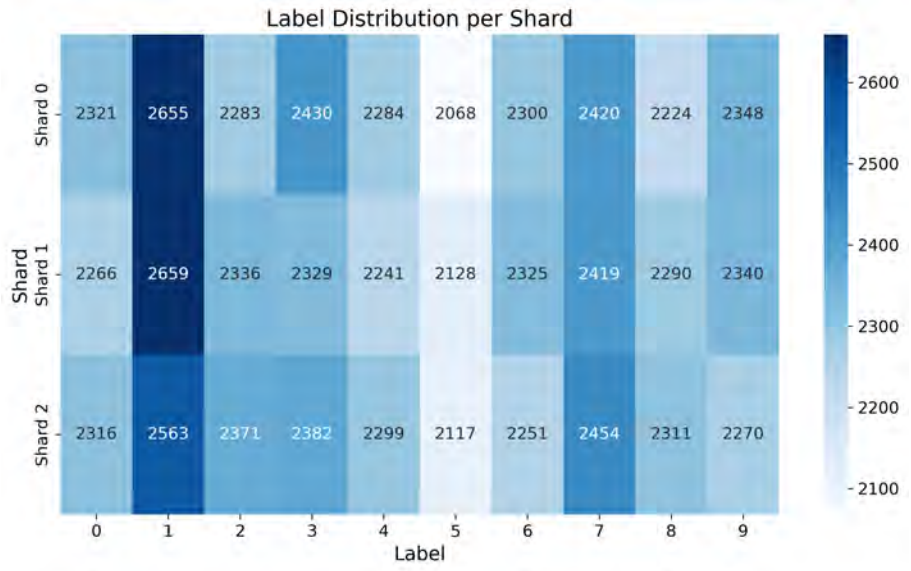

Figure 2.10: Label Distribution per Shard

2.8.3 Training

The SISA framework for training on the MNIST dataset uses a Convolutional Neural Network (CNN) designed for handwritten digit classification. The model architecture consists of two convolutional layers with 32 and 64 filters respectively, both

using ReLU activation functions. Each convolutional layer is followed by max pooling to reduce spatial dimensions. A dropout layer with 50% probability is included to prevent overfitting. The network connects to two fully connected layers, with the final layer containing 10 units corresponding to the 10 digit classes. The output layer applies a softmax function to produce class probabilities.

Before training, images are normalized by scaling pixel values from the original range of 0-255 to 0-1. The data is split into training and validation sets using an 80:20 ratio. The training process uses the Adam optimizer with weight decay for regularization. A learning rate scheduler reduces the learning rate when validation loss plateaus, while early stopping with patience of 5 epochs prevents unnecessary training iterations.

The training follows the SISA framework's incremental approach. For each shard, training proceeds slice by slice, with each slice building upon the knowledge learned from the previous slice. After each slice is trained, the model weights and training metrics are saved as checkpoints. This checkpoint system enables efficient unlearning by allowing selective retraining of only the affected portions when data needs to be removed.

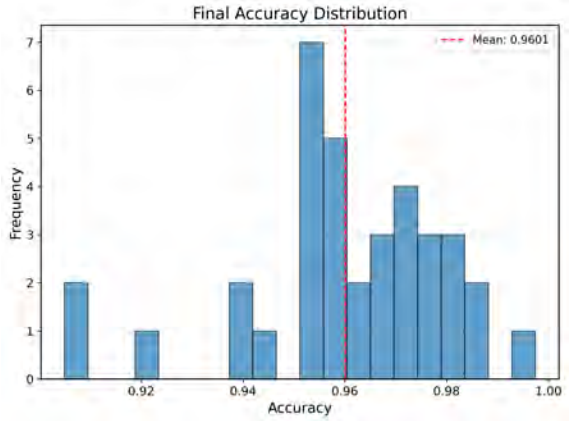

Figure 2.11: Accuracy Distribution per Shard

The initial training achieved an average validation accuracy of 96.01%, which meets the expected performance for MNIST using this model architecture.

2.8.4 Unlearning and Evaluation

The unlearning process consists of three phases: search, delete, and retrain. In the search phase, the system locates the data point to be removed by checking through the index lists of each shard and slice. Once found, it identifies the corresponding shard index, slice index, and position within the slice. In the deletion phase, the identified data point is removed from the dataset arrays and the modified data is saved.

In the retraining phase, the system loads the last valid model checkpoint before the affected slice. It then retrains the model from the affected slice onward, updating

each subsequent slice and saving new checkpoints. This selective retraining approach reduces computational overhead while ensuring the removed data no longer influences the model.

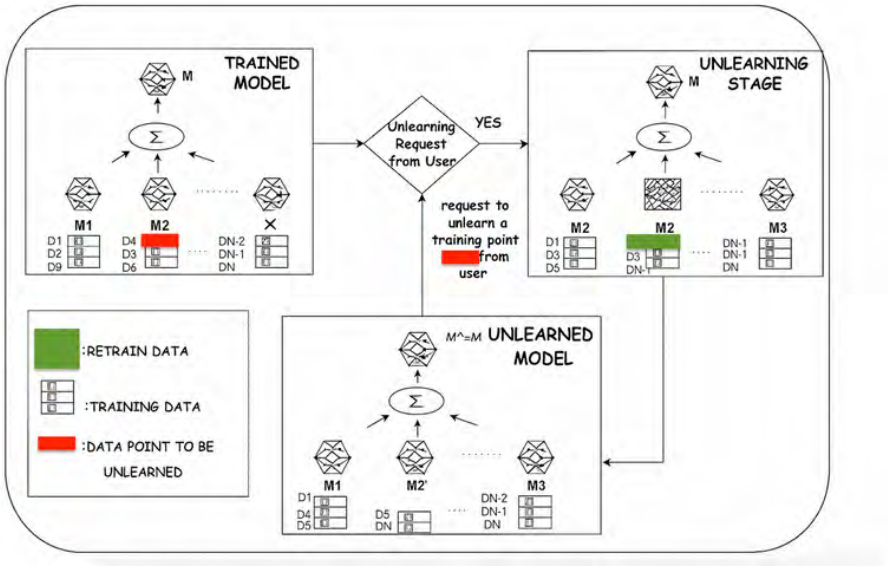

Figure 2.12: Unlearning Workflow

The unlearning performance was evaluated using several metrics. Search operations completed within 1-5 milliseconds, deletion operations required 100-500 milliseconds, and retraining took 5-20 seconds per slice. After performing an unlearning operation where all samples of digit "3" were removed from one slice, the model's accuracy dropped to 89.33%, representing a decrease of approximately 6-7% from the original accuracy. This performance loss was primarily due to the data imbalance introduced by removing an entire digit class from a single slice.

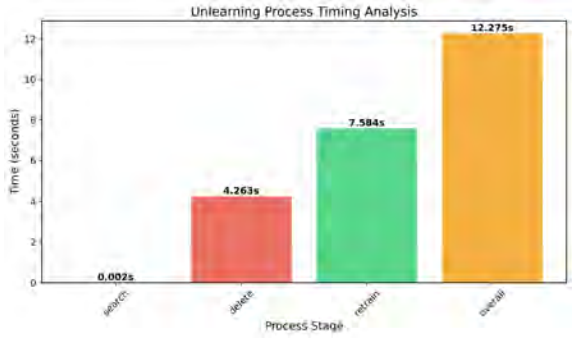

Figure 2.13: Unlearning Process Timing Analysis

2.9 Key Findings

Machine unlearning is essential for ensuring legal, ethical, security, environmental and economical feasibility in modern day AI systems. Regulations like GDPA and

CCPA mandate privacy preservation and the removal of data. However practical unlearning remains a challenge due to model's distributed storing of knowledge through billions of parameters. Unlearning methods can be broadly categorized into exact unlearning and approximate unlearning. Exact unlearning is computationally expensive while approximate unlearning is feasible but compromises model accuracy and performance due to residual data traces. Existing frameworks like SISA are useful in reducing overhead via sharded training while EMN and KGA optimizes unlearning through error-maximizing noise and knowledge alignment. Unlearning scenarios vary based on zero-glance, zero-shot and few-shot unlearning. Machine unlearning spans in the context of federated learning, lifelong learning, LLMs and generative models each facing diverse technical constraints. Generative models mitigate hallucination while preserving privacy through leveraging models like EFUF and SalUn. In a nutshell, no universal solution exists, opening research scopes to address scenario specific method selection to balance privacy, performance and practicality

Chapter 3

Work Plan

The research was divided into three main phases: Pre-Thesis 1 (P1), Pre-Thesis 2 (P2), and Final Thesis (P3). Each phase focused on different aspects of machine unlearning using the SISA framework.

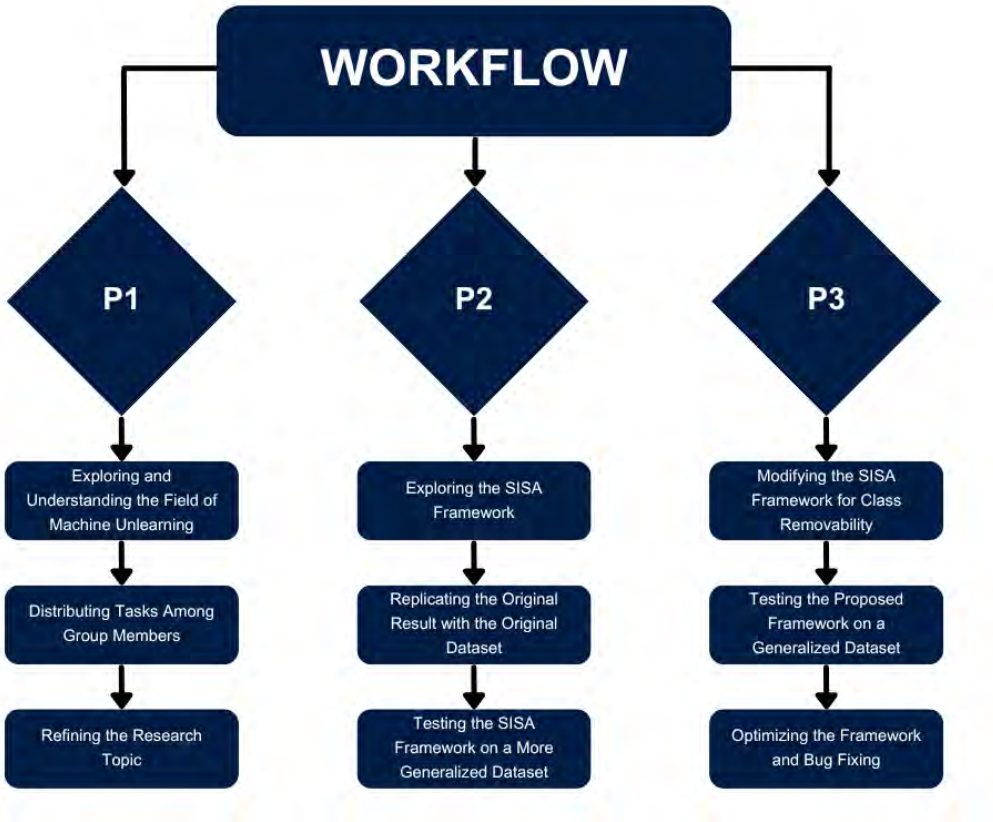

Figure 3.1: Work Plan

Phase 1: Pre-Thesis 1 (P1)

The first phase focused on building foundational knowledge. We started by exploring and understanding the field of machine unlearning through studying existing research papers. This helped us understand the current state of the field, identify research gaps, and learn about different approaches to data removal from trained

models. During this phase, we also distributed tasks among group members based on their strengths and interests. After understanding the field, we refined our research focus to class-level unlearning using the modified SISA framework.

Phase 2: Pre-Thesis 2 (P2)

The second phase focused on understanding and implementing the original SISA framework. We studied the original SISA framework in detail to understand how it works. We analyzed the architecture, including the concepts of sharding, isolation, slicing, and aggregation. After gaining theoretical understanding, we implemented the original SISA framework and tested it with the MNIST dataset to replicate the results from the original paper. This helped us verify our understanding of the framework. After that, we applied the original SISA framework to the CIFAR-10 dataset, which is more complex than MNIST. This step helped us understand how the framework performs on more realistic image data with multiple color channels and more diverse visual features.

Phase 3: Final Thesis (P3)

The final phase of this research is focused on modifying the SISA framework for class-level unlearning and evaluating its performance. We modified the original SISA framework to support class-level unlearning instead of just point-level unlearning. This involved changing how data is sharded and how models are retrained when an entire class needs to be removed. After implementing the modifications, we tested our modified SISA framework on the CIFAR-10 dataset to evaluate its effectiveness in removing entire classes. We measured the impact on model accuracy, computational efficiency, and privacy guarantees. Finally, we refined our implementation by fixing bugs, optimizing performance, and improving the efficiency of the class unlearning process. We also conducted additional experiments on other Generalized datasets to validate the robustness of our approach.

Chapter 4

Dataset Analysis and Preprocessing

4.1 Dataset Selection and Characteristics

For this research, we have selected the CIFAR-10 dataset as the primary benchmark for implementing and evaluating the modified SISA framework for class-level unlearning. CIFAR-10 was created in 2009 by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton at the University of Toronto and has since become one of the most widely used datasets in computer science research. The dataset contains 60,000 color images distributed across 10 mutually exclusive object classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image measures 32×32 pixels with three RGB color channels, resulting in 3,072 values per image.

Figure 4.1: CIFAR-10 Dataset Sample

CIFAR-10 occupies a critical middle ground in the landscape of computer vision benchmarks. Unlike MNIST, which has become too simple for modern research with models achieving over 99% accuracy, CIFAR-10 provides a meaningful challenge while remaining computationally accessible. At the same time, it avoids the computational demands of ImageNet, which requires hours or days of training time. This balance makes CIFAR-10 ideal for rapid prototyping and experimentation, allowing us to train complete models in minutes rather than days. This computational

efficiency is particularly important for us as we have limited resources for conducting the machine unlearning research.

The low resolution of 32×32 pixels presents both challenges and opportunities for our research. While this resolution limits the visual information available and creates some inherently ambiguous samples, it also ensures that our unlearning framework works with limited data representations. This constraint mirrors real-world scenarios where the companies often work with compressed or low-resolution data. The dataset's careful manual annotation ensures label reliability, which is critical for evaluating the effectiveness of unlearning.

4.2 Data Preprocessing Pipeline

Our preprocessing pipeline follows the traditional approach used in modern CIFAR-10 research while incorporating specific requirements for the SISA framework. The first step involves converting images from their original format to PyTorch format. This conversion transforms image dimensions from Height×Width×Channels (HWC) to Channels×Height×Width (CHW), which is the format expected by PyTorch's convolutional layers. After the conversion, we apply normalization using dataset-specific statistics computed across all training images for stable gradient flow during training.

4.3 Data Partitioning Strategy

The SISA framework requires a sophisticated data partitioning strategy that divides the training data into shards and slices. Our implementation follows a two-phase approach: first creating shards through class-based assignment, then dividing each shard into sequential slices. This partitioning strategy is fundamental to enabling efficient class-level unlearning, as it determines how much of the model needs to be retrained when a class is removed.

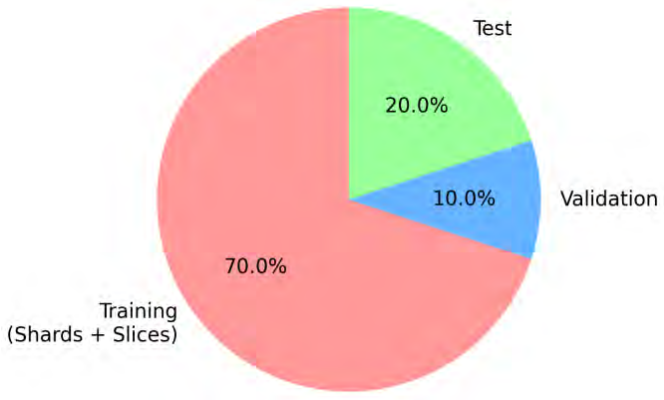

Figure 4.2: Dataset Distribution

We split the original 60,000 images using a 70-10-20 ratio, allocating 70% for train-

ing (42,000 images), 10% for validation (6,000 images), and 20% for testing (12,000 images). This split uses stratified sampling to ensure that each subset maintains proportional representation of all classes. The validation set is used for hyperparameter tuning and early stopping, while the test set provides an unbiased evaluation of model performance after training and unlearning operations.

4.4 Sharding Implementation

The sharding phase divides the training data so that each class goes to exactly one shard. This means a class is never split across multiple shards. This design is important for our class-level unlearning approach. When we need to remove a specific class, we only retrain the one shard that contains that class, not all shards. This saves significant time and computational resources.

The algorithm works in a simple way. First, it groups all training data by class and counts how many samples each class has. Then it sorts the classes from largest to smallest. After that, it assigns each class to a shard one by one. When assigning a class, the algorithm tries to keep all shards roughly the same size. This is called load balancing, and it prevents one shard from being much larger than others.

To decide which shard gets a class, the algorithm checks what would happen if that class was added to each shard. It calculates the imbalance ratio, which is simply the size of the largest shard divided by the size of the smallest shard. The class goes to whichever shard keeps this ratio smallest. Figure 4.3 shows the result of this sharding process for our CIFAR-10 dataset using two shards. The left panel shows how the 10 classes are distributed across the two shards, with each class assigned to exactly one shard. Five classes are grouped into Shard 1, and the remaining five classes are grouped into Shard 2. The right panel shows the total number of samples in each shard. Both shards contain 21,000 samples, which corresponds to five classes with 4,200 images each. This perfectly balanced distribution keeps the imbalance ratio at 1.0, ensuring that training time remains consistent across shards when we run them in parallel.

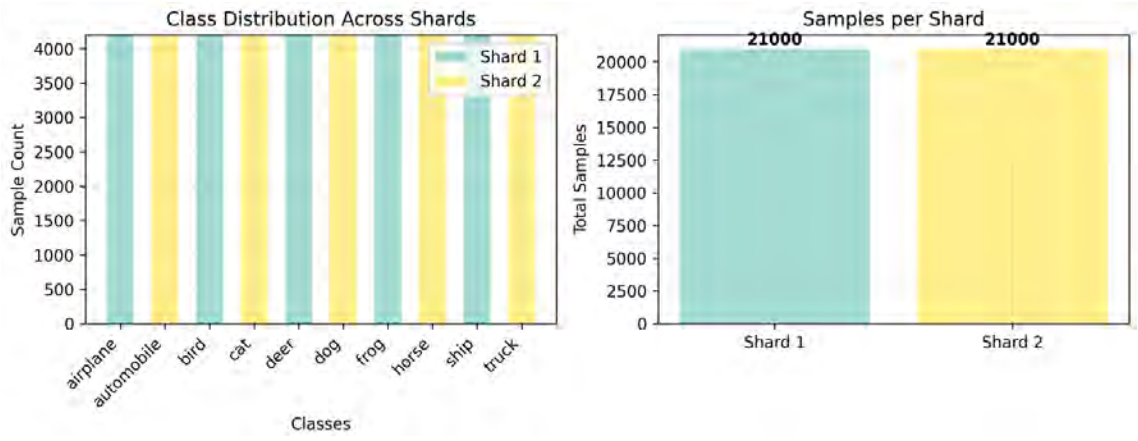

Figure 4.3: Class Distribution and Samples per Shard

4.5 Slicing Implementation

After sharding, each shard is divided into smaller sequential portions called slices. This slicing process is different from sharding. While sharding assigns complete classes to shards, slicing simply divides the data in each shard into equal-sized portions for incremental training.

The algorithm first calculates how big each slice should be. It divides the total number of samples in a shard by the number of slices we want. If there is a remainder, the first few slices get one extra sample. For example, if a shard has 8,000 samples and we want 3 slices, the first slice gets 2,667 samples, and the other two slices each get 2,666 samples.

The filling process is straightforward. It starts with the first slice and fills it with samples from the first class in the shard. It continues adding samples from that class until either the slice is full or all samples from that class are used up. If the slice gets full, the algorithm moves to the next slice and continues adding the remaining samples from the same class. If the class runs out of samples, the algorithm stays in the current slice and starts adding samples from the next class.

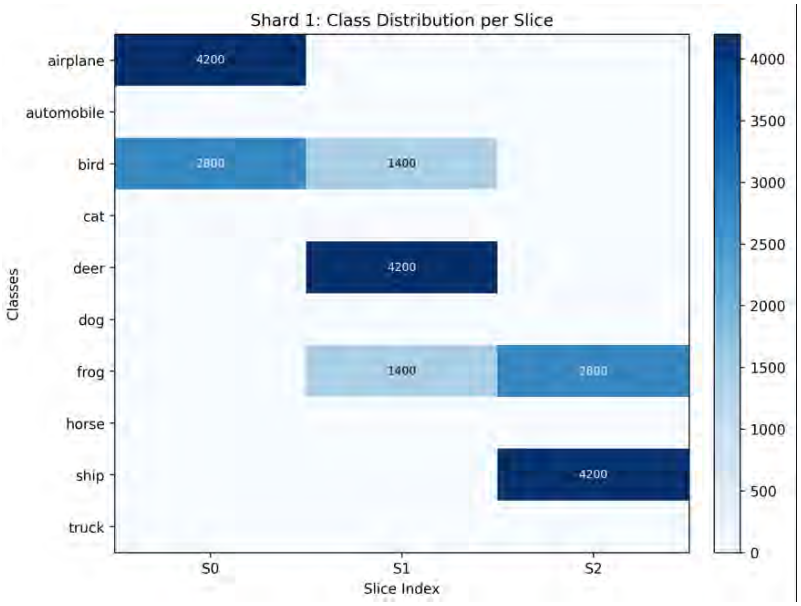

Figure 4.4: Class Distribution per Slice on Shard 1

Figure 4.4 illustrates this sequential filling for Shard 1, which now contains five classes with 4,200 samples each. The shard is divided into three slices (S0 to S2), with each slice holding exactly 7,000 samples. Slice S0 contains all airplane samples together with the first 2,800 bird samples, Slice S1 continues with the remaining 1,400 bird samples, all deer samples, and the first 1,400 frog samples, while Slice S2 completes the shard with the remaining 2,800 frog samples and all ship samples. This pattern shows how the algorithm keeps samples from the same class together as much as possible while filling each slice to capacity.

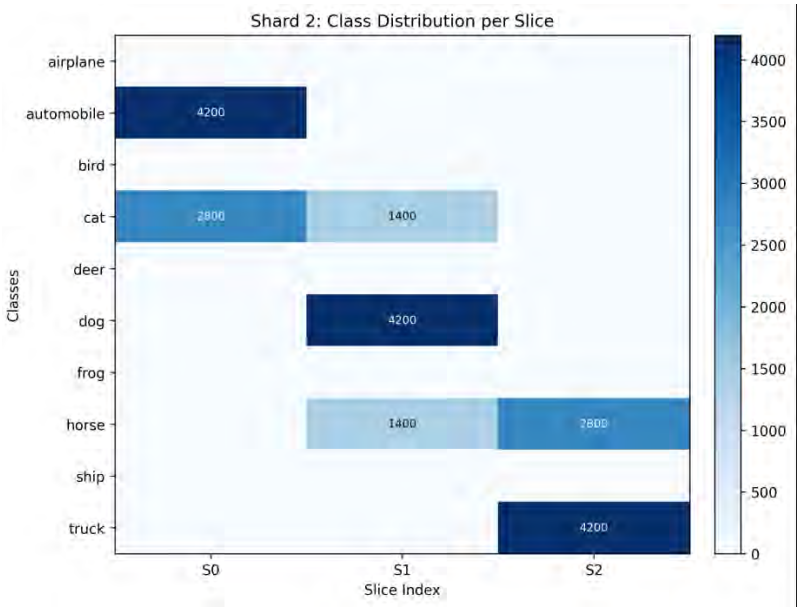

Figure 4.5: Class Distribution per Slice on Shard 2

Figure 4.5 shows the distribution for Shard 2, which also contains five classes with 4,200 samples each, divided into three slices of 7,000 samples. Slice S0 holds all automobile samples and the first 2,800 cat samples. Slice S1 continues with the remaining 1,400 cat samples, all dog samples, and the first 1,400 horse samples. Slice S2 completes the shard with the remaining 2,800 horse samples and all truck samples. This organization demonstrates how the sequential filling approach adapts when slice boundaries do not perfectly align with class sizes, while still enabling incremental training across slices.

4.6 Data Persistence and Organization

After processing, all data is saved in NumPy binary format, known as NPY files. These files load quickly and take up less storage space. We organize the data into separate folders for shards, slices, validation data, and test data. Each shard has its own folder, and inside that folder, each slice has three separate files: one for images, one for labels, and one for the original index numbers. This organization is helpful because we can load just the data we need for training a specific shard or slice, instead of loading everything at once. This saves memory and makes training faster.

We also save important information about the data in JSON files, which are easy to read. This metadata includes the normalization statistics we calculated from the training data. Saving these statistics ensures that we use the same preprocessing settings every time we train a model. The metadata also records which classes are in which shards, and suggests good batch sizes for training.

4.7 Implementation Considerations

Our implementation focuses on using memory and time efficiently. We use stratified sampling when splitting the data, which means each split has the same proportion of each class. This balanced distribution is important because it ensures fair testing when we evaluate our unlearning operations.

The design is flexible and easy to configure. We can change important settings like the number of shards, the number of slices in each shard, and how we split the data between training, validation, and testing. All these settings are stored in one configuration file that is easy to edit. This flexibility allows us to experiment with different ways of organizing the data to find the best balance between fast unlearning and good model performance. Another advantage of our approach is that we can train multiple shards at the same time on different GPUs. This parallel training significantly reduces the total time needed for both initial training and retraining when we need to unlearn data.

Chapter 5

Training and Model Development

This section outlines the progressive development of our models, beginning with a baseline Convolutional Neural Network (CNN) and gradually extending toward increasingly sophisticated variants of the SISA framework. Each stage in this development pipeline was guided by specific limitations identified in the previous stage, leading to methodical architectural and procedural enhancements.

Initially, the baseline CNN served as a reference point for evaluating both classification accuracy and retraining time. However, this approach exhibited clear inefficiencies during unlearning operations: the removal of an entire class required retraining the entire model $\mathcal{M}$ on a reduced dataset $\mathcal{D}_u \setminus \mathcal{C}_x$, resulting in significant computational overhead (see Section 5 for a formal unlearning formulation).

To address these inefficiencies, we explored the SISA (Sharded, Isolated, Sliced, and Aggregated) learning paradigm, which partitions the dataset into smaller, structured subsets to localize retraining to specific model components [14]. Formally, the original dataset $\mathcal{D}_u$ was divided into K disjoint shards $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$ such that

$$\mathcal{D}_u = \bigcup_{k=1}^{K} \mathcal{S}_k, \quad \mathcal{S}_i \cap \mathcal{S}_j = \emptyset \quad \forall i \neq j. \tag{5.1}$$

Each shard $\mathcal{S}_k$ was further partitioned into slices $\{\mathcal{S}_{k,1}, \mathcal{S}_{k,2}, \ldots, \mathcal{S}_{k,L_k}\}$, and an independent constituent model $\mathcal{M}_k$ was trained sequentially on these slices with checkpointing after each stage.

Through successive iterations, we introduced several key modifications to the standard SISA structure to improve unlearning efficiency and preserve model performance:

- Shard and Slice Configuration: Initial integration of sharding and balanced slicing to reduce retraining scope.
- Sequential Class Slicing: Redesigning slice allocation to group entire classes within specific slices, enabling class-level unlearning without full retraining.
- Reinforced Replay Mechanism: Introducing a replay buffer to mitigate catastrophic forgetting during sequential training, improving accuracy while maintaining efficiency.

- Gating Network Integration: Employing a lightweight CNN-based gating network $\mathcal{G}_\phi$ to route inputs to the most relevant constituent model, thereby enhancing prediction accuracy and reducing inference cost.

At each development stage, we jointly present model architecture and training methodology, enabling a structured comparison of how incremental design changes influence learning dynamics, retraining latency, and empirical performance. Corresponding experimental outcomes, including accuracy, training time, and retraining time, are reported in Chapter 7.

5.1 Baseline Model

The baseline architecture is a single Convolutional Neural Network parameterized by θ, denoted $\mathcal{M}_\theta$, trained on the full dataset $\mathcal{D}_u$. Let

$$\mathcal{D}_u = \{(x_i, y_i)\}_{i=1}^m, \qquad y_i \in \mathcal{Y}, \quad |\mathcal{Y}| = C, \tag{5.2}$$

where x_i is an input image and y_i its class label. The baseline model learns the mapping

$$\mathcal{M}_\theta : \mathbb{R}^{H\times W\times 3} \to \Delta^{C-1}, \tag{5.3}$$

where Δ^{C-1} is the probability simplex over C classes.

Training the baseline involves minimizing the empirical categorical cross-entropy loss over $\mathcal{D}_u$:

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{CE}}(\mathcal{M}_\theta, \mathcal{D}_u) = \arg\min_\theta -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C}\{y_i = c_j\}\log p(y = c_j \mid x_i), \tag{5.4}$$

where $p(y \mid x)$ denotes the class probability predicted by $\mathcal{M}_\theta$.

A principal drawback of this design in the context of machine unlearning is that removing a class $c^* \in \mathcal{Y}$ requires forming the reduced dataset

$$\mathcal{D}_{u\setminus c^*} = \{(x_i, y_i) \in \mathcal{D}_u \mid y_i \neq c^*\}, \tag{5.5}$$

and retraining the model from scratch:

$$\theta^*_{-c^*} = \arg\min_\theta \mathcal{L}_{\text{CE}}(\mathcal{M}_\theta, \mathcal{D}_{u\setminus c^*}). \tag{5.6}$$

This full-retraining procedure is computationally expensive and motivates the partition-based SISA variants described in subsequent sections.

5.1.1 Training Methodology

All training uses the same preprocessing and optimization conventions adopted across the thesis to ensure consistency: Preprocessing

- Resize inputs to fixed dimensions (H, W) to match the CNN input.

- Normalize pixel values to $[0, 1]$ (or standardize with dataset mean/std as appropriate).
- (Optionally) apply standard augmentation (random crop/flip) during training — if used, specify in experiments.

Optimization

- Optimizer: Adam with learning rate η and standard moment estimates where $\theta(t)$ denotes the parameters at t. The parameter update follows

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{5.7}$$

- Loss: categorical cross-entropy $\mathcal{L}_{\text{CE}}$ as above.
- Batch size: B (chosen to balance stability and runtime).
- Early stopping / model selection: monitor a held-out validation set $\mathcal{D}_{\text{val}}$; select the checkpoint with best validation accuracy or lowest validation loss to avoid overfitting.
- Number of epochs: chosen so that validation metrics converge.

Monitoring & Evaluation

- Evaluate validation metrics (accuracy, loss) at the end of each epoch.
- Report both final test accuracy and wall-clock training time $T_{\text{train}}(\mathcal{M}_\theta, \mathcal{D}_u)$ so the baseline provides a precise reference for retraining cost comparisons in later SISA variants.

5.2 SISA Framework with Balanced Class Slicing

To improve retraining efficiency, we adopted the SISA (Sharded, Isolated, Sliced, Aggregated) framework by partitioning the original dataset $\mathcal{D}_u$ into K shards. Each shard was assigned its own independent CNN model $\mathcal{M}_{\theta_k}$, enabling distributed training across multiple smaller networks rather than relying on a single large model. During inference, the predictions from these shard-specific models were aggregated to produce the final output.

Formally, the dataset was partitioned as per Equation (5.1), where $\mathcal{S}_k$ denotes the dataset assigned to shard k. Each shard $\mathcal{S}_k$ contained a balanced subset of classes, i.e.,

$$P(y \mid x \in \mathcal{S}_k) \approx P(y), \quad \forall k, \tag{5.8}$$

to prevent any shard-specific model from being biased toward certain classes.

Each shard was trained independently, resulting in a set of constituent models:

$$\{\mathcal{M}_{\theta_1}, \mathcal{M}_{\theta_2}, \ldots, \mathcal{M}_{\theta_K}\}. \tag{5.9}$$

For a test input x, each shard-specific model produced a class probability vector over the classes it was trained on:

$$\mathbf{p}_k(x) = \mathcal{M}_{\theta_k}(x), \tag{5.10}$$

and the final prediction was determined via aggregation using the argmax operator:

$$\hat{y}(x) = \arg\max_{c \in \mathcal{Y}} \max_{k \in \{1,\ldots,K\}} \mathbf{p}_k^{(c)}(x), \tag{5.11}$$

where $\mathbf{p}_k^{(c)}(x)$ denotes the probability assigned to class c by shard model k. In other words, the prediction from the shard exhibiting the highest confidence score determined the final class label.

This sharded design offers a crucial efficiency advantage during unlearning. When a request to remove class c^* is received, only the shard $\mathcal{S}_k$ that contains c^* needs to be updated. The unlearning process involves removing all samples belonging to c^* from $\mathcal{S}_k$, forming $\mathcal{S}_{k \setminus c^*}$, and retraining only the corresponding model $\mathcal{M}_{\theta_k}$ on this reduced shard:

$$\theta_k^* \leftarrow \arg\min_{\theta_k} \mathcal{L}_{\text{CE}}(\mathcal{M}_{\theta_k}, \mathcal{S}_{k \setminus c^*}). \tag{5.12}$$

All other shard models remain untouched. This localized retraining results in a significant reduction in total retraining time compared to the baseline, where the entire model was retrained on $\mathcal{D}_{u \setminus c^*}$.

Although this partitioning slightly reduced overall generalization capacity—since each model only had access to a fraction of the data—it achieved a substantial gain in retraining efficiency, making it a strong step forward in unlearning-oriented architecture design.

5.2.1 Training Methodology

For this framework, the partitioning of $\mathcal{D}_u$ into shards was performed such that each shard $\mathcal{S}_k$ represented a balanced sample of all classes. This ensured that every shard was a good approximation of the global data distribution, reducing shard bias while maintaining the benefits of isolation.

Each shard-specific model $\mathcal{M}_{\theta_k}$ was trained independently, but with the same training configuration used for the baseline model to ensure a controlled comparison:

- Optimizer: Adam with learning rate η.
- Loss: Categorical cross-entropy.
- Batch size and number of epochs: kept identical across all shards.
- Preprocessing: resizing, normalization, and optional augmentations were applied consistently across all shards.

After training, the inference process used the aggregation scheme described earlier. Each shard model outputted its probability distribution $\mathbf{p}_k(x)$, and the shard with

the maximum confidence determined the final class label through a global argmax operation. This ensemble-like behavior allowed the system to maintain competitive predictive performance while enabling localized unlearning, thus achieving a more favorable trade-off between accuracy and retraining cost compared to the baseline model.

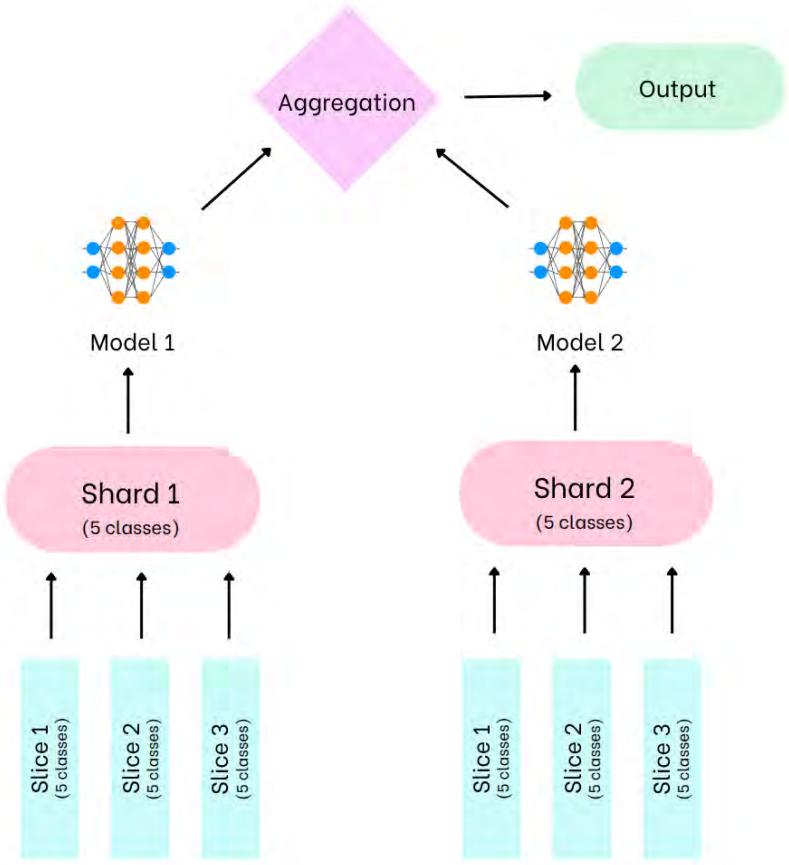

Figure 5.1: Balanced Class Slicing Model Architecture

5.3 SISA Framework with Sequential Class Slicing

This version of the SISA framework extends the shard-based structure by introducing semi slice-level isolation. After dividing the dataset into shards, each shard is further partitioned into multiple smaller slices, each containing a minimal subset of classes arranged in a sequential order.

Let the original dataset be denoted as per class distribution

$$\mathcal{D} = \bigcup_{c=1}^{C} \mathcal{D}_c, \tag{5.13}$$

where $\mathcal{D}_c$ represents the set of samples belonging to class c, and C is the total number of classes.

The dataset is divided into K shards per Equation (5.1):

$$\mathcal{D} = \bigcup_{s=1}^{S} \mathcal{D}^{(s)}, \tag{5.14}$$

where $\mathcal{D}^{(s)} \cap \mathcal{D}^{(s')} = \emptyset$ for all $s \neq s'$.

For each shard $\mathcal{D}^{(s)}$, we introduce a slicing mechanism that partitions it into $L^{(s)}$ class-based slices:

$$\mathcal{D}^{(s)} = \bigcup_{\ell=1}^{L^{(s)}} \mathcal{D}_{\ell}^{(s)}, \tag{5.15}$$

where each slice $\mathcal{D}_{\ell}^{(s)}$ contains samples from a minimal number of classes and

$$\mathcal{D}_{\ell}^{(s)} \cap \mathcal{D}_{\ell'}^{(s)} = \emptyset \quad \text{for all} \quad \ell \neq \ell'. \tag{5.16}$$

5.3.1 Training Methodology

For each shard s, a separate CNN model $M^{(s)}$ is trained incrementally across its slices in sequential order:

$$M_{\ell}^{(s)} = \text{Train}\big(M_{\ell-1}^{(s)}, \mathcal{D}_{\ell}^{(s)}\big), \quad \ell = 1, 2, \ldots, L^{(s)}, \tag{5.17}$$

where $M_0^{(s)}$ represents the initial randomly initialized model. After the training of each slice, the model parameters $\theta_{\ell}^{(s)}$ are stored through checkpointing:

$$\theta_{\ell}^{(s)} = \text{Checkpoint}\big(M_{\ell}^{(s)}\big), \tag{5.18}$$

allowing the framework to preserve the learned state after each slice. This checkpointing mechanism later enables efficient rollback without retraining the entire model from scratch.

The same optimizer (Adam), loss function (categorical cross-entropy), learning rate, and batch size are used as in previous frameworks to ensure a fair performance comparison.

5.3.2 Inference

During inference, an input $x \in \mathcal{X}$ is passed through all shard models $\{M^{(s)}\}_{s=1}^{S}$. Each shard model produces a probability distribution $p^{(s)}(y \mid x)$ over its local classes. The final prediction is obtained by aggregating all shard outputs using an argmax operation:

$$\hat{y} = \arg\max_{y} \max_{s} \; p^{(s)}(y \mid x). \tag{5.19}$$

5.3.3 Catastrophic Forgetting

A major drawback of this sequential slicing structure is catastrophic forgetting. Since each slice contains disjoint classes, and training occurs sequentially without revisiting previous slices, the model tends to forget earlier knowledge as it progresses

through later slices.

Let $\mathcal{A}_\ell$ denote the accuracy on classes from slice ℓ, and $\mathcal{A}_{\leq k}$ the accuracy on all slices up to k. As training advances to slice n,

$$\mathcal{A}_{\leq n-1} \downarrow \quad \text{as} \quad n \uparrow, \tag{5.20}$$

indicating a consistent decay in retention of earlier knowledge. This phenomenon limits the model's overall accuracy and motivates the introduction of replay mechanisms in subsequent versions of the framework to mitigate forgetting while maintaining computational efficiency.

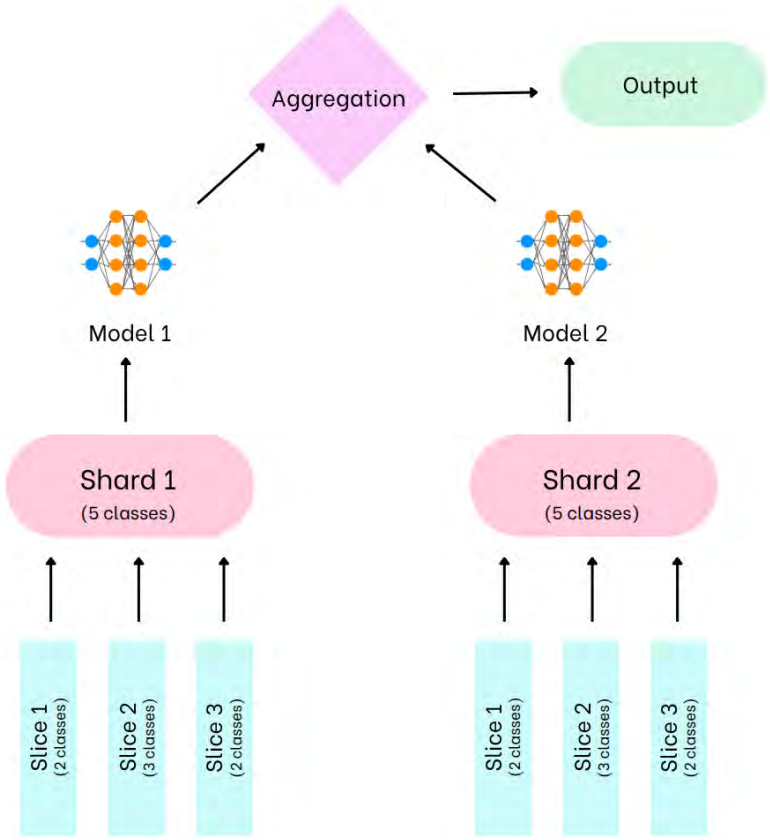

Figure 5.2: Sequential Class Slicing Model Architecture

5.4 SISA Framework with Sequential Class Slicing and Replay Mechanism

To address catastrophic forgetting observed in the previous sequential slice-based framework, a replay mechanism was incorporated into the slice-isolated SISA architecture. In this setup, the model no longer trains solely on the current slice but also revisits a subset of samples from previously trained slices at each training step. This approach allows the model to reinforce prior knowledge while learning new information, improving stability and preventing rapid degradation of earlier learning.

Let $\mathcal{D}_\ell^{(s)}$ denote the data in slice ℓ of shard s. For a given slice $\ell > 1$, a replay subset $\mathcal{R}_\ell^{(s)} \subseteq \bigcup_{j=1}^{\ell-1} \mathcal{D}_j^{(s)}$ is sampled. The size of $\mathcal{R}_\ell^{(s)}$ is determined by a replay ratio $\rho \in (0, 1)$:

$$|\mathcal{R}_\ell^{(s)}| = \rho \cdot \sum_{j=1}^{\ell-1} |\mathcal{D}_j^{(s)}|. \tag{5.21}$$

The model $M_\ell^{(s)}$ is then trained on the union of the current slice and the replay data:

$$\theta_\ell^{(s)} = \arg\min_\theta \mathcal{L}\Big(\theta; \mathcal{D}_\ell^{(s)} \cup \mathcal{R}_\ell^{(s)}\Big), \tag{5.22}$$

where $\mathcal{L}$ is the cross-entropy loss:

$$\mathcal{L}(\theta; \mathcal{B}) = -\frac{1}{|\mathcal{B}|} \sum_{(x,y)\in\mathcal{B}} \log p_\theta(y \mid x), \tag{5.23}$$

and $\mathcal{B}$ is the training batch containing both current and replayed data.

5.4.1 Checkpointing

As in previous frameworks, checkpoints of the model parameters are stored at the end of training for each slice:

$$\theta_\ell^{(s)} = \text{Checkpoint}(M_\ell^{(s)}). \tag{5.24}$$

These checkpoints enable efficient rollback during unlearning without retraining from scratch.

5.4.2 Training Methodology

Training proceeds incrementally across slices, with the key modification that each slice $\ell > 1$ includes the replay subset $\mathcal{R}_\ell^{(s)}$ sampled from previously trained slices. The replay data is dynamically refreshed throughout training to maintain diversity and prevent bias toward older slices.

The model uses the same optimizer, learning rate, batch size, and loss function as in previous SISA variants to ensure that improvements in performance are attributable to the replay mechanism rather than changes in training hyperparameters.

Replay Ratio	**Accuracy**	**Unlearning Acc.**	**Main Model Time (s)**
20%	69.7%	67.8%	52.7
30%	**73.1%**	**70.7%**	**55.4**
40%	73.9%	71.1%	58.2

Table 5.1: Performance Comparison of Different Replay Ratios (2 Shards, 5 Slices)

To determine the optimal replay ratio, we evaluated several configurations across our two-shard, five-slice setup. As shown in Table 5.1, a replay ratio of 30% achieves 73.1% accuracy with minimal training overhead (55.4 seconds), representing a favorable balance between forgetting mitigation and computational cost. While 40%

replay yields marginally higher accuracy (73.9%), the additional 5% training time does not justify the modest 0.8% accuracy improvement. The 20% ratio, though faster, results in substantially lower accuracy (69.7%), indicating insufficient reinforcement of prior knowledge. Therefore, we adopt 30% replay as the standard configuration for all subsequent experiments.

This replay-augmented slice training effectively mitigates catastrophic forgetting, maintaining higher accuracy across all slices while still benefiting from the reduced retraining time enabled by sequential class slicing.

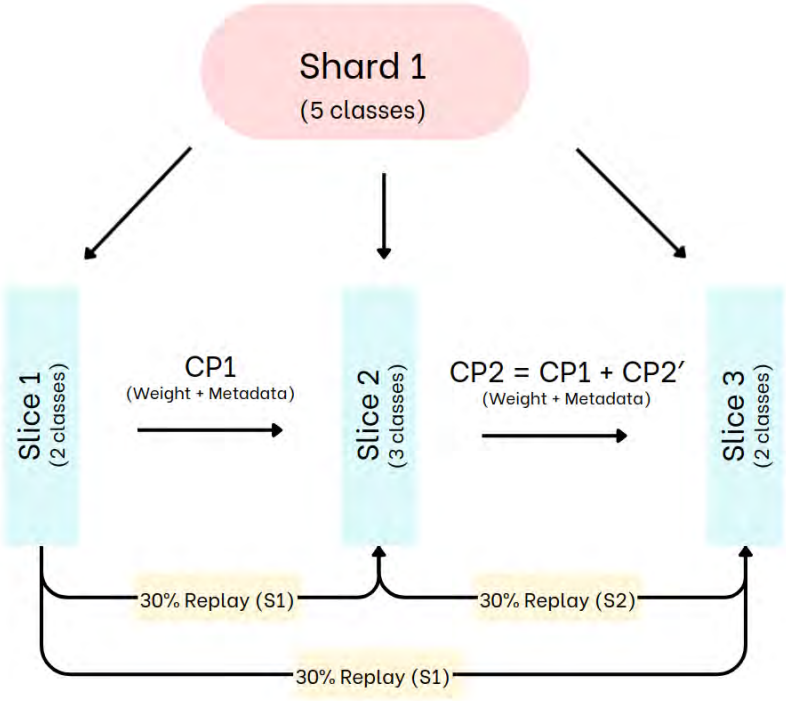

Figure 5.3: Replay and Checkpointing Mechanism

5.5 SISA Framework with Replay Mechanism and Gating Network

The final version of the SISA framework incorporates a gating network to improve prediction accuracy and leverage shard specialization. The gating network functions as a router, determining which shard-specific CNN model is most appropriate for processing a given input.

Let the set of shard-specific CNN models be

$$\{M^{(s)}\}_{s=1}^{S}, \tag{5.25}$$

where S is the total number of shards. Each shard model $M^{(s)}$ is trained using the replay-augmented sequential slice methodology described in Section 4.

5.5.1 Gating Network

A separate CNN-based gating network $G(\cdot;\phi)$ is trained to map input samples $x \in \mathcal{X}$ to shard indices $s \in \{1,\ldots,S\}$:

$$G : x \mapsto s, \quad s = \arg\max_{j\in\{1,\ldots,S\}} p_\phi(s = j \mid x), \tag{5.26}$$

where $p_\phi(s \mid x)$ is the probability assigned by the gating network to shard s given input x.
During training, the targets for the gating network are shard identifiers rather than class labels. The loss function for training G is categorical cross-entropy:

$$\mathcal{L}_G(\phi) = -\frac{1}{|\mathcal{D}|}\sum_{(x,s)\in\mathcal{D}} \log p_\phi(s \mid x), \tag{5.27}$$

where $\mathcal{D}$ is the training dataset with inputs x and **corresponding shard labels** s. The network is optimized using an adaptive optimizer (e.g., Adam) to learn the mapping from input features to shard selection.

5.5.2 Inference Procedure

Given an input $x \in \mathcal{X}$, inference proceeds in two stages:

1. **Shard selection:** The gating network predicts the most appropriate shard:

$$\hat{s} = \arg\max_{s\in\{1,\ldots,S\}} p_\phi(s \mid x). \tag{5.28}$$

2. **Class prediction:** The input x is forwarded to the selected shard model $M^{(\hat{s})}$, which outputs the class probability distribution:

$$\mathbf{p}^{(\hat{s})}(y \mid x) = M^{(\hat{s})}(x;\theta^{(\hat{s})}), \tag{5.29}$$

and the final predicted class is

$$\hat{y} = \arg\max_y \mathbf{p}^{(\hat{s})}(y \mid x). \tag{5.30}$$

This routing mechanism improves accuracy by directing each input to the shard that is most likely to have specialized knowledge for that input, while also reducing unnecessary computation by avoiding evaluation of all shard models.

5.5.3 Training Methodology

Training proceeds in two stages:

- **Shard-specific CNN training:** Each $M^{(s)}$ is trained incrementally using the replay-augmented sequential slicing method, combining the current slice with a portion of replayed samples from earlier slices. Checkpoints are saved at the end of each slice to preserve learned weights. This ensures each shard maintains stability and minimizes catastrophic forgetting.

- **Gating network training:** After the shard models are trained, the gating network G is trained with shard identifiers as targets. The same preprocessing (normalization, resizing, etc.) and optimization setup (cross-entropy loss, adaptive optimizer) as the shard models is applied to maintain consistency.

While adding the gating network slightly increases training time due to the additional network, it significantly improves the overall accuracy and computational efficiency, as only the selected shard is activated during inference rather than all shard models.

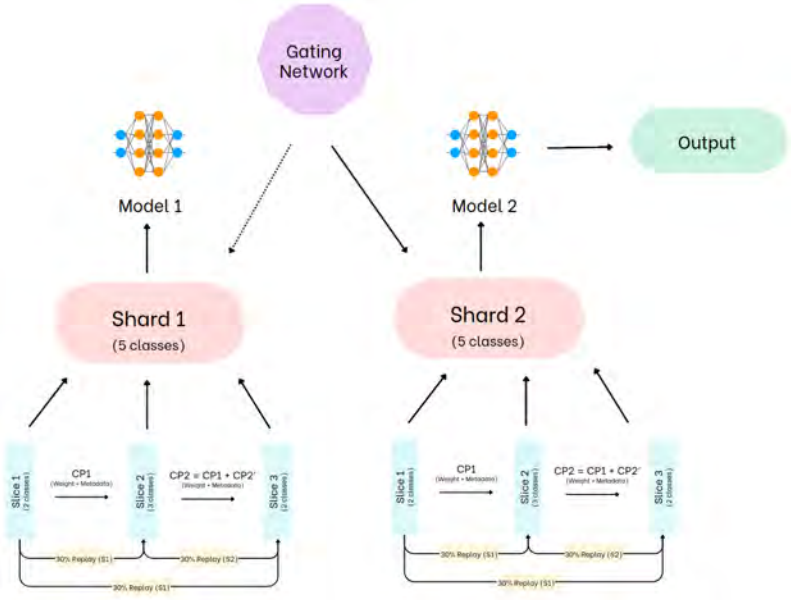

Figure 5.4: Full Model Architecture with Gating Network

Chapter 6

Unlearning Methodology and Evolution

This section presents the methodology adopted to implement unlearning using our modified SISA (Sharded, Isolated, Sliced, and Aggregated) architecture. The discussion begins with the baseline model and progressively details the development towards the final model, which demonstrated superior performance in terms of accuracy and retraining efficiency. Our research investigates the applicability of the SISA framework within the domain of machine unlearning. While the original SISA approach focused primarily on the removal of individual data points, our work extends this concept by enabling the removal of entire classes, an approach that aligns more closely with real-world scenarios. To enhance post-unlearning model performance, we integrated additional techniques such as replay ratios and gated networks.

The primary objective of our unlearning strategy is to minimize retraining time relative to the baseline model. To achieve this, the dataset is partitioned into multiple shards, with each shard responsible for training a constituent model. Upon receiving an unlearning request, the shard containing the target class is first identified. The specified class is then removed from the corresponding slice(s), followed by retraining the constituent model on the remaining data. This approach significantly reduces retraining time by limiting the amount of data involved in the process. Since the target class is completely eliminated from the dataset, this method ensures exact unlearning. The validity of this claim is supported by the confusion matrix analysis, as discussed in Chapter 7. The following subsection outlines the ablation study, which systematically examines each step of the unlearning methodology.

6.1 Baseline Model : Full Retraining Approach

Our baseline architecture consists of a standard Convolutional Neural Network (CNN) trained on the complete dataset without any sharding or structural modifications. Let the initial dataset be denoted as per Equation (5.1):
Let $\mathcal{M}_\theta$ denote the CNN model parameterized by θ. Initially, the model is trained on as per Equation (5.1), which can be represented as

$$\theta^* = \arg\min_\theta \mathcal{L}(\mathcal{M}_\theta, \mathcal{D}_u), \tag{6.1}$$

where $\mathcal{L}$ is the empirical loss function, typically cross-entropy for classification tasks. We denote the trained model as $\mathcal{M}(\mathcal{D}_u)$.

Suppose an unlearning request is received to forget a specific class $c \in \mathcal{Y}$. The baseline approach first scans through the entire dataset to identify and remove all samples belonging to class c:

$$\mathcal{D}_{u\setminus c} = \{(x_i, y_i) \in \mathcal{D}_u \mid y_i \neq c\}. \tag{6.2}$$

After this filtering, the model is fully retrained from scratch on the reduced dataset $\mathcal{D}_{u\setminus c}$:

$$\theta^*_{-c} = \arg\min_{\theta} \mathcal{L}(\mathcal{M}_\theta, \mathcal{D}_{u\setminus c}), \tag{6.3}$$

resulting in a new model $\mathcal{M}(\mathcal{D}_{u\setminus c})$ that has no exposure to class c.

In terms of training time, this baseline method requires

$$T^{\text{baseline}}_{\text{unlearn}} \approx T_{\text{train}}(\mathcal{D}_u), \tag{6.4}$$

since the model is retrained on almost the entire dataset, and the computational cost is proportional to the dataset size. Similarly, the accuracy of the baseline model before and after unlearning can be expressed as

$$\text{Acc}(\mathcal{M}(\mathcal{D}_u)) \geq \text{Acc}(\mathcal{M}(\mathcal{D}_{u\setminus c})), \tag{6.5}$$

where the inequality reflects the expected performance degradation due to both the reduced training set size and the removal of class c.

This baseline serves as a reference point for evaluating the efficiency and accuracy trade-offs of our proposed unlearning methods. Detailed empirical comparisons are presented in Chapter 7.

6.2 SISA Framework with Balanced Class Slicing

To reduce the retraining time associated with full model retraining, we implement the SISA (Sharded, Isolated, Sliced, and Aggregated) framework, which partitions the dataset into multiple shards and slices, and introduces intermediate checkpoints at the slice level to further minimize the computational cost during unlearning. The structural details of this implementation are described in Chapter 4.

As per Equation (5.1), each shard is further divided into L slices:

$$\mathcal{S}_k = \bigcup_{\ell=1}^{L} \mathcal{S}_{k,\ell}, \quad \mathcal{S}_{k,\ell_1} \cap \mathcal{S}_{k,\ell_2} = \emptyset \ \ \forall \ell_1 \neq \ell_2. \tag{6.6}$$

For each shard $\mathcal{S}_k$, a constituent model $\mathcal{M}_k$ is trained sequentially slice by slice:

$$\theta^*_{k,\ell} = \arg\min_{\theta} \mathcal{L}(\mathcal{M}_\theta, \mathcal{S}_{k,\ell}), \quad \ell = 1, \ldots, L, \tag{6.7}$$

with checkpoints saved after each slice to allow partial retraining in the event of unlearning.

Let $c \in \mathcal{Y}$ denote the class to be unlearned. Upon receiving an unlearning request, the system consults a metadata table $\mathcal{M}_{\text{meta}}$ that stores the class-to-shard mapping:

$$\mathcal{M}_{\text{meta}}(c) \mapsto k^*, \tag{6.8}$$

where k^* is the shard index containing samples of class c. Within shard $\mathcal{S}_{k^*}$, a linear scan is performed to locate and remove all samples associated with class c:

$$\mathcal{S}_{k^*,\ell}^{-c} = \{(x_i, y_i) \in \mathcal{S}_{k^*,\ell} \mid y_i \neq c\}, \quad \forall \ell \in \{1, \ldots, L\}. \tag{6.9}$$

In the original SISA framework, data points are distributed among slices in a balanced fashion to facilitate fine-grained unlearning of individual samples. Formally, if n_c denotes the number of samples of class c, then under balanced slicing each slice contains approximately $\frac{n_c}{L}$ samples of class c. While this is efficient for single-sample unlearning, it introduces significant overhead for class-level unlearning, since the removal of class c requires modifications to all slices within shard k^*:

$$\text{Slices to retrain for class } c = L. \tag{6.10}$$

This effectively negates some of the time-saving advantages of slicing, as each affected slice must be purged and retrained.

After the constituent model $\mathcal{M}_{k^*}$ is retrained on the modified slices

$$\mathcal{S}_{k^*}^{-c} = \bigcup_{\ell=1}^{L} \mathcal{S}_{k^*,\ell}^{-c}, \tag{6.11}$$

the overall model resumes inference. For evaluation, each input sample is forwarded to all K constituent models $\{\mathcal{M}_1, \ldots, \mathcal{M}_K\}$. The outputs are then aggregated through an aggregation layer, which selects the final prediction via the argmax function over the class probability distributions:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \left(\sum_{k=1}^{K} p_k(y \mid x) \right), \tag{6.12}$$

where $p_k(y \mid x)$ is the predicted probability of class y from constituent model $\mathcal{M}_k$.

6.3 SISA Framework with Sequential Class Slicing

While the balanced slicing strategy in the original SISA framework improved retraining efficiency compared to the baseline model, it was not fully optimized for class-level unlearning. The original design targets single data point removal and distributes samples of each class uniformly across all slices of a shard. This ensures that during sequential training, the constituent model repeatedly encounters similar patterns, enhancing its overall generalization capability. However, for class-level

deletion, this balanced distribution loses its practical advantage. From the model's perspective, since the unlearning process requires modifying every slice within a shard, slice-level granularity provides no additional benefit; effectively, only the shard-level isolation is being leveraged.

To address this inefficiency, we introduce a Sequential Class-Level Slicing (SCLS) strategy. In this approach, instead of distributing all classes uniformly across slices, we assign distinct classes to different slices within each shard:

$$\mathcal{S}_k = \bigcup_{\ell=1}^{L} \mathcal{S}_{k,\ell}, \quad \text{with} \quad \mathcal{Y}_{k,\ell} \cap \mathcal{Y}_{k,\ell'} = \emptyset \quad \forall\, \ell \neq \ell', \tag{6.13}$$

where $\mathcal{Y}_{k,\ell}$ denotes the set of class labels present in slice ℓ of shard k.
In the ideal case, this assignment achieves full slice-level class isolation, meaning each class resides in exactly one slice. However, this is not always guaranteed, as the number of slices L may be smaller than the number of classes, leading to a semi-isolated structure.

Under SCLS, when an unlearning request is issued for class c, the metadata table identifies both the shard k^* and the specific slice index ℓ^* containing class c:

$$\mathcal{M}_{\text{meta}}(c) \mapsto (k^*, \ell^*). \tag{6.14}$$

The targeted slice $\mathcal{S}_{k^*,\ell^*}$ is then deleted, and the corresponding constituent model $\mathcal{M}_{k^*}$ is partially retrained starting from the previous slice's checkpoint:

$$\theta^*_{k^*,\ell^*:L} = \arg\min_{\theta} \mathcal{L}\left(\mathcal{M}_\theta, \bigcup_{j=\ell^*+1}^{L} \mathcal{S}_{k^*,j}\right). \tag{6.15}$$

This significantly reduces retraining overhead compared to balanced slicing, since

$$\text{Slices to retrain for class } c = L - \ell^* + 1, \tag{6.16}$$

as opposed to retraining all L slices in the balanced case. The worst-case scenario occurs when the class resides in the first slice ($\ell^* = 1$), requiring retraining of all subsequent slices.

For inference, the overall architecture remains identical: all constituent models $\{\mathcal{M}_1, \ldots, \mathcal{M}_K\}$ process the input in parallel, and their outputs are aggregated through the argmax function over predicted class probabilities:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \left(\sum_{k=1}^{K} p_k(y \mid x)\right). \tag{6.17}$$

Despite its computational efficiency, Sequential Class-Level Slicing introduces a critical accuracy trade-off due to catastrophic forgetting. Because each slice now contains samples from distinct classes, the model does not revisit earlier class patterns during later slice training. By the time training reaches slice $\ell = n$, the model has effectively overwritten the representations learned from earlier slices $\ell = 1, \ldots, n-2$.

Empirically, we observe that each constituent model can reliably predict at most two classes which is the most recent slices, resulting in a significant drop in overall accuracy:

$$\text{Acc}_{\text{SCLS}} \ll \text{Acc}_{\text{Balanced Slicing}}, \tag{6.18}$$

a finding corroborated by the confusion matrix analysis presented in Chapter 7.

6.4 SISA Framework with Sequential Class Slicing aided with a Reinforced Replay Training Mechanism

To mitigate the catastrophic forgetting observed in the Sequential Class-Level Slicing (SCLS) approach, we propose a Reinforced Replay Training Mechanism (RRTM) integrated into the constituent model training phase. As previously discussed, the sequential isolation of classes across slices leads to a steep drop in accuracy, as the model gradually overwrites earlier representations during later slice training. To address this, we introduce a 30% replay mechanism that selectively reintroduces critical samples from earlier slices during training on the current slice, enabling the model to retain prior knowledge without significantly compromising computational efficiency.

Formally, consider the training of the k-th constituent model $\mathcal{M}_k$ over L sequential slices $\{\mathcal{S}_{k,1}, \ldots, \mathcal{S}_{k,L}\}$. At slice n, instead of training solely on $\mathcal{S}_{k,n}$, we augment the training set with a subset of replay data sampled from earlier slices $\mathcal{S}_{k,1}, \ldots, \mathcal{S}_{k,n-1}$. Let $\mathcal{R}_{k,n}$ denote this replay buffer. The effective training set for slice n becomes:

$$\widetilde{\mathcal{S}}_{k,n} = \mathcal{S}_{k,n} \cup \mathcal{R}_{k,n}. \tag{6.19}$$

The replay buffer $\mathcal{R}_{k,n}$ is constructed by selecting approximately $\rho = 0.3$ (i.e., 30%) of the most important samples from all previous slices:

$$|\mathcal{R}_{k,n}| \approx \rho \times \sum_{j=1}^{n-1} |\mathcal{S}_{k,j}|. \tag{6.20}$$

Importance is determined based on criteria such as class representation balance or sample-level loss contribution during previous training iterations. This selective replay acts as a reinforcement mechanism, helping the model maintain representations from earlier slices while learning new patterns in the current slice.

The training objective at slice n becomes:

$$\theta^*_{k,n} = \arg\min_{\theta} \mathcal{L}\big(\mathcal{M}_\theta, \widetilde{\mathcal{S}}_{k,n}\big), \tag{6.21}$$

which encourages the model to jointly optimize over current and replayed data distributions.

The unlearning mechanism remains identical to that of the SCLS version: upon receiving an unlearning request for class c, the relevant shard k^* and slice ℓ^* are identified via metadata lookup, the targeted slice is removed, and retraining proceeds from the previous checkpoint. The only difference is that each retraining step now incorporates replay samples, improving post-unlearning accuracy.

Empirical results show a substantial improvement in accuracy compared to SCLS without replay, confirming the effectiveness of this reinforcement mechanism. However, since each slice now trains on approximately $(1+\rho)$ times more data, both the training and retraining times increase proportionally:

$$T_{\text{train}}^{\text{RRTM}} \approx (1+\rho) \times T_{\text{train}}^{\text{SCLS}}, \tag{6.22}$$

reflecting a deliberate trade-off between computational efficiency and performance.

6.5 SISA Framework with Sequential Class Slicing aided with a Reinforced Replay Training Mechanism guided by a Gating Network

To further improve prediction accuracy and optimize inference efficiency, we incorporate a Gating Network into the SISA framework. This component acts as a learned router, directing each input to the most relevant constituent model based on its shard-level representation. The Gating Network is implemented as a lightweight CNN, referred to as $\mathcal{G}_\phi$, parameterized by ϕ.

6.5.1 Gating Network Training

For the Gating Network, we transform the class labels $y_i \in \mathcal{Y}$ into their corresponding shard identifiers $s_i \in \{1, \ldots, K\}$, where K is the total number of shards. This label transformation can be formalized using a mapping function:

$$g : \mathcal{Y} \to \{1, \ldots, K\}, \quad s_i = g(y_i). \tag{6.23}$$

The Gating Network is then trained as a standard multi-class classifier to predict the shard label s_i for each input:

$$\phi^* = \arg\min_{\phi} \mathcal{L}_{\text{gate}}(\mathcal{G}_\phi(x_i), s_i)\,, \tag{6.24}$$

where $\mathcal{L}_{\text{gate}}$ is typically a cross-entropy loss over shard labels. Since the Gating Network is trained only on shard IDs and not the actual class labels, it does not retain class-specific information and cannot be directly used for downstream tasks, ensuring data isolation and unlearning compliance.

6.5.2 Inference Procedure

In previous versions, the inference pipeline involved sending each input to all K constituent models and aggregating their outputs using an argmax function. In

this version, the Gating Network replaces the aggregation layer by predicting which shard is most likely to contain the correct class for a given input.
For a new input x, the Gating Network outputs a probability distribution over shards:

$$p_\phi(s \mid x) = \text{softmax}(\mathcal{G}_\phi(x)), \quad s \in \{1, \ldots, K\}. \tag{6.25}$$

We select the most probable shard based on a confidence threshold $\tau \in [0, 1]$:

$$s^* = \arg\max_s p_\phi(s \mid x), \quad \text{if } p_\phi(s^* \mid x) \geq \tau. \tag{6.26}$$

The input x is then forwarded to the selected constituent model $\mathcal{M}_{s^*}$, and the final prediction is produced by this single model:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} p_{s^*}(y \mid x), \tag{6.27}$$

where $p_{s^*}(y \mid x)$ is the class probability predicted by the chosen shard's constituent model. This routing mechanism increases the probability of selecting the correct constituent model for a given input, leading to improved accuracy. Empirically, we observe an approximate 10% accuracy boost compared to the previous (replay-based) version.

6.5.3 Unlearning and Computational Considerations

The unlearning mechanism remains unchanged from the previous version: class-level unlearning is handled at the shard and slice level, while the Gating Network is unaffected since it only stores shard-level mappings, not class-level knowledge.

From a computational perspective, although the introduction of the Gating Network adds a small training cost initially, it reduces inference overhead by querying only one constituent model per input, rather than all K. Let $\gamma_{\text{inference}}^{\text{prev}}$ be the cost of evaluating all constituent models and aggregating. The new cost becomes approximately:

$$\gamma_{\text{inference}}^{\text{gate}} \approx \gamma_{\text{Gating}} + \gamma_{\mathcal{M}_{s^*}}, \tag{6.28}$$

where γ_{Gating} is negligible relative to the full set of constituent models.

The number of trainable parameters in $\mathcal{G}_\phi$ is kept deliberately low—around 10–15% of the total parameters of all constituent models combined, making it a lightweight and tunable component. The capacity of the Gating Network can be treated as a hyperparameter, allowing practitioners to balance routing accuracy and computational cost depending on dataset characteristics.

Chapter 7

Result Analysis and Evaluation

This section presents a detailed evaluation of all five developed models across four different shard–slice configurations to assess their training performance, unlearning efficiency, and retraining behavior. The configurations tested were (2 shards $\times$ 3 slices), (2 shards $\times$ 5 slices), (3 shards $\times$ 3 slices), and (3 shards $\times$ 5 slices).
For each configuration, we measured four key metrics:

- Accuracy before unlearning
- Training time before unlearning
- Accuracy after unlearning
- Average retraining time after unlearning

An early stopping mechanism was applied during training with a patience of 7 steps, meaning training stopped automatically if the validation loss failed to improve for seven consecutive checks. This ensured that every model trained efficiently without overfitting. The recorded data, along with visual analyses, allows us to understand how accuracy and computational time varied with different architectures and setups, both before and after unlearning.

7.1 Accuracy and Time Performance Overview

Across all configurations, there was a clear and consistent trade-off between model accuracy and training/retraining efficiency. As the number of shards and slices increased, models trained faster but achieved lower overall accuracy. This is expected because dividing the dataset into smaller shards limits the amount of data each model sees, slightly restricting generalization. However, this same structure provides the crucial advantage of selective retraining, where only specific shards or slices need to be updated when unlearning a class.

After unlearning, all models exhibited some degree of accuracy drop. This drop was not a failure of learning but a direct confirmation of exact unlearning, since the test dataset still contained samples from the deleted class, the model could no longer predict them, automatically lowering overall accuracy. This validates that the model truly forgot the targeted class information.

7.2 Quantitative Results Summary

The following general trends were observed when averaging across all configurations:

- Model 1 (Baseline CNN) achieved the highest accuracy overall but required almost the same time to retrain as it did to train initially.
- Model 2 (SISA without slicing) showed slightly lower accuracy but significantly reduced retraining time since only one shard needed to be retrained.
- Model 3 (SISA with slice isolation) had the lowest accuracy due to catastrophic forgetting but achieved the fastest retraining because it could roll back using slice-level checkpoints.
- Model 4 (SISA with replay) restored most of the lost accuracy by introducing memory reinforcement while maintaining efficient retraining.
- Model 5 (SISA with replay and gating) provided the best balance, maintaining near-baseline accuracy with drastically lower retraining time.

7.3 Model-wise Evaluation and Interpretation

7.3.1 Model 1 – Baseline CNN

The baseline model served as a reference point. It achieved the highest training accuracy since it learned from the full dataset without any partitioning. However, this advantage came with a major limitation: whenever a class was removed, the entire model had to be retrained from scratch. Retraining required nearly the same time as the initial training, making it inefficient for scalable unlearning tasks.
Still, its strong performance established the upper bound of achievable accuracy against which the SISA-based methods could be compared. This model confirms that while monolithic training maximizes learning capacity, it lacks the flexibility required for data deletion and fast updates.

The model achieves a 81.67% validation accuracy before unlearning and 75.78% validation accuracy after unlearning.

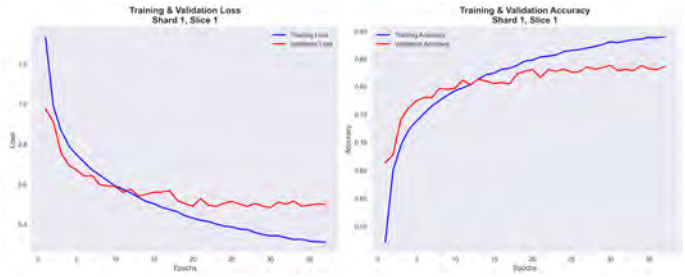

Figure 7.1: Training and Validation Curves of Baseline CNN Model

The model achieves exact unlearning as the “dog” class is deleted from scratch. This can be validated by the confusion matrix. In the below confusion matrix we

can see a diagonal trend which shows that the remaining classes are being predicted accurately while the forgotten "dog" has a zero prediction.

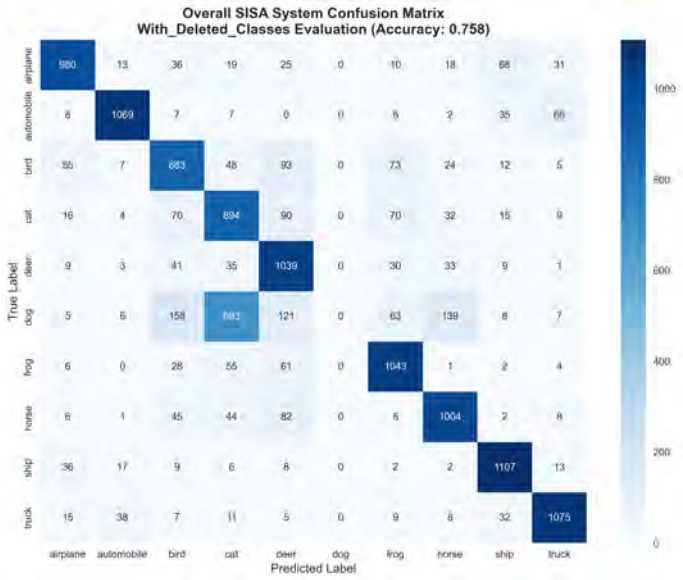

Figure 7.2: Confusion Matrix after Unlearning "Dog" Class for Baseline CNN Model

The below matrix shows randomly picked test samples of the "Dog" class. The model predicted each sample as some other class.

Figure 7.3: Verification of Exact Unlearning for "Dog" Class in Baseline CNN Model

7.3.2 Model 2 – SISA Framework without Slice Isolation

By dividing the dataset into multiple shards, this model achieved faster training and retraining compared to the baseline. Each shard-specific CNN was trained independently, and during unlearning, only the shard containing the deleted class needed

retraining. This reduced retraining time significantly.

However, because each shard was exposed to fewer classes, the model's overall accuracy dropped compared to the baseline. The reduced dataset diversity led to mild overfitting within individual shards. Despite this, the framework effectively demonstrated that partitioning can substantially improve retraining efficiency, even at a modest accuracy cost.

This version of our architecture achieved a 67.21% before unlearning accuracy in the 2 shard 3 slice setup and a 59.99% after unlearning accuracy in the same setup.

In order to demonstrate the training process, we will look into the before unlearning training vs validation curve of all the constituent models in a 3 shard 3 slice setup.

During training of each shard the accuracy and the validation accuracy will be significantly higher as it is being validated by only the assigned classes of that shard. However this accuracy is prone to drop as the whole model will be tested with all the class labels.

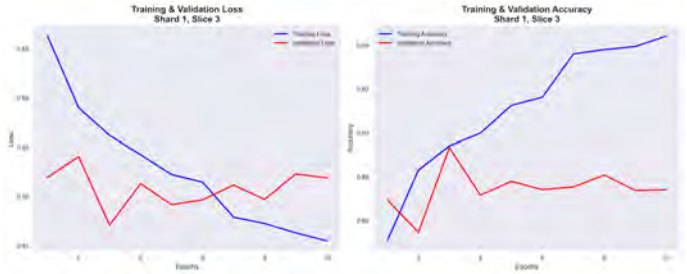

Figure 7.4: Training Curves of Shard 1, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup

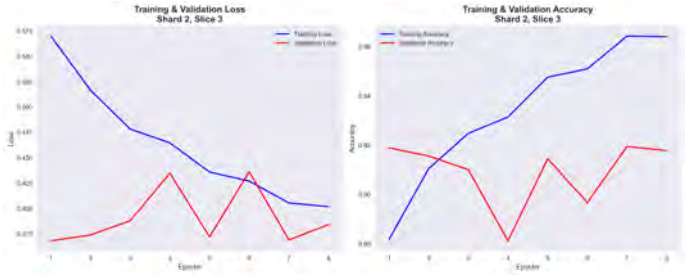

Figure 7.5: Training Curves of Shard 2, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup

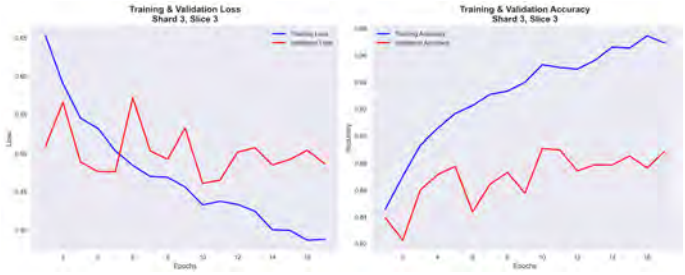

Figure 7.6: Training Curves of Shard 3, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup

The below curve shows the retraining of the shard from where our desired class was deleted.

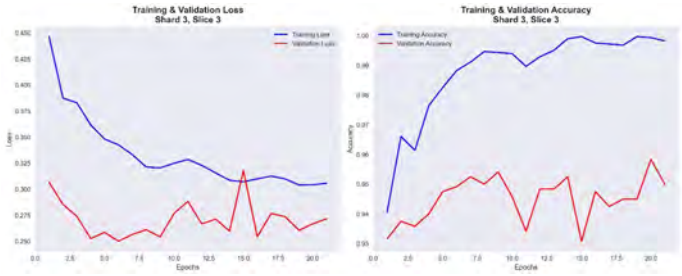

Figure 7.7: Retraining Curves after Unlearning "Dog" Class from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup

7.3.3 Model 3 – SISA Framework with Slice Isolation

This version introduced semi slice-level isolation inside each shard, creating much finer partitions. Each slice contained a very small number of classes, trained sequentially. Although this structure minimized retraining requirements, since only slices after the unlearned class needed to be retrained, it suffered from catastrophic forgetting.

The model quickly lost knowledge from earlier slices as it advanced through training, leading to very poor accuracy. Since each slice had limited data, the model tended to overfit early, triggering early stopping quickly. On the positive side, because every slice training step was saved as a checkpoint, retraining after unlearning became extremely fast. This model highlighted the importance of balancing isolation with memory retention.

This version of the architecture achieved a 18.55% before unlearning accuracy and a 15.60% after unlearning accuracy in the 2 shard 3 slice setup.

This version of our architecture had a very poor performance. Although we have achieved computational efficiency, it significantly lacks accuracy. Our next goal will be to keep this efficiency while upgrading the model accuracy.

7.3.4 Model 4 – SISA Framework with Reinforced Replay Training Mechanism (RRTM)

To address forgetting, a replay mechanism was added. At every slice training step, the model retrained on the current slice data plus a random portion (30% of previous slices) of samples from previously seen slices. This helped it remember older knowledge and maintain more stable learning.

The replay mechanism slightly increased training time due to extra data but significantly improved performance.

The Training Progress Graph illustrates this behavior. Compared to earlier frameworks, loss curves stabilize faster, and validation accuracy remains consistently high across all shards. Replay also worked seamlessly with checkpointing, if a class was deleted, the model could revert to its corresponding checkpoint and retrain only from that point onward. This combination provided an excellent balance between learning stability and retraining speed.

This version of our architecture gets a significant jump in model performance due to the replay mechanism. The before unlearning accuracy was 66.19% while the after unlearning accuracy was 61.39%. This is a before unlearning accuracy jump of 47.64% and after unlearning accuracy jump of 45.39%. This is a significant milestone in our research considering overall 30% more data input.

In order to demonstrate the training process, we will look into the before unlearning training vs validation curve of all the constituent models in a 3 shard 3 slice setup.

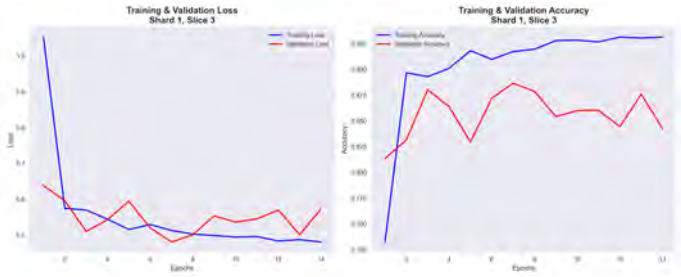

Figure 7.8: Training Curves of Shard 1, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup with Replay

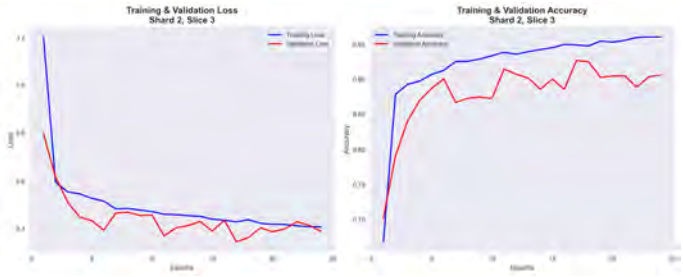

Figure 7.9: Training Curves of Shard 2, From Slice 1 to Slice 3 with Replay in 3 Shard 3 Slice Setup

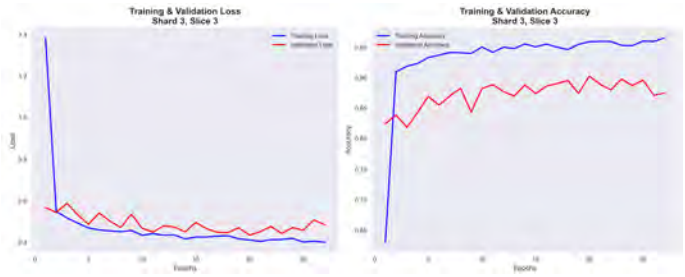

Figure 7.10: Training Curves of Shard 3, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup with Replay

The below curve shows the retraining of the shard from where the desired class was deleted.

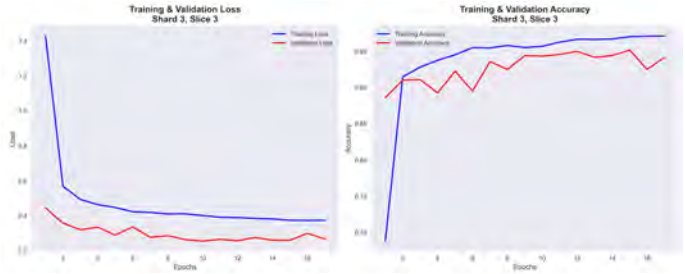

Figure 7.11: Retraining Curves after Unlearning "Dog" Class with Replay

7.3.5 Model 5 – SISA Framework with RRTM and Gating Network

The final model incorporated a gating network to determine which shard should process a given input. This addition improved overall accuracy while keeping retraining efficient. During inference, the gating model acted as a router—first classifying which shard was most suitable for an input, and then passing it to that shard's CNN for final prediction.

This adaptive routing allowed the system to combine the strengths of multiple shard models. The gating network itself did not require retraining during unlearning, since it learned shard mappings rather than class labels. As a result, retraining time stayed as low as in Model 4, while accuracy improved significantly.

This is the state of the art version of our modified architecture. This achieved the highest accuracy among all of the versions. The before unlearning accuracy was 73.12% and after unlearning accuracy was 70.12%. This is the closest to our baseline model which had the highest accuracy. Our final architecture is behind only 8.55% before unlearning accuracy and 5.66% after unlearning accuracy.

In order to demonstrate the training process, we will look into the before unlearning training vs validation curve of all the constituent models in a 3 shard 3 slice setup.

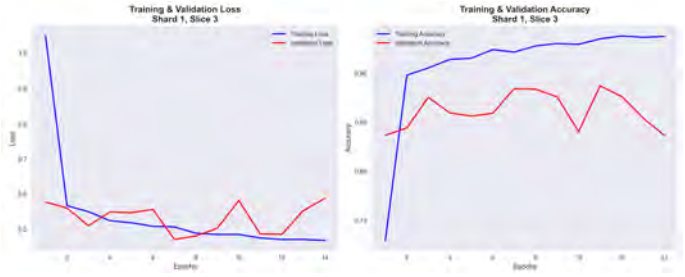

Figure 7.12: Training Curves of Shard 1, From Slice 1 to Slice 3 with Replay and Gating in from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup

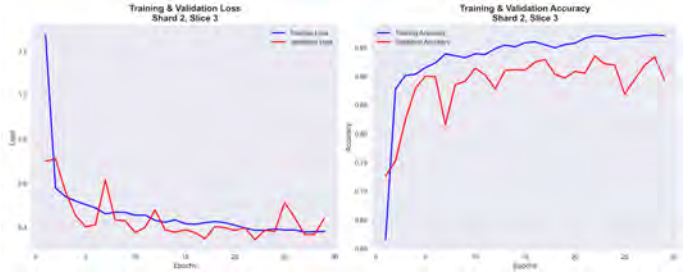

Figure 7.13: Training Curves of Shard 2, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup with Replay and Gating

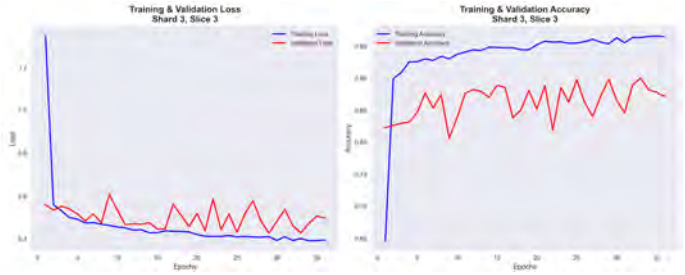

Figure 7.14: Training Curves of Shard 3, from Slice 1 to Slice 3 in 3 Shard 3 Slice Setup with Replay and Gating

The below graph shows the training and validation accuracy and loss of the constituent model trained by the shard from where the desired class was deleted.

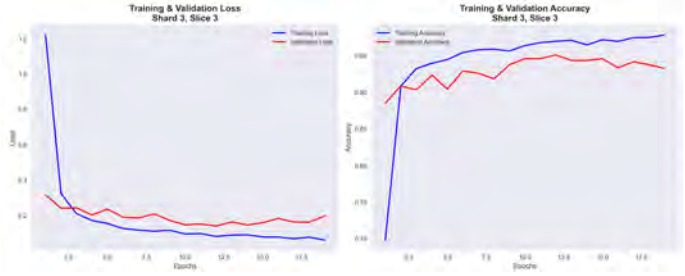

Figure 7.15: Retraining Curves after Unlearning “Dog” class with Replay Mechanism enabled and Gating Network applied

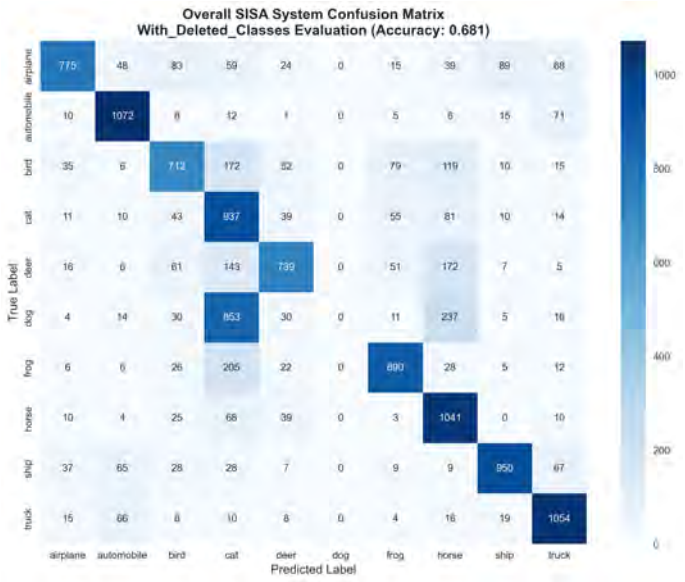

Figure 7.16: Overall SISA Confusion Matrix with Deleted "Dog" Class in Final Model

Figure 7.17: Verification of Exact Unlearning for "Dog" Class in Final Model

The Confusion Matrix (Figure 7.16) and Deleted-Class Verification Chart (Figure 7.17) clearly confirm exact unlearning. In the matrix, the deleted class ("Dog") has zero true predictions, meaning the model no longer associates any input with that class.

7.4 Configuration-Based Observations

When comparing across shard–slice setups, several patterns emerged:

- More shards reduced training and retraining time because each model received a smaller subset of data, but this also slightly lowered accuracy.
- More slices further decreased retraining time by limiting how much of a shard needed to be updated after unlearning, but at the cost of increased risk of forgetting.

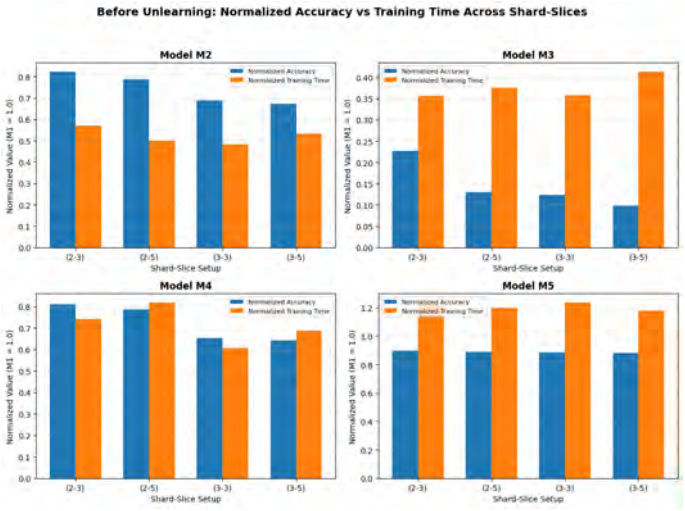

Figure 7.18: Average Normalized Accuracy and Training Time Comparison Before Unlearning Across Shard-Slice Setups

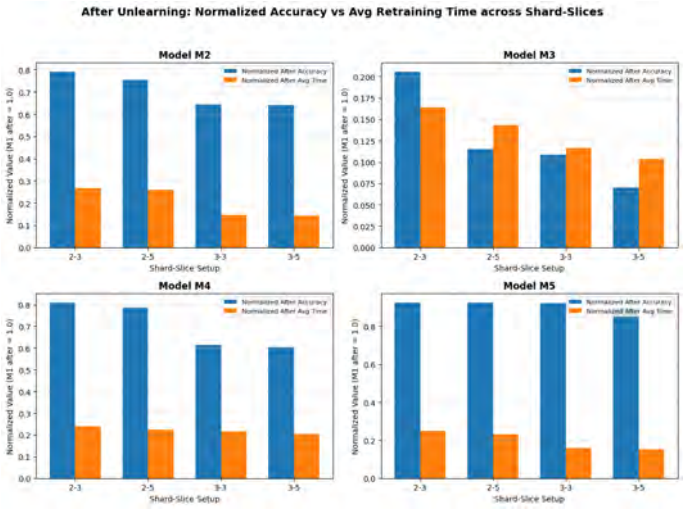

Figure 7.19: Average Normalized Accuracy and Retraining Time Comparison After Unlearning Across Shard-Slice Setups

- Replay largely countered this issue by helping the model remember earlier classes, making high-slice configurations more stable.
- Gating ensured that even if shard-level specialization caused imbalance, the final output remained accurate by routing inputs to the most relevant shard.

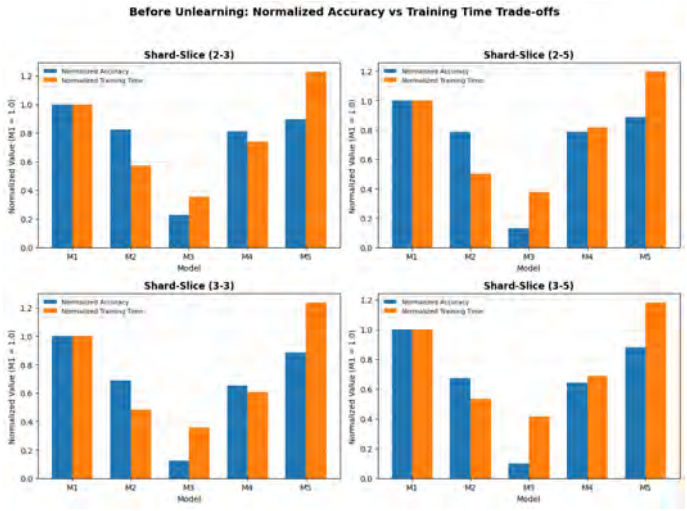

Figure 7.20: Accuracy and Training Time Comparison Before Unlearning Across Shard-Slice Setups

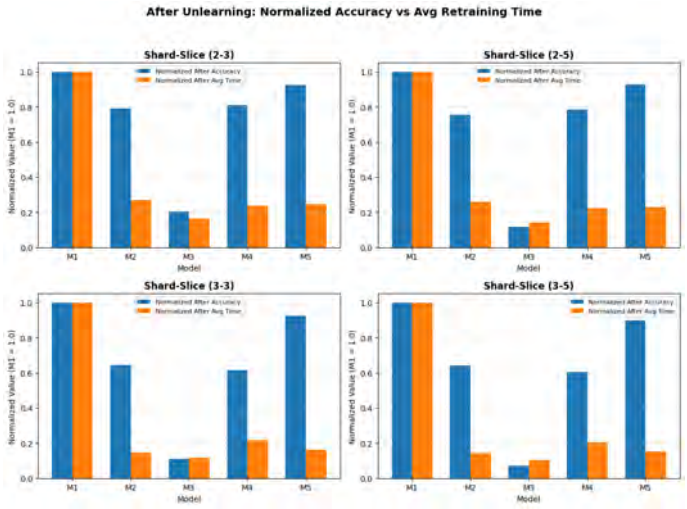

Figure 7.21: Normalized Accuracy and Average Retraining Time Comparison After Unlearning Across Shard-Slice Setups

Retraining time also depended on which slice contained the deleted class. Removing a class from a later slice required minimal retraining, while deleting one from an early slice forced all subsequent slices to retrain, increasing total retraining time. In our research, while comparing the retraining time we have taken the average retraining time which was calculated by studying the time to delete and retrain each of the classes.

7.5 Quantitative Highlights

- The Baseline model achieved the highest accuracy but required nearly full retraining time for every unlearning event.
- The SISA (no slicing) model showed reduced training and retraining time, but accuracy dropped due to limited data exposure per shard.

- The SISA (sliced) model achieved the fastest retraining due to checkpoints but failed to maintain stable accuracy.
- The SISA + Replay model recovered accuracy while preserving efficiency, stabilizing both training and validation performance.
- The SISA + Replay + Gating model demonstrated the best overall performance, combining near-baseline accuracy with minimal retraining effort.

The table below shows the accuracy and training time of before unlearning for all of our architecture versions for all the shard-slice setups. This table also shows the accuracy and average retraining time after unlearning for all the architecture versions for all the shard-slice setups. In this table the Baseline Model is being represented as Model 1 and similarly SISA Balanced Class Distribution as Model 2, SISA Sequential Class Distribution as Model 3, SISA with Replay Mechanism as Model 4 and SISA with Replay and Gating as Model 5. Here the Model 1 is not dependent on shard-slice setup. However it is placed in the table in shard-slice sections for a reference point for comparison.

		Before Unlearning		After Unlearning	
Shard-Slice	Model	Accuracy%	T. Time(s)	A. Accuracy%	A. R.T. Time(s)
(2-3)	1*	81.67	77.79	75.78	69.86
	2	67.21	44.34	59.99	18.71
	3	18.55	27.7	15.6	11.45
	4	66.19	57.62	61.39	16.66
	5	**73.12**	59.16+36.37	**70.12**	17.33
(2-5)	1*	81.67	77.79	75.78	69.86
	2	64.31	39.03	57.22	18.1
	3	10.67	29.21	8.7	10.01
	4	64.17	63.74	59.47	15.53
	5	**72.55**	55.7+37.41	**70.17**	16.1
(3-3)	1*	81.67	77.79	75.78	69.86
	2	56.23	37.53	48.73	10.23
	3	10.07	27.8	8.21	8.15
	4	53.33	47.3	46.47	15.1
	5	**72.42**	55.11+41.07	**69.98**	11.2
(3-5)	1*	81.67	77.79	75.78	69.86
	2	54.88	41.47	48.51	10.01
	3	7.95	32.16	5.3	7.25
	4	52.4	53.4	45.8	14.3
	5	**71.83**	51.5+40.3	**68.01**	10.59

Table 7.1: Performance comparison across different shard-slice configurations

7.6 Comparison of Different Versions of the Architecture

The bar chart(s) below shows the comparison of accuracy and training time before unlearing and accuracy and average retraining time after unlearning.

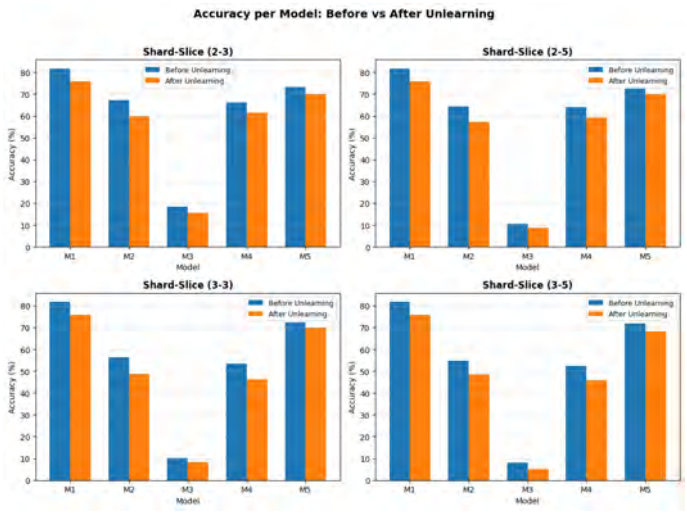

Figure 7.22: Accuracy per Architecture Before and After Unlearning

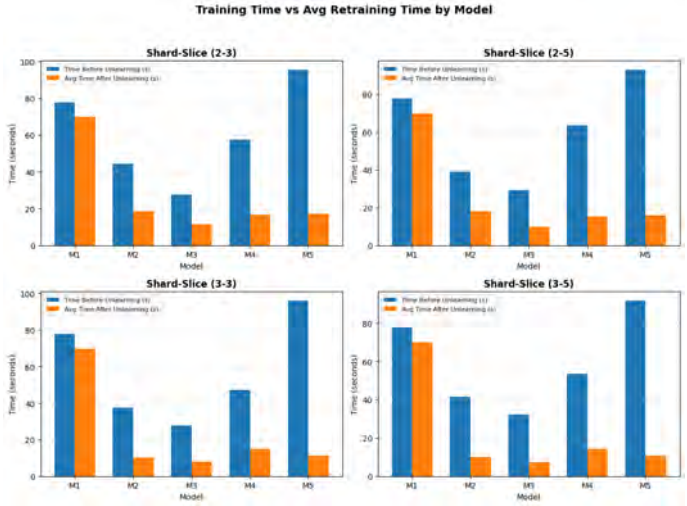

Figure 7.23: Training and Average Retraining Time per Architecture Before and After Unlearning

We can see that, as we progress towards our state of the art model the accuracy is increasing gradually and the average retraining time is also decreasing compared to the original baseline model.

7.7 Summary of Findings

From the complete set of experiments, we can conclude that the evolution of the SISA framework—from simple sharding to slice isolation, replay, and finally gating—led to steady improvements in both learning stability and retraining efficiency.

In order to reduce the computational overhead of the classical SISA framework we introduced sequential class distribution. However due to lack of variation in learning data the model could not capture any significant knowledge. It only retained the knowledge from the last two slices. This further validates the traditional claim that

variation in training data is essential for a model to capture knowledge.

In order to achieve computational efficiency and incorporate variation, we introduced a reinforced replay mechanism. With the help of this, the slice distribution is kept sequential. While training and retraining we are passing 30% most important data from the past slices in order to achieve variance in training data. This significantly boosted our model performance while keeping the retraining time very low.

While the baseline CNN remains unmatched in raw accuracy, the SISA with Replay and Gating framework achieves near-equivalent performance with a fraction of the retraining cost, and it demonstrates true exact unlearning, as validated by the confusion matrices and verification graphs.

The final model thus represents an effective balance between computational efficiency, data privacy compliance, and accuracy, making it a robust and scalable approach for real-world unlearning systems.

Chapter 8

Conclusion

This research investigated the implementation of machine unlearning using the SISA (Sharded, Isolated, Sliced, and Aggregated) framework for class-level deletion in CNN models, conducted on the CIFAR-10 dataset with the goal of reducing retraining time while maintaining model accuracy after removing entire classes. We began with a baseline CNN model trained on the complete dataset, where unlearning a class required full retraining on the reduced dataset, which proved computationally expensive and motivated the exploration of more efficient alternatives. The original SISA framework partitioned the dataset into multiple shards, with each shard training an independent constituent model, but the balanced class distribution across slices meant that class-level unlearning still required modifying all slices within the affected shard, limiting efficiency gains. To address this, we introduced Sequential Class-Level Slicing (SCLS), where distinct classes were assigned to different slices within each shard, enabling targeted slice removal and reducing the number of slices requiring retraining. However, this approach suffered from catastrophic forgetting, as the model lost knowledge of earlier classes during sequential training. To mitigate this issue, we incorporated a Reinforced Replay Training Mechanism (RRTM) that introduced 30% of earlier samples during each slice's training phase, substantially improving accuracy by allowing the model to retain prior knowledge while learning new classes. Finally, we integrated a lightweight Gating Network that routes each input to the most appropriate constituent model based on learned shard representations, improving prediction accuracy by approximately 10% compared to the aggregation-based approach while also reducing inference cost by activating only one constituent model per input rather than all models. The experimental results demonstrated that our final framework achieved competitive accuracy of 73.1% with replay, further improved with the gating network, while significantly reducing retraining time compared to the baseline, with confusion matrix analysis confirming exact unlearning by showing zero predictions for the removed class after the unlearning process.

8.1 Future Work

While this research successfully demonstrated the effectiveness of the modified SISA framework for class-level unlearning, several directions remain open for future exploration. The current framework handles single-class removal requests, but extending it to support multi-label class unlearning. Our experiments were conducted ex-

clusively on CIFAR-10, which contains only 10 classes with balanced distribution and relatively simple image structure, so testing the framework on more diverse datasets such as CIFAR-100, ImageNet, or domain-specific datasets like medical images would provide stronger validation and help identify potential limitations across different data characteristics. Additionally, several key parameters in our framework, including the number of shards, slices per shard, replay ratio, gating network capacity, and confidence threshold for shard selection, were chosen based on limited empirical testing, could reveal configurations that further improve the accuracy-efficiency trade-off. Our future work should also include formal privacy guarantees and verification mechanisms to ensure that unlearned data truly has no influence on the model's behavior.

Bibliography

[1] M. Bowling, J. Fürnkranz, T. Graepel, and R. Musick, "Machine learning and games," *Machine Learning*, vol. 63, pp. 211–215, 2006. DOI: 10.1007/s10994-006-8919-x (cit. on p. 1).

[2] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 463–480. DOI: 10.1109/SP.2015.35 (cit. on pp. 2, 4, 9, 11).

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015 (cit. on p. 1).

[4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, *Membership inference attacks against machine learning models*, 2017. arXiv: 1610.05820 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1610.05820 (cit. on p. 4).

[5] T. M. S. Tax, P. Mediano, and M. Shanahan, "The partial information decomposition of generative neural network models," *Entropy*, vol. 19, p. 474, 2017. DOI: 10.3390/e19090474 (cit. on p. 2).

[6] A. Ginart, M. Y. Guan, G. Valiant, and J. Zou, *Making ai forget you: Data deletion in machine learning*, 2019. arXiv: 1907.05012 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1907.05012 (cit. on pp. 5, 11).

[7] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in nlp*, 2019. arXiv: 1906.02243 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1906.02243 (cit. on p. 5).

[8] X. Chen, T. Yang, W.-L. Yang, and W.-M. Liu, *Turbulent cascade induced persistent current of cold atomic superfluids*, 2020. arXiv: 2007.02274 [cond-mat.quant-gas]. [Online]. Available: https://arxiv.org/abs/2007.02274 (cit. on p. 5).

[9] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 9301–9309. DOI: 10.1109/CVPR42600.2020.00932 (cit. on pp. 6, 7, 11–13).

[10] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 3832–3842. [Online]. Available: https://proceedings.mlr.press/v119/guo20c.html (cit. on pp. 10, 11).

[11] T. Henighan, J. Kaplan, M. Katz, *et al.*, *Scaling laws for autoregressive generative modeling*, 2020. arXiv: 2010.14701 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2010.14701 (cit. on p. 5).

[12] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 6999–7019, 2020. DOI: 10.1109/TNNLS.2021.3084827 (cit. on pp. 1, 2).

[13] S. Neel, A. Roth, and S. Sharifi-Malvajerdi, *Descent-to-delete: Gradient-based methods for machine unlearning*, 2020. arXiv: 2007.02923 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2007.02923 (cit. on pp. 6, 12, 13).

[14] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, *et al.*, "Machine-unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 141–159. DOI: 10.1109/SP40001.2021.00019 (cit. on pp. 4–6, 8, 11, 12, 20, 37).

[15] M. Umer and R. Polikar, *Adversarial targeted forgetting in regularization and generative based continual learning models*, 2021. arXiv: 2102.08355 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2102.08355 (cit. on p. 16).

[16] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, *Training compute-optimal large language models*, 2022. arXiv: 2203.15556 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2203.15556 (cit. on p. 5).

[17] B. Liu, Q. Liu, and P. Stone, *Continual learning and private unlearning*, 2022. arXiv: 2203.12817 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2203.12817 (cit. on pp. 10, 20).

[18] N. G. Marchant, B. I. P. Rubinstein, and S. Alfeld, *Hard to forget: Poisoning attacks on certified machine unlearning*, 2022. arXiv: 2109.08266 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2109.08266 (cit. on p. 7).

[19] C. F. G. Santos and J. Papa, "Avoiding overfitting: A survey on regularization methods for convolutional neural networks," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1–25, 2022. DOI: 10.1145/3510413 (cit. on p. 1).

[20] C. Wu, S. Zhu, and P. Mitra, *Federated unlearning with knowledge distillation*, 2022. arXiv: 2201.09441 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2201.09441 (cit. on p. 19).

[21] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli, "Zero-shot machine unlearning," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2345–2354, 2023, ISSN: 1556-6021. DOI: 10.1109/tifs.2023.3265506. [Online]. Available: http://dx.doi.org/10.1109/TIFS.2023.3265506 (cit. on pp. 12–14).

[22] Y. Dukler, B. Bowman, A. Achille, A. Golatkar, A. Swaminathan, and S. Soatto, *Safe: Machine unlearning with shard graphs*, 2023. arXiv: 2304.13169 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2304.13169 (cit. on p. 15).

[23] Y. Huang and C. L. Canonne, *Tight bounds for machine unlearning via differential privacy*, 2023. arXiv: 2309.00886 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.00886 (cit. on p. 17).

[24] N. Si, H. Zhang, H. Chang, W. Zhang, D. Qu, and W. Zhang, *Knowledge unlearning for llms: Tasks, methods, and challenges*, 2023. arXiv: 2311.15766 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2311.15766 (cit. on p. 21).

[25] L. Wang, T. Chen, W. Yuan, X. Zeng, K.-F. Wong, and H. Yin, "KGA: A general machine unlearning framework based on knowledge gap alignment," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 13 264–13 276. DOI: 10.18653/v1/2023.acl-long.740. [Online]. Available: https://aclanthology.org/2023.acl-long.740/ (cit. on p. 9).

[26] Y. Yoon, J. Nam, H. Yun, J. Lee, D. Kim, and J. Ok, *Few-shot unlearning by model inversion*, 2023. arXiv: 2205.15567 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2205.15567 (cit. on p. 13).

[27] V. Boža, *Fast and effective weight update for pruned large language models*, 2024. arXiv: 2401.02938 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2401.02938 (cit. on p. 16).

[28] C. Fan, J. Liu, Y. Zhang, E. Wong, D. Wei, and S. Liu, *Salun: Empowering machine unlearning via gradient-based weight saliency in both image classification and generation*, 2024. arXiv: 2310.12508 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2310.12508 (cit. on p. 22).

[29] X. Gao, X. Ma, J. Wang, *et al.*, "Verifi: Towards verifiable federated unlearning," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 6, pp. 5720–5736, 2024. DOI: 10.1109/TDSC.2024.3382321 (cit. on p. 19).

[30] Y. Hong, L. Yu, H. Yang, S. Ravfogel, and M. Geva, *Intrinsic evaluation of unlearning using parametric knowledge traces*, 2024. arXiv: 2406.11614 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2406.11614 (cit. on p. 15).

[31] Z. Liu, Y. Jiang, J. Shen, *et al.*, *A survey on federated unlearning: Challenges, methods, and future directions*, 2024. arXiv: 2310.20448 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2310.20448 (cit. on p. 5).

[32] Z. Liu, H. Ye, Y. Jiang, *et al.*, *Privacy-preserving federated unlearning with certified client removal*, 2024. arXiv: 2404.09724 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2404.09724 (cit. on p. 13).

[33] T. T. Nguyen, T. T. Huynh, Z. Ren, *et al.*, *A survey of machine unlearning*, 2024. arXiv: 2209.02299 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.02299 (cit. on pp. 12, 13).

[34] Y. Sinha, M. Mandal, and M. Kankanhalli, *Distill to delete: Unlearning in graph networks with knowledge distillation*, 2024. arXiv: 2309.16173 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.16173 (cit. on p. 14).

[35] A. K. Tarun, V. S. Chundawat, M. Mandal, and M. Kankanhalli, "Fast yet effective machine unlearning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 9, pp. 13 046–13 055, Sep. 2024, ISSN: 2162-2388. DOI: 10.1109/tnnls.2023.3266233. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2023.3266233 (cit. on p. 8).

[36] D. Trippa, C. Campagnano, M. S. Bucarelli, G. Tolomei, and F. Silvestri, $\nabla\tau$*: Gradient-based and task-agnostic machine unlearning*, 2024. arXiv: 2403.14339 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.14339 (cit. on pp. 10, 14).

[37] S. Wu, X. Zhao, and X. Huang, *Data augmentation for continual rl via adversarial gradient episodic memory*, 2024. arXiv: 2408.13452 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2408.13452 (cit. on p. 17).

[38] S. Xing, F. Zhao, Z. Wu, *et al.*, *Efuf: Efficient fine-grained unlearning framework for mitigating hallucinations in multimodal large language models*, 2024. arXiv: 2402.09801 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.09801 (cit. on p. 21).

[39] J. Xu, Z. Wu, C. Wang, and X. Jia, "Machine unlearning: Solutions and challenges," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 3, pp. 2150–2168, 2024. DOI: 10.1109/TETCI.2024.3379240 (cit. on p. 7).

[40] Y. Yang, X. Han, Y. Chai, R. Ebrahimi, R. Behnia, and B. Padmanabhan, *From machine learning to machine unlearning: Complying with gdpr's right to be forgotten while maintaining business value of predictive models*, 2024. arXiv: 2411.17126 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2411.17126 (cit. on p. 17).

[41] J. Yao, E. Chien, M. Du, *et al.*, "Machine unlearning of pre-trained large language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 8403–8419. DOI: 10.18653/v1/2024.acl-long.457. [Online]. Available: https://aclanthology.org/2024.acl-long.457/ (cit. on pp. 20, 21).