NeuSemSlice: Towards Effective DNN Model Maintenance via Neuron-level Semantic Slicing

SHIDE ZHOU, Huazhong University of Science and Technology, China
TIANLIN LI*, Nanyang Technological University, Singapore
YIHAO HUANG, Nanyang Technological University, Singapore
LING SHI, Nanyang Technological University, Singapore
KAILONG WANG*, Huazhong University of Science and Technology, China
YANG LIU, Nanyang Technological University, Singapore
HAOYU WANG, Huazhong University of Science and Technology, China

Deep Neural networks (DNNs), extensively applied across diverse disciplines, are characterized by their integrated and monolithic architectures, setting them apart from conventional software systems. This architectural difference introduces particular challenges to maintenance tasks, such as model restructure (*e.g.*, model compression), re-adaptation (*e.g.*, fitting new samples), and incremental development (*e.g.*, continual knowledge accumulation).

Prior research addresses these challenges by identifying task-critical neuron layers, and dividing neural networks into semantically-similar sequential modules. However, such layer-level approaches fail to precisely identify and manipulate neuron-level semantic components, restricting their applicability to finer-grained model maintenance tasks.

In this work, we implement NeuSemSlice, a novel framework that introduces the *semantic slicing* technique to effectively identify critical neuron-level semantic components in DNN models for *semantic-aware model maintenance* tasks. Specifically, *semantic slicing* identifies, categorizes and merges critical neurons across different categories and layers according to their semantic similarity, enabling their flexibility and effectiveness in the subsequent tasks.

For *semantic-aware model maintenance* tasks, we provide a series of novel strategies based on *semantic slicing* to enhance NeuSemSlice. They include semantic components (*i.e.*, critical neurons) preservation for model restructure, critical neuron tuning for model re-adaptation, and non-critical neuron training for model incremental development. A thorough evaluation has demonstrated that NeuSemSlice significantly outperforms baselines in all three tasks.

CCS Concepts: • **Software and its engineering** → **Maintaining software**; • **Computing methodologies** → *Artificial intelligence*.

Additional Key Words and Phrases: Deep Neural Networks, Model Maintenance

*Corresponding author.

Authors' Contact Information: Shide Zhou, Huazhong University of Science and Technology, Wuhan, China, shidez@hust.edu.cn; Tianlin Li, Nanyang Technological University, Singapore, Singapore, tianlin001@e.ntu.edu.sg; Yihao Huang, Nanyang Technological University, Singapore, Singapore, huang.yihao@ntu.edu.sg; Ling Shi, Nanyang Technological University, Singapore, Singapore, ling.shi@ntu.edu.sg; kailong Wang, Huazhong University of Science and Technology, Wuhan, China, wangkl@hust.edu.cn; Yang Liu, Nanyang Technological University, Singapore, Singapore, yangliu@ntu.edu.sg; Haoyu Wang, Huazhong University of Science and Technology, Wuhan, China, haoyuwang@hust.edu.cn.

ACM 1557-7392/2025/4-ART
https://doi.org/10.1145/3731556

1 INTRODUCTION

In the evolving landscape of artificial intelligence, DNNs represent a dynamic shift towards data-driven programming paradigms, extensively adopted across various fields [4, 22, 57]. As these models increasingly serve as core components in modern software, their maintenance transcends mere optimization challenges and requires strategies grounded in established software engineering principles [27, 51, 61]. Analogous to how traditional software evolves to meet shifting requirements and enhance functionality, effective maintenance of DNNs is crucial for combating model drift and ensuring sustained accuracy and reliability as they are continuously exposed to new and changing data.

Unlike traditional software systems characterized by their modular and discrete structures, DNNs are typically designed as cohesive, monolithic entities. The unique architecture introduces distinct and challenging maintenance tasks, including **model restructure**, **re-adaptation**, and **incremental development**. They are pivotal elements for the operational robustness and longevity of neural network-based systems in a rapidly advancing technological milieu [2, 3, 7]. By framing DNN maintenance as an extension of traditional software maintenance practices, we address a critical need within the software engineering community, ensuring that DNNs continue to function correctly, evolve smoothly, and remain aligned with evolving requirements.

Despite various proposals in the literature for maintaining DNN models like the DeepArc framework [51], these methods continue to face several limitations. One major challenge with traditional strategies, which depend largely on periodic model retraining, is their ineffectiveness in combating the unavoidable problem of model drift. DeepArc addresses this problem by modularizing neural networks, organizing layers into modules based on their semantic similarities. This strategy streamlines maintenance tasks, facilitating the pruning and fine-tuning of specific modules or layers for more efficient updates. However, this approach is constrained by its focus on layer-level manipulation. A more refined strategy that decomposes semantic components at a finer level, such as individual neurons, could enhance model restructuring and adaptation. Specifically, selectively pruning and tuning a small number of neurons might offer better effectiveness. Furthermore, the incremental development of models [59] requires preserving the semantics and functions of previous versions while introducing new features. Given that semantics and functions in DNNs are primarily structured around neurons [10, 32–34, 37, 53, 63, 64], the method of identifying and updating whole layers falls short of preserving the original semantics or functions during the incremental development process.

Acknowledging the existing limitations, our goal is to develop techniques that enhance DNN model maintenance. Drawing inspiration from the significant benefits of dynamic slicing in the maintenance and evolution of traditional software [5, 19], we adopt similar slicing techniques for DNNs [35, 39, 56, 63, 65]. Specifically, dynamic slicing excels in identifying and isolating essential code segments for implementing changes without causing unintended effects. By analogy, DNN slicing provides a systematic way to identify and separate semantic components at the neuron level, mirroring how code slicing pinpoints essential segments that influence a program's behavior. Focusing interventions on the neurons most critical to the model's functionality ensures targeted changes without introducing unintended side effects in non-critical parts.

Our work. We introduce the NeuSemSlice framework, building on the concept of semantic decomposition through DNN slicing, to innovate model maintenance practices. This framework features two main components: *semantic slicing* and *semantic-aware model maintenance*. The former facilitates the precise identification of semantic elements within DNNs, such as critical neurons, at a granular level. Following this, the latter delivers tailored strategies for model maintenance tasks that demand an in-depth semantic comprehension, enabling effective and targeted strategies.

The *semantic slicing* module is designed to accurately identify critical neurons across the entire search space by employing a category-specific[1] strategy for optimal performance. Specifically, we compute *neuron contribution scores* [17, 35, 42, 50, 63] for each neuron category individually, selecting a specific range of critical neurons for each category before combining them across the entire dataset. Given the potential variability in the optimal range of critical neurons across different categories and layers—which may hinder effective combination—we refine this process to focus on neurons that encapsulate the layer's full semantic range. This is achieved by assessing the similarity between the characteristics of these chosen neurons and the entire layer. The subsequent merging strategy further enhances the utility and relevance of the identified critical neurons.

Mirroring and inspired by the traditional software engineering tasks, the *semantic-aware model maintenance* module utilizes semantic slicing, and provides the following maintenance tasks for DNN models:

❶ **Model Restructure**: In traditional software maintenance, code slicing can be used to identify redundant or non-essential code, streamlining the program and improving efficiency [23]. Similarly, in DNN model maintenance, model restructure focuses on retaining only the critical neurons necessary for maintaining predictive accuracy, while minimizing storage requirements. This step isolates non-essential components of the model, simplifying the overall structure by focusing on the critical parts.

❷ **Model Re-adaptation**: Code slicing in software engineering helps identify and modify the core parts of a program when fixing bugs [31, 47, 55]. In DNNs, model re-adaptation focuses on retraining critical neurons to address prediction errors, reducing the need to adjust many parameters. This mirrors how targeted changes in code slicing resolve issues without overhauling the entire program.

❸ **Model Incremental Development**: In traditional software maintenance, slicing can be used to store the historical semantic information of code, thereby assisting software evolution [36, 52]. Similarly, model incremental development stabilizes critical neuron parameters to prevent catastrophic forgetting when integrating new knowledge. This ensures that core logic remains intact while new functionality is added seamlessly, akin to how code slicing maintains program stability during incremental updates.

We conduct extensive experiments to evaluate the performance of NeuSemSlice. For model restructure, NeuSemSlice enhances the efficiency of existing model compression techniques by an average of 13.46%, while also improving compression rate. For model re-adaptation, NeuSemSlice only trains an average of 61.49% of the parameters, enhancing both accuracy and efficiency. For incremental model development, NeuSemSlice performs well, boosting average accuracy by approximately 40% compared to direct retraining.

Contributions. The contributions of this paper is summarized as follows: ❶ We propose NeuSemSlice, including *semantic slicing* for finer-grained neuron-level decomposition, and *semantic-aware model maintenance*, offering novel and effective strategies for semantic-aware DNN model maintenance tasks. ❷ We detail a sophisticated *semantic slicing* technique that employs a category-specific strategy to accurately identify and combine critical neurons across the dataset, refining the process of neuron selection based on semantic range to optimize model performance. ❸ We tackle the challenging maintenance tasks via the *semantic-aware model maintenance* module, which significantly advances the tasks of *model restructure*, *re-adaptation*, and *incremental development*. ❹ We conduct extensive experiments on three model maintenance tasks, showcasing the superior performance of NeuSemSlice.

[1]To be precise, we identify critical neurons within the network model that have a significant impact on the output of each output neuron, where different categories refer to different output neurons. For example, in classification tasks, categories can be determined based on the labels in the dataset; in regression tasks, it can be considered a single-category task; in other types of tasks, different output neurons are treated as different categories.

2 BACKGROUND AND RELATED WORK

2.1 Deep Neural Networks

A DNN typically consists of multiple layers of neurons [58], which can be formally defined as follows.

Definition 1. *A Deep Neural Network (DNN) f consists of L multiple layers $\langle l_0, l_1, \ldots, l_{L-1} \rangle$, where l_0 is the input layer, l_{L-1} is the output layer, and $l_1, \ldots, l_{L-2}$ are hidden layers. The inputs of each layer are the outputs of the previous layer.*

In this work, we mainly focus on the classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is a set of inputs and $\mathcal{Y}$ is a set of classes. Given an input $x \in \mathcal{X}$, we use $f^l(x)$ to represent the internal features extracted by the layer l and $f_n(x)$ to represent the internal features extracted by the neuron n (*i.e.*, the output values of neurons at l). $\hat{y}$ represents the output of DNN (*i.e.*, $f(x)$) and y is the label of the given input x.

There usually exists redundancy in DNNs which refers to components within the network that do not significantly contribute to its performance [8, 63]. This redundancy can manifest in various forms, such as redundant neurons and redundant layers. Nevertheless, determining which components are redundant and can be removed without impacting performance requires sophisticated analysis and careful validation.

2.2 DNN Model Maintenance

In the realm of software engineering, maintaining and evolving systems has long been recognized as a critical challenge, and DNNs are no exception. As these models increasingly integrate into complex, safety-critical, and rapidly changing environments, practitioners must apply well-established maintenance principles—such as iterative improvement, modular refactoring, and continuous quality assurance—to keep DNNs robust, adaptable, and high-performing. This concept, akin to the established practices of traditional software maintenance, addresses the evolving needs within the software engineering domain for more efficient and adaptable DNN systems.

DeepArc [51] has introduced a new neural network architecture strategy to cut model maintenance costs. This approach identifies semantically similar layers as redundant, allowing for model restructuring by pruning these layers. Only non-redundant layers are fine-tuned on the target data, based on calculated semantic similarities of layer representations: Given a dataset D, the matrix M^{l_i} is denoted as the similarity matrix for the i-th network layer l_i, where

$$M^{l_i}[a, b] = f^{l_i}(x_a) \cdot f^{l_i}(x_b), \tag{1}$$

with x_a and x_b representing any two inputs from the dataset D. Intuitively, M^{l_i} indicates the embedding landscape of the dataset on the i-th layer and the shape of the matrix is solely dependent on the size of the dataset. Similar embedding landscapes (*i.e.*, similar matrices) in two layers indicate they extract comparable semantic information from inputs. With the matrix M^l for each layer l, DeepArc then follows the centered kernel alignment [24] (CKA) metric to calculate the similarity between M^{l_i} and M^{l_j} to represent the semantic similarity between layer f^{l_i} and f^{l_j}.

$$\text{sim}_{\text{sem}}(f^{l_i}, f^{l_j}) = \text{CKA}(M^{l_i}, M^{l_j}) = \frac{M^{l_i} \cdot M^{l_j}}{||M^{l_i}|| \cdot ||M^{l_j}||}. \tag{2}$$

A higher sim_{sem} indicates greater semantic similarity between consecutive layers, indicating redundancy. Removing these redundant layers aids in model restructuring, while focusing retraining efforts on only the essential, non-redundant layers facilitates model re-adaptation. This strategy, however, only targets redundant layers without considering redundant neurons, causing limitations to be detailed in Section 2.4.

2.3 DNNs Slicing

DNNs, essentially programs made up of artificial neurons, have seen recent innovations in "slicing" techniques. These techniques, although termed as "slicing," essentially represent advanced methods for **neuron selection**. The key idea behind DNN slicing is to evaluate and identify the most influential neurons in a network for various tasks [63, 65]. In this context, DNN slicing is closely related to *neuron contribution metrics*, which quantify the impact of each individual neuron on the overall performance of the model, particularly on a dataset D. The essence of DNN slicing research is not only to partition the network but to systematically select neurons that significantly contribute to the network's functionality, based on their contribution scores. A *neuron selection strategy* is then applied to identify and extract these critical neurons, thereby enabling focused modifications, optimizations, or analysis of the network's performance.

Specifically, the process could be formulated as follows: we calculate $\mathbf{C}_n^{l,D}$ for each neuron n in layer l based on a dataset D. The *neuron contribution* of the layer l composed of neuron set $\mathbf{N}^l$ could be represented as $\mathbf{C}^{l,D} = \{\mathbf{C}_{n_0}^{l,D}, \mathbf{C}_{n_1}^{l,D}, ..., \mathbf{C}_{n_{|\mathbf{N}^l|-1}}^{l,D}\}$. We further determine the *selection* of critical neurons according to the calculated neuron contribution and denote the selected critical neurons as $\binom{l,D}{k}$:

$$\binom{l,D}{k} = \text{top}k(\mathbf{N}^l, \mathbf{C}^{l,D}) \tag{3}$$

where top$k(\cdot)$ represents the top k maximum instances of the input set $\mathbf{N}^l$ based on certain metrics. We can further depict the critical neurons of the model f as $\binom{D}{k} = \{\binom{l_0,D}{k}, \binom{l_1,D}{k}, ..., \binom{l_{L-1},D}{k}\}$. Note the proportion of selected critical neurons is directly controlled by hyperparameter k.

The predominant approaches of *neuron contribution metric* design for neuron n are as follows: ❶ **Avg** (Average of Neuron Activation [50]). We compute the mean activation value for each neuron across the dataset, defined as: $avg_n(D) = \frac{\sum_{x \in D} f_n(x)}{|D|}$. ❷ **Var** (Variance of Neuron Activation [17]). We compute the activation value variance for each neuron using the formula: $var_n(D) = \frac{\sum_{x \in D} (f_n(x) - \overline{f_n(x)})^2}{|D|}$. ❸ **Ens** (Ensemble of Neuron Activation). For each neuron, we compute its $avg_n(D)$ and $var_n(D)$ on dataset D, normalize both values to [0,1], and then combine them into a single ensemble-based contribution score. ❹ **DeepLIFT** (Interpretability-based Score [35, 63]). Given its interpretability, our analysis predominantly utilizes DeepLIFT [53] for computations. Specifically, DeepLIFT assigns contribution scores to each neuron based on the *gradient* during the backward propagation process. ❺ **Taylor** (Pruning-oriented Score [42]). Taylor utilizes *model weights* and *gradient information* to assess the contribution of neurons.

Leveraging the current neuron contribution metrics, there are intuitively two neuron selection strategies. The *global neuron selection method* calculates neuron contribution scores across the entire dataset D, followed by a selection of a specific range of neurons. In contrast, the *category-wise neuron selection approach* computes scores for each category of data (D_{c_0}, D_{c_1}, ...) separately, identifies a range of critical neurons for each category (*i.e.*, identifying $\binom{l,D_{c_0}}{k}, \binom{l,D_{c_1}}{k}, \ldots$), and ultimately merges these category-wise critical neurons for the entire dataset as a whole ($\binom{l,D}{k}$).

2.4 Motivation

Maintaining DNN models presents significant challenges, especially as both models and datasets continue to scale in size and complexity. Current state-of-the-art techniques [51] exhibit several limitations that impede effective model maintenance:

❶ **Insufficient Granularity.** Existing methods predominantly operate at the layer level, which restricts their ability to perform precise model restructuring and adaptation. This coarse granularity overlooks critical details at

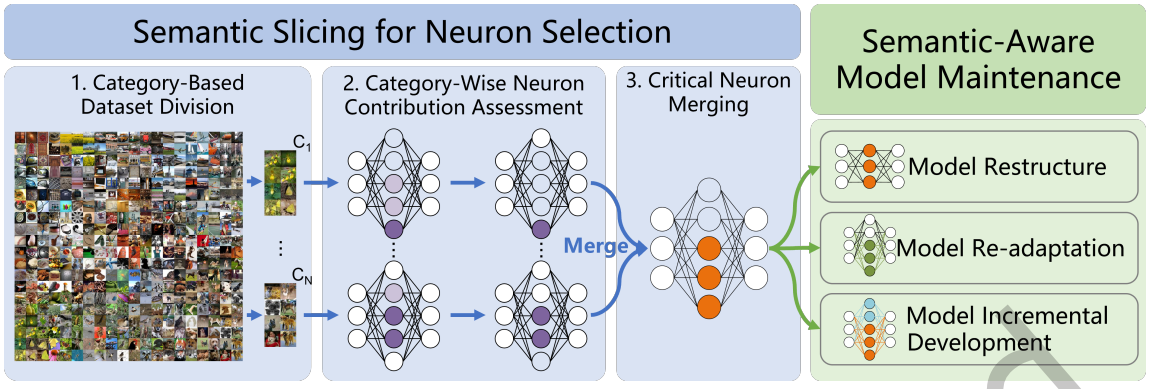

Fig. 1. Overview of NeuSemSlice Framework. In Semantic Slicing, Neuron Contributions Are Highlighted in *purple*, With Darker Shades Signifying Greater Contributions. Critical Neurons Are Indicated in *orange*. For Model Restructure, We Preserve Only These Critical Neurons. During Model Re-Adaptation, We Train Solely the Critical Neurons, Marked in *green*. For Model Incremental Development, Only the Non-Critical Neurons Are Trained, Shown in *blue*.

the neuron level, where fine-grained adjustments to individual neurons—especially those supporting specific functionalities—can significantly enhance model performance and efficiency.

❷ **Ineffectiveness for Relatively Small Models.** The term "small models" in the context of this paper often refers to models that are relatively small compared to the size of their training datasets, rather than being absolutely small. DeepArc [51] has demonstrated that layer-level modularization is most effective when the model's scale significantly exceeds that of the dataset. For instance, DeepArc's experiments utilized large models such as ResNet-110, Wide-ResNet-38, and VGGNet-19 trained on relatively small datasets like MNIST, FMNIST, and CIFAR-10. Our own experiments (RQ1) further confirm that when the disparity between model size and dataset size is reduced, the benefits of layer-level modularization diminish or disappear entirely. This indicates that layer-level maintenance techniques are less effective for models that are not excessively large relative to their training data. Consequently, there is a need for more refined, neuron-level approaches to ensure efficient and scalable model maintenance across varying model and dataset scales.

❸ **Lack of Incremental Model Development.** Incremental model development is a crucial aspect of model maintenance that remains inadequately addressed by current techniques. Existing approaches struggle to identify and preserve key components at the neuron level [10], which are essential for maintaining the model's functionality. This limitation poses significant challenges in developing new features or functions while ensuring that existing capabilities are retained, especially when relying solely on layer-level adjustments. A neuron-level maintenance strategy would enable more precise and incremental updates, facilitating the continuous evolution of DNN models without compromising their existing performance.

Addressing these limitations requires a shift towards more granular maintenance methodologies. By adopting neuron-level strategies, it becomes possible to achieve finer control over model restructuring and adaptation, enhance the effectiveness of maintenance for models of varying sizes relative to their datasets, and support incremental development processes. This approach promises to provide a more flexible and robust framework for DNN model maintenance, ultimately leading to more efficient and scalable deployment of deep learning models in diverse applications within the field of software engineering.

2.5 NeuSemSlice Overview

To address the issues, we propose NeuSemSlice which focuses on neuron-level DNN semantic decomposition and involves slicing DNNs to identify and isolate critical neuron components, facilitating a variety of model

maintenance tasks. Figure 1 shows a detailed workflow of our NeuSemSlice framework, which mainly consists of two modules: *Semantic Slicing* and *Semantic-aware Model Maintenance*.

Semantic slicing. This module initially identifies and extracts crucial neurons across the entire model. These neurons are representative of the semantics inherent in each layer of the model. By focusing on a more detailed level, it overcomes the initial two limitations, enabling more precise restructuring and adaptation. Additionally, it improves the ability to identify redundant neurons in the smaller layers of models.

Semantic-aware model maintenance. By utilizing the aggregated critical neurons in the model, this approach supports various applications, offering more robust solutions for long-term DNN model maintenance. Beyond restructuring and re-adapting the model, this method also addresses incremental model development. It specifically counteracts catastrophic forgetting, which occurs when an AI model learns new information, by stabilizing the critical neurons' parameters. This enables the model to incrementally develop functions with limited access to the original training data.

In Section 3, we will provide a detailed introduction to the design methodology behind *Semantic Slicing*. Following that, in Section 4, we will discuss *Semantic-aware Model Maintenance*: the application of *Semantic Slicing* in the context of model maintenance tasks.

3 SEMANTIC SLICING

As discussed in Section 2.3, various neuron contribution metrics and two possible neuron selection strategies can be adopted for neuron-level DNNs slicing. Yet, how well these methods perform in model maintenance tasks has not been fully examined. To address this, Section 3.1 details a pilot study designed to evaluate their effectiveness and efficiency in maintaining models. Following these experiments, we present our approach to semantic slicing in Section 3.3, guided by the insights gained.

3.1 A Pilot Study on Existing DNNs Slicing Strategies

Global Slicing. One possible strategy is the global approach, which calculates contribution metrics across the entire dataset and ranks all neurons accordingly. This process, outlined in Section 2.3, results in five distinct sets of critical neurons identified using different metrics. We then group neurons into different percentile ranges (*e.g.*, top 0–10%, 10–20%, etc.) and devise a strategy to evaluate the importance of each set to determine the most effective contribution metric. Specifically, to assess and compare the significance of the critical neurons identified by various metrics, we adopt the method from Xie et al. [63] by zeroing the outputs of these neurons and observing the impact on model predictions. Greater changes in predictions suggest higher neuron significance.

The study is conducted on CIFAR-10 [25], using VGG-16 [54] for classifying. Table 1 shows that in a global setting, all neuron contribution metrics have limitations. Surprisingly, masking neurons deemed less critical often leads to more significant accuracy losses, suggesting a misalignment between globally calculated contributions and the neurons' true importance. For example, using the Avg metric, masking 20%→30% of neurons has a greater impact than masking the top 10%→20%, as highlighted by the green arrows in the table. The results suggest that global slicing is not effective for model maintenance tasks.

Category-wise Slicing. Another possible strategy is the category-wise approach. Acknowledging that neuron contributions can vary widely between categories, which may limit the effectiveness of global DNN slicing, we conduct a category-specific analysis. We calculate the five metrics for each category and then conduct masking experiments on data from each category (*i.e.*, zeroing the outputs of neurons deemed critical for that category) to assess accuracy on a per-category basis. The averaged and individual category results are shown in Tables 2 and 3, respectively. Masking the top 10% of neurons shows that certain metrics (*e.g.*, Avg and Ens) drastically lowered classification performance, nearly to zero. This demonstrates that a category-specific approach is more effective in identifying critical neurons than that of the global-based approach.

Table 1. Model Accuracy After Masking Selected Critical Neurons Globally Calculated by Different Contribution Metrics. The Anticipated Trend in Model Accuracy Is an Increase (Signified by ↑), Correlating With the Diminishing Contribution of Masked Neurons. Nonetheless, Within the Context of Global Slicing, Numerous Anomalous Instances of Decreased Model Accuracy Are Observed, as Indicated by ↓.

VGG16-CIFAR10					
performance after masking	Avg ↑	Var ↑	Ens ↑	DeepLIFT ↑	Taylor ↑
0%->10%	42.85%	46.78%	50.64%	25.78%	41.29%
10%->20%	65.45%	76.91%	62.27%	36.83%	47.94%
20%->30%	60.32%↓	71.28%↓	57.82%↓	35.49%↓	60.13%
30%->40%	69.15%	69.53%↓	70.53%	65.94%	64.67%
40%->50%	65.34%↓	49.47%↓	62.55%↓	63.84%↓	72.05%

Table 2. The Model Accuracy Averaged by Category After Masking Critical Neurons Derived by Various Metrics in Categorical Settings. The Anticipated Trend in Model Accuracy Is an Increase (Signified by ↑), Correlating With the Diminishing Contribution of Masked Neurons.

VGG16-CIFAR10					
performance after masking	Avg ↑	Var ↑	Ens ↑	DeepLIFT ↑	Taylor ↑
0%->10%	0.00%	0.09%	0.00%	0.30%	11.80%
10%->20%	0.07%	5.16%	0.12%	8.48%	27.94%
20%->30%	1.25%	11.02%	2.49%	21.49%	48.24%
30%->40%	31.39%	37.81%	19.58%	59.12%	53.90%
40%->50%	55.45%	58.63%	57.61%	80.79%	62.64%

3.2 Identified Challenges for Semantic Slicing

Identifying critical neurons for model maintenance involves two main challenges: the optimal proportion of critical neurons varies by category and by layer. ❶ **Different category-wise distribution.** Integrating category-specific critical neurons into a set for the entire dataset is complex as the ideal proportion of these neurons differs across categories. For example, masking 10% to 20% of critical neurons in Category 4 hardly affects performance, suggesting redundancy. In comparison, masking 30% to 40% in Category 1 significantly impacts performance, indicating their importance for that category, as shown in Table 3. ❷ **Different layer-wise distribution.** The distribution of critical neurons changes across layers, often becoming sparser in higher layers. Current methods propose a uniform selection range for all categories and layers, which can either overlook crucial neurons or include too many non-critical ones, affecting accuracy and efficiency in model maintenance. Therefore, determining the optimal range of critical neurons for each category and layer is essential for effective DNN slicing.

However, achieving the goal is a challenging endeavor. The challenge lies in the impracticality of conducting an exhaustive search across all possible combinations of critical neuron proportions for each layer and category to assess performance after merging. Both the exhaustive search and the post-merging evaluation are inefficient in their current forms. To address this, we propose *Semantic Slicing*, to be detailed next in Section 3.3.

Table 3. Model Performance on Selected Categories After Masking Selected Critical Neurons Calculated by Different Contribution Metrics. The Anticipated Trend in Model Accuracy Is an Increase (Signified by ↑), Correlating With the Diminishing Contribution of Masked Neurons.

VGG16-CIFAR10 (DeepLIFT)				
	Category 1 ↑	Category 4↑	Category 7 ↑	Category 10 ↑
0%->10%	0.00%	3.00%	0.00%	0.00%
10%->20%	0.00%	81.80%	0.00%	0.00%
20%->30%	13.30%	80.40%	31.70%	5.90%
30%->40%	33.20%	90.20%	39.20%	86.70%
40%->50%	78.50%	95.80%	90.80%	96.30%

3.3 The Design of Semantic Slicing

The essence of existing DNNs slicing methods revolves around employing a contribution metric and then taking a neuron selection strategy to determine the range of critical neurons. Compared with these approaches, the key novelty of semantic slicing lies in the semantic-based neuron selection strategy. The aim of semantic slicing is to address the issue of determining the appropriate k value in Equation 3 for various layers and categories. To address this issue, we here recognize that the crux of choosing critical neurons lies in *selecting those neurons whose semantics are adequately representative of the entire layer.*

Problem Definition. Inspired by prior layer similarity evaluation works [45, 51], we formally reformulate the task of selecting top k neurons for each layer and category as follows. For each layer l and category c, identify k neurons such that they fulfill the condition: $\text{sim}_{\text{sem}}(f_k^{l,c}, f^{l,c}) > \Theta$, where Θ represents a unified threshold applicable across various categories and layers, and $\Omega_k^{l,c}$ is the selected neuron set. Employing the CKA metric as in Equation 2, we calculate a similarity metric in the range of $[0, 1]$ as follows:

$$min|^{l,c}| \quad \text{subject to} \quad \frac{M^{l,c} \cdot M^{l,c}}{||M^{l,c}|| \cdot ||M^{l,c}||} > \Theta, \tag{4}$$

where ^{l,c} is a neurons subset of layer l selected on category c, and $M^{l,c}$ is the embedding landscape calculated on category c according to Equation 1. Given the neuron contribution set $\mathbf{C}^{l,c} = \{\mathbf{C}_{n_0}^{l,c}, \mathbf{C}_{n_1}^{l,c}, ..., \mathbf{C}_{n_{|\mathbf{N}^l|-1}}^{l,c}\}$, we can further reformulate the given problem into a variant of the *Prefix Sum Problem* [2]. Specifically, the objective is to identify a particular value of k such that $\text{sim}_{\text{sem}}(f_k^{l,c}, f^{l,c}) > \Theta$ given the layer-wise contribution value set $\mathbf{C}^{l,c}$.

Algorithm Implementation. To address the aforementioned issue, we adopt a *Linear Scan* strategy as described by Algorithm 1. We break down the implementation into three steps:

(1) **Initialization:** We begin by selecting the top i neurons with the highest contribution scores from $\mathbf{C}^{l,c}$ to form the initial set ^{l,c}. At this stage, ^{l,c} contains a small but promising subset of neurons for category c at layer l. Using this initial set, we compute $\text{sim}_{\text{sem}}(f^{l,c}, f^{l,c})$ to assess how well it captures the semantic characteristics of the entire neuron population $\mathbf{N}^l$.

(2) **Iteration Until Threshold:** If the current similarity does not surpass the predefined threshold Θ, we continue the selection process. This involves identifying the next top i neurons from the remaining pool $\mathbf{N}^l \setminus {}^{l,c}$ according to their contribution scores. By iteratively adding these high-contribution neurons, we gradually refine the subset ^{l,c}, ensuring that it becomes more semantically aligned with $f^{l,c}$.

(3) **Stopping Condition:** The iteration proceeds until we achieve $\text{sim}_{\text{sem}}(f^{l,c}, f^{l,c}) > \Theta$. Once the threshold is met, the algorithm terminates, and the resulting set ^{l,c} is deemed critical for capturing the semantic

[2]The Prefix Sum problem, in its simplest form, involves computing an array of cumulative sums (prefix sums) of a given sequence of numbers.

Algorithm 1: Linear Scan Algorithm.

Data: All neurons $\mathbf{N}^l$ of layer l, the neuron contribution set $\mathbf{C}^{l,c}$, the semantic similarity threshold Θ, the selection interval i.
Result: The critical neuron set for category c at layer l: ^{l,c}.

$\Omega^{l,c} \leftarrow \text{top } i(\mathbf{N}^l, \mathbf{C}^{l,c})$
while $\text{sim}_{\text{sem}}\left(f^{\Omega^{l,c}}, f^{l,c}\right) < \Theta$ **do**
  $\Omega^{l,c} \leftarrow \Omega^{l,c} \cup \text{top } i(\mathbf{N}^l \setminus \Omega^{l,c}, \mathbf{C}^{l,c})$

features of category c at layer l. To further reduce the computational load, practitioners may choose a larger increment i, thereby requiring fewer iterations and faster convergence.

Categorical Critical Neurons Aggregation. After determining the critical neurons for each category, they can be combined on a category basis to establish the critical neuron set for the entire model, as shown in Figure 1. For an entire dataset D containing m categories, this can be expressed as: $^{l,D} = {}^{l,D_{c_0}} \cup {}^{l,D_{c_1}} \cup \ldots \cup {}^{l,D_{c_{m-1}}}$. We can further represent the critical neuron set of the model f as $= \{^{l_{0,D}}, ^{l_{1,D}}, ..., ^{l_{L-1,D}}\}$. Given the neuron contribution metrics $\mathbf{C}$, the selection threshold Θ directly determines , simplifying the process by eliminating the need to consider varying optimal neuron proportions across categories and layers.

4 SEMANTIC-AWARE MODEL MAINTENANCE

4.1 Model Restructure

Identifying critical neurons enables the removal of non-critical ones, keeping only essential neurons in each layer. This reduces the model's size by eliminating 'redundant' elements while preserving prediction accuracy, as shown in Figure 2. This process uses an indicator function $I(n)$ for each neuron n to decide whether to retain it based on its contribution. The model restructure strategy can be expressed as:

$$I(n) = \begin{cases} 1 & \text{if } n \in , \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Then each neuron n' in the restructured model f' could be expressed as $n' = n \cdot I(n)$. Note that the threshold value Θ can be dynamically adjusted to balance model performance and size during restructuring. By adjusting Θ, this method allows for a restructured model that varies in accuracy and size to suit diverse needs. Such restructuring will lead to an effective decrease in the storage space.

Moreover, some existing pruning techniques (e.g., [11, 12, 29, 43]) primarily operate at the weight level using unstructured approaches. These methods rely on approximate calculations to reduce weights while maintaining approximate model performance. In contrast, our proposed model restructure method identifies semantic slices (i.e., critical neurons) at the neuron level and removes non-critical neurons to compress the model. By leveraging model restructure, our method effectively complements existing pruning techniques, enabling these weight-based pruning methods to focus on redundant weights within critical neurons, thereby accelerating the pruning process and achieving better model compression.

4.2 Model Re-adaptation

Model re-adaptation improves adaptability and robustness by focusing on training parameters related to critical neurons. Previous research [34, 63, 64] has shown that in cases of poor performance, such as adversarial examples, adversarial noise is amplified and propagated through critical neurons. Therefore, the fundamental cause is the abnormal behavior of critical neurons leading to errors, while other neurons are less affected. Retraining these

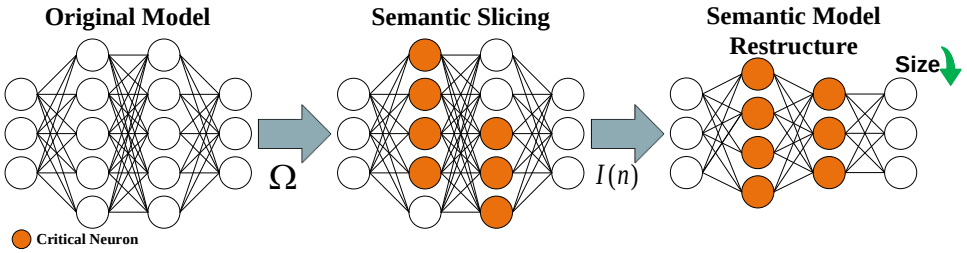

Fig. 2. Model Restructure Based on Semantic Slicing. We Only Retain the Critical Neurons for Model Restructure.

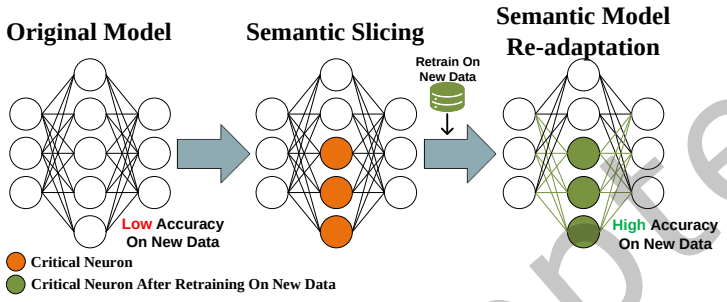

Fig. 3. Model Re-Adaptation Based on Semantic Slicing. We Train Only the Critical Neurons for Model Re-Adaptation.

critical neurons' parameters can thus more efficiently repair and enhance the model in terms of cases with poor performance. The complete model re-adaptation workflow is shown in Figure 3.

Using adversarial samples as an example, the retraining strategy explicitly incorporates both clean and adversarial inputs into the training objective. By including adversarial examples (adv_x, y) alongside their benign counterparts (x, y), the retraining process directly addresses the model's susceptibility to adversarial perturbations. Concretely, the loss function is defined as:

$$\mathcal{L} = \mathcal{L}_c(x, y; \theta) + \mathcal{L}_c(adv_x, y; \theta), \tag{6}$$

where L_c denotes the cross entropy, adv_x is the adversarial counterpart of the sample x, and the notation θ represents the parameters in model f. By summing the losses from both clean and adversarial samples, this objective ensures that the critical neurons learn to correctly classify benign inputs while simultaneously increasing their resistance to adversarial noise. The inclusion of adversarial data in the objective effectively provides stronger gradient signals focused on those regions of the parameter space associated with vulnerability, guiding critical neurons to adjust their representations.

To implement this targeted retraining, the parameter update strategy is carefully restricted. Instead of applying gradient-based updates to all parameters in the model, the procedure focuses exclusively on the parameters associated with the critical neuron set . Formally, given a learning rate η, the parameter update rule is:

$$\theta' = \theta - \eta \cdot \nabla_\theta \mathcal{L}, \tag{7}$$

This selective update confines optimization to parameters identified as critical, effectively targeting the root causes of performance issues.

4.3 Model Incremental Development

Model incremental development refers to the ability of a DNN model to learn from new data over time without forgetting previously learned information. This is crucial for creating DNN models that can adapt and evolve

in dynamic environments. Taking an image classification designed to recognize different types of animals as an example, we initially only have data to train a DNN to classify images into three categories: Dogs, Cats, and Birds (old task). After the initial training, we collect more data and want the DNN to learn to recognize two new categories Fish and Horses (new task). This poses a significant challenge for DNN models: ❶ In practical applications, the training data for the old task might be discarded to save storage space and we only have access to the data of the new task. When training only with the data of the new task, the incremental development of DNNs is prone to "catastrophic forgetting", where learning new tasks can lead to the loss of old tasks. ❷ Even with the training data of the old task, the training process is less resource-efficient because the DNN models have to be retrained with the entire dataset (new and old data).

This issue is particularly prominent in real-world scenarios, such as in IoT devices [40], where incremental development is essential. Due to space constraints, these devices might initially be trained on a limited set of data categories. However, as new data becomes available, it is often impractical to retrain the entire model from scratch or to delete previously learned categories.

Our strategy enables model incremental development by leveraging the identified critical neurons as stable anchors of previously acquired knowledge. By freezing the weights of these critical neurons during subsequent training phases, the model retains the essential semantic features learned for earlier tasks, thus mitigating catastrophic forgetting. At the same time, the non-critical neurons remain free to adapt, allowing the model to integrate new skills associated with a novel task without diluting its mastery of older tasks.

Concretely, suppose the model has been previously trained on task A. We begin by determining the critical neuron set that captures task A's essential semantics. These critical neurons, extracted through our earlier semantic slicing procedure, form a robust internal representation of the concepts necessary for solving task A. When the model encounters a new task B, it relies on these preserved, critical neurons to maintain performance on task A while directing the training process for task B to the non-critical neurons (i.e., those not in). Figure 4 illustrates this idea: the critical neurons, having proven essential for the original task, remain fixed, while the rest of the network updates to learn the new task.

This approach ensures that the parameters governing the original task's key semantic representations remain intact, thus safeguarding previously learned knowledge. The model's representational capacity outside is repurposed for incremental learning, enabling it to acquire new skills with minimal interference to existing knowledge. Through this selective updating mechanism, the model incrementally expands its repertoire of competencies in a stable and controlled manner.

Formally, for training on the new task B, we adopt the following loss function:

$$\mathcal{L} = \mathcal{L}_c(x_B, y_B; \theta), \tag{8}$$

where $\mathcal{L}_c$ denotes the cross-entropy loss and (x_B, y_B) are the training samples and labels from task B. Given this objective, the parameter update rule becomes:

$$\theta'_{\mathbf{N}\setminus} = \theta_{\mathbf{N}\setminus} - \eta \cdot \nabla_{\theta_{\mathbf{N}\setminus}} \mathcal{L}, \tag{9}$$

where η is the learning rate, $\mathbf{N}$ represents the full set of neurons, and $\mathbf{N} \setminus$ is the set of neurons not identified as critical for task A.

In this manner, only the parameters not involved in encoding the critical semantics from task A are adjusted, allowing the model to assimilate the new task B without eroding the prior knowledge. During inference, the model employs the frozen, critical parameters for predictions on task A, ensuring stable performance on previously learned content. For task B, it leverages the updated subset of parameters dedicated to the new skill, allowing the model to switch between tasks while maintaining overall system integrity and performance.

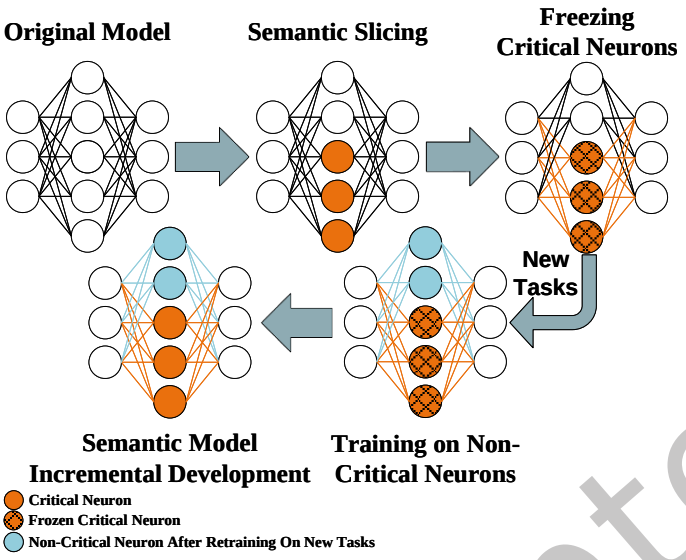

Fig. 4. Model Incremental Development Based on Semantic Slicing. We Train Only the Non-Critical Neurons for Model Incremental Development.

5 EVALUATION

In this section, we evaluate the performance of NeuSemSlice. We first outline the experimental setup and then we introduce our evaluation aiming to answer the following research questions:

- **RQ1:** How does semantic slicing outperform other DNN slicing methods?
- **RQ2:** Is NeuSemSlice effective for model restructure?
- **RQ3:** Is NeuSemSlice effective for model re-adaptation?
- **RQ4:** Is NeuSemSlice effective for model incremental development?

5.1 Evaluation Setup

To ensure the comprehensiveness and depth of the experiments, we select four widely utilized datasets in the deep neural network domain—**MNIST** [28], **Fashion-MNIST** [62], **CIFAR-10** [25], and **ImageNet** [9]—along with five network models for experimental evaluation: **MLP**, **AlexNet** [26], **VGG-16** [54], **ResNet-101** [14], and **VGG-19** [54]. These models and datasets are not only widely used in the deep learning domain but also extensively applied in software engineering research [6, 16, 18, 20, 30, 38, 60].

❶ **MNIST and Fashion-MNIST.** For training, we use a Multilayer Perceptron (MLP) model with two hidden layers containing 512 and 256 neurons, respectively.

❷ **CIFAR-10.** Three deep neural network models—AlexNet, VGG-16, and ResNet-101—are utilized for training on this dataset.

❸ **ImageNet.** Following the setup in [63], we select data from the top 10 categories for experiments, employing the VGG-19 model.

Based on the datasets and models mentioned, six experimental settings are constructed: MLP-MNIST, MLP-FMNIST, AlexNet-CIFAR10, VGG16-CIFAR10, ResNet101-CIFAR10, and VGG19-ImageNet. These experiments include a variety of configurations, ranging from small to large models and datasets. In each experimental setting, we train at least five different models to repeat the experiments to reduce randomness and ensure robustness. The results are presented as mean values to enhance the experiment reliability and precision.

All these experiments are conducted on the Intel Xeon Silver 4214 Processor with 2 Tesla V100 GPUs with 32GB memory. We have implemented our tool based on Pytorch [48]. The source code, related information of datasets and models, and complete experimental results are released at [44].

5.2 RQ1: The performance of semantic slicing

In RQ1, we aim to evaluate and compare various slicing methods. Specifically, we follow [34, 63] to mask the redundant components of the DNN model (*i.e.*, zeroing non-critical neurons) and then compare the model accuracy with only the critical neurons retained. Greater accuracy suggests the slicing method could better identify critical neurons with precision.

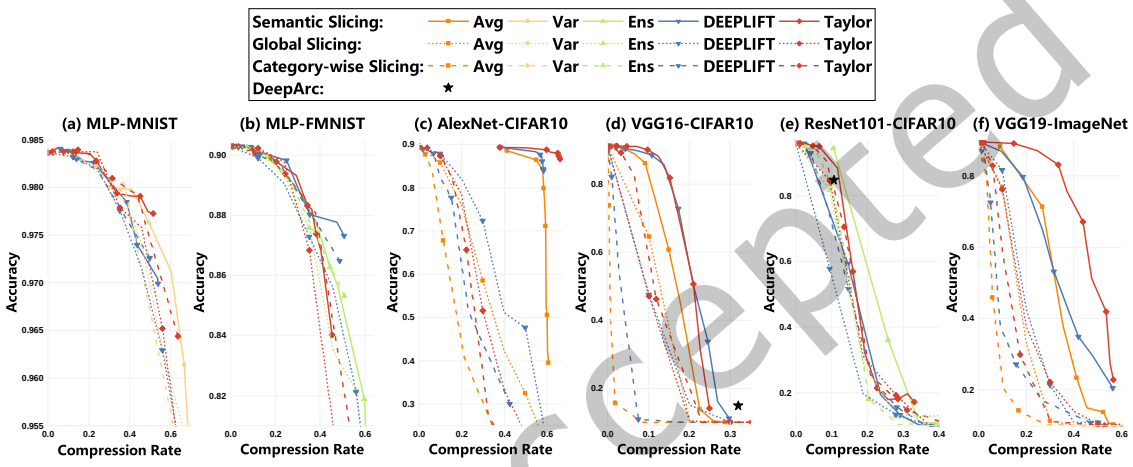

Fig. 5. Comparison Results of Performance Across Different Slicing Methods Are Presented. Here, We Only Present the Results for the Top 3 Contribution Metrics for Better Observation. Each Dot in the Figure Represents the Model Performance (*i.e.*, Compression Rate and Accuracy) After Masking the Redundant Components of Neurons Identified by a Slicing Method, With DeepArc Represented by a ★. The Lines Representing Semantic Slicing Are Plotted by Varying Θ Values From 0.9 to 0.99. For Global and Category-Wise Slicing, the Top 10%, 20%, Up to 100% of Neurons per Layer or Category Are Selected for Slicing Respectively. The Results for DeepArc Are Obtained by Removing Redundant Layers With Semantic Similarity Greater Than 0.99. Ideally, Dots (Lines) Located at the **Upper Right** Are Favored for Their Higher Compression Rates and Better Accuracy Performance, Which Also Suggest That the Corresponding Slicing Methods Better Identify Critical Neurons in the Model.

Detailed Configurations. In RQ1, we compare the following three slicing methods: global slicing, category-wise slicing, and semantic slicing. To assess neuron contribution and select critical neurons, we utilize five neuron contribution metrics: Avg, Var, Ens, DeepLIFT, and Taylor. As outlined in Section 3.1, global slicing and category-wise slicing choose top 10%, 20%, ... up to 100% of neurons in each layer or category for slicing respectively. Additionally, we introduce DeepArc for comparative analysis. Following the settings used in [51], we set the threshold at 0.99 to identify redundant layers in the model. Subsequently, we remove these redundant layers and compare the model accuracy when only critical semantic layers are retained.

Semantic slicing identifies critical neurons for each category using a set of thresholds $\Theta = 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99$, following the similar settings used by [51]. Then it merges them into semantic slices, as described in Section 3.3. For RQ1, the compression rate is defined as the percentage reduction of parameters in the model, while model accuracy denotes the predictive accuracy of the model slices on

Table 4. Statistical Comparison of Semantic Slicing With Global and Category-Wise Slicing Across Various Experimental Settings.

Settings	Semantic vs Global		Semantic vs Category-wise	
	p-value	Cohen's d	p-value	Cohen's d
MLP-MNIST	**	**	*	*
MLP-FMNIST	***	***	**	**
AlexNet-CIFAR10	***	***	***	***
VGG16-CIFAR10	**	***	***	***
ResNet101-CIFAR10	***	***	***	***
VGG19-ImageNet	***	***	***	***

p-value: *** < 0.01 (highly significant), ** < 0.05 (significant), * > 0.05 (not significant).
Cohen's d: *** (> 0.8, large effect), ** (0.5 - 0.8, medium effect), * (0.2 - 0.5, small effect).

the test dataset. We conducted comparisons of slicing precision across the previously mentioned six experimental settings.

Comparison with Global Slicing and Category-wise Slicing. Figure 5 compares the slicing precision differences among semantic slicing, global slicing, and category-wise slicing (the complete experimental results can be found in [44]). For complex models like AlexNet and VGG16, global slicing and category-wise slicing can only maintain certain model performance at very low compression rates, with accuracy significantly declining as the compression rate increases. Semantic slicing demonstrates significant improvement over the other two techniques across all five neuron contribution metrics. For example, in the AlexNet-CIFAR10 case (Figure 5(c)), semantic slicing uses the neuron contribution metric Taylor to maintain model performance, with only a 0.38% loss in accuracy at a 50% compression rate. Conversely, the other two techniques experience significant accuracy losses of around 70% at the same compression rate. Similarly, on VGG16 using the DeepLIFT metric (Figure 5(d)), semantic slicing at around a 10% compression rate reduces accuracy loss by 41.18% and 77.77% compared to global slicing and category-wise slicing.

On larger models like ResNet101 and VGG19 (Figure 5(e) and (f)), global slicing and category-wise slicing are also less effective. Even for datasets like MNIST and Fashion-MNIST (Figure 5(a) and (b), which require fewer neurons for high accuracy, semantic slicing still slightly outperforms them, maintaining higher model accuracy across all evaluated neuron contribution metrics at the same levels of compression rate.

To further clarify the extent of the differences among these approaches, we conduct paired t-tests and compute effect sizes using Cohen's d on the AUC values obtained within the same range from the compression rate-accuracy curves (see Table 4). These tests confirm that semantic slicing achieves statistically significant improvements over global slicing in all experimental configurations (p-values < 0.05) and, except for the MLP-MNIST setting, also significantly outperforms category-wise slicing. Moreover, the corresponding effect sizes are medium to large in most cases, indicating that the advantages of semantic slicing are not only statistically robust but also practically meaningful.

These results adequately confirm that an inadequate account of the uneven distribution of critical neurons across different categories and network layers leads to poor slicing performance. This highlights the advantage of the proposed technique, which can dynamically identify critical features and components of the model based on semantic similarity.

The Effects of DeepArc. For DeepArc, we identify redundant layers in six settings based on layer similarity, as shown in Figure 5. Firstly, for the MLP, the model only has two hidden layers. When any layer is removed, we follow the model restructuring detailed in DeepArc and introduce a "sewing layer" to align the input and output of the layers. However, this results in no change in the size of the model, indicating that DeepArc cannot effectively

identify redundancies in such models. In the other four configurations, DeepArc only identifies redundant layers with a similarity greater than 0.99 in VGG16 and ResNet101.

For AlexNet, the maximum layer similarity is 0.95. If the threshold is lowered to 0.95 and the corresponding redundant layer in the module is removed, a fully connected layer containing 87.97% of the model's parameters is eliminated, reducing the model accuracy to 10.94%. The results for VGG19 are similar, with the highest layer similarity being only 0.97. For VGG16, removing redundant layers within the module leads to a model compression rate of 32.06% but an accuracy of only 14.84%. For ResNet101, after removing 10.68% of redundant layers, the accuracy remains at 84.77%. However, its precision in identifying critical model components is still not as high as that of NeuSemSlice. These results suggest that DeepArc is relatively effective only in large models with small datasets, due to the difficulty in sharing semantics in smaller models or models with lower redundancy.

Comparing different Neuron Contribution Metrics. Our NeuSemSlice framework is orthogonal to the neuron contribution metrics, meaning it can be integrated seamlessly with a range of contribution metrics. Through our framework, we can fairly compare different neuron contribution metrics. We can observe that in relatively simple models like MLP, the performance of these metrics is close. They all effectively assess neuron contributions, enabling the precise selection of critical neurons. For example, in the case of MLP-MNIST, even at a compression rate of 70%, the model's accuracy only drops by approximately 4%. In contrast, for more complex models such as AlexNet and VGG16, DeepLIFT and Taylor metrics perform better. For AlexNet, the Taylor metric achieves a mere 3% accuracy drop at a compression rate of 66.05%; for VGG16, both DeepLIFT and Taylor metrics maintain the model's accuracy within a 2% decrease at a compression rate of approximately 10%. For larger models like ResNet101 and VGG19, the performance of the metrics is similarly close in ResNet101, with Ens achieving only a 1.54% drop in accuracy at a 10.56% compression rate, while in VGG19, Taylor significantly outperforms other metrics, maintaining a 2.54% accuracy drop at a 24.67% compression rate.

The Var metric exhibits lower accuracy after model compression, which is especially evident in AlexNet and VGG16. We also calculate the time required to compute various metrics: Avg, Var, Ens, DeepLIFT, and Taylor, which take 5.4s, 5.5s, 5.6s, 30.9s, and 6.2s respectively on the MNIST dataset. In comparison to other methods, DeepLIFT exhibits significantly higher time consumption. *Overall, the DeepLIFT and Taylor metrics outperform the other three metrics, while DeepLIFT is less efficient.*

Answer to RQ1: Compared to global slicing and category-wise slicing, semantic slicing better identifies critical neurons with precision. At 50% compression rate, semantic slicing increases model performance from 23.24% and 11.55% to 89.20% compared to two other techniques respectively (see AlexNet-CIFAR10, neuron contribution metric Taylor, Figure 5(c)).

5.3 RQ2: Model Restructure

Detailed Configuration. To systematically assess NeuSemSlice for the model restructure task, we adopt the same experimental setup detailed in [51], and validate by comparing the compression efficiency and effects on the model before and after applying NeuSemSlice. Specifically, after removing the non-critical neurons of the model, we expect the compression technology to directly focus on the redundant parts within the critical neurons, improving the iteration efficiency[3]. Based on this, we apply four of the most advanced model compression techniques currently available—LAMP [29], Global [43], Uniform+ [12], and ERK [11]—for optimal compression results. Additionally, we also present the results of global slicing and category-wise slicing for comparison.

In our experiments, we take the following metrics to evaluate the performance of model compression. The Compression Rate (**CR**) indicates the reduction in parameter size, the Number of Iterations (**NI**) reflects the

[3]It is important to note that in this experiment and the subsequent RQ3 and RQ4, our method is based on the optimal configuration identified in RQ1, while global slicing and category-wise slicing use configurations with similar slice sizes under the same neuron contribution metric.

Table 5. Results of Model Restructure.

	MLP-MNIST			MLP-FMNIST			AlexNet-CIFAR10			VGG16-CIFAR10			ResNet101-CIFAR10			VGG19-ImageNet		
Compression Algorithm	CR ↑	NI ↓	MA ↑	CR ↑	NI ↓	MA ↑	CR ↑	NI ↓	MA ↑	CR ↑	NI ↓	MA ↑	CR ↑	NI ↓	MA ↑	CR ↑	NI ↓	MA ↑
LAMP	52.17%	7	98.26%	36.97%	9	90.01%	30.17%	7	89.49%	18.55%	4	91.28%	45.96%	12	90.77%	30.17%	7	**89.54%**
Global+LAMP	48.27%	5	98.26%	38.82%	7	89.78%	33.85%	6	87.87%	14.65%	1	46.84%	45.79%	10	76.29%	33.86%	6	76.31%
Category-wise+LAMP	50.82%	6	98.30%	39.19%	8	90.04%	33.95%	6	82.12%	20.08%	4	90.31%	46.03%	10	81.27%	33.92%	6	58.42%
NeuSemSlice+LAMP	**53.05%**	5	**98.32%**	**39.70%**	8	**90.21%**	**34.88%**	6	**89.52%**	**20.36%**	4	**91.29%**	**46.46%**	10	**92.11%**	**37.25%**	6	89.19%
Glob	40.95%	5	98.31%	33.66%	8	90.09%	60.28%	18	**89.44%**	22.62%	5	91.25%	53.67%	15	72.21%	22.62%	5	**89.42%**
Global+Glob	36.13%	3	98.33%	35.60%	6	89.80%	62.37%	17	87.75%	14.65%	1	47.00%	53.53%	13	46.29%	22.86%	3	76.15%
Category-wise+Glob	39.28%	4	98.31%	32.62%	6	90.18%	62.43%	17	81.99%	24.07%	5	90.28%	53.73%	13	47.82%	22.93%	3	58.15%
NeuSemSlice+Glob	**42.04%**	3	**98.37%**	**36.52%**	7	**90.18%**	**62.96%**	17	**89.44%**	**24.34%**	5	**91.28%**	**54.10%**	13	**73.96%**	**26.93%**	3	89.12%
Unif	40.95%	5	98.34%	40.13%	10	89.81%	55.99%	16	88.59%	14.26%	3	91.24%	51.23%	14	83.95%	55.99%	16	85.85%
Global+Unif	36.13%	3	98.29%	41.88%	8	89.64%	58.31%	15	87.08%	14.65%	1	47.19%	51.08%	12	55.52%	53.81%	13	67.19%
Category-wise+Unif	39.28%	4	98.28%	39.19%	8	89.89%	58.37%	15	81.60%	15.87%	3	90.21%	51.29%	12	58.68%	53.86%	13	46.46%
NeuSemSlice+Unif	**42.04%**	3	**98.36%**	**42.71%**	9	**89.95%**	**58.96%**	15	**88.68%**	**16.17%**	3	**91.31%**	**51.68%**	12	**84.03%**	**56.25%**	13	**87.50%**
ERK	34.39%	4	98.30%	30.17%	7	90.08%	55.99%	16	89.37%	26.49%	6	91.19%	14.26%	3	**95.66%**	26.49%	6	**89.58%**
Global+ERK	36.13%	3	98.31%	32.21%	5	89.73%	58.31%	15	87.66%	14.65%	1	46.95%	22.38%	3	80.87%	30.38%	5	76.23%
Category-wise+ERK	32.53%	3	98.34%	32.62%	6	90.18%	58.37%	15	82.15%	27.87%	6	90.34%	22.72%	3	82.84%	30.45%	5	58.31%
NeuSemSlice+ERK	**42.04%**	3	**98.33%**	**33.18%**	6	**90.24%**	**58.96%**	15	**89.41%**	**28.13%**	6	**91.28%**	**23.34%**	3	93.97%	**34.06%**	5	89.19%

efficiency of the compression process, and Model Accuracy (**MA**) measures the predictive performance on the test dataset.

Results and Findings. Table 5 presents the experimental results of NeuSemSlice in model restructure. Across all experimental settings, the application of NeuSemSlice in model restructure allows models to achieve higher compression ratios with fewer iterations, indicating that NeuSemSlice can improve both compression efficiency and effectiveness. Apart from the VGG19-ImageNet configuration, NeuSemSlice not only enhances compression efficiency and rates but also slightly improves model accuracy. Specifically, in six experimental settings, the average number of iterations required for model compression decreases from 8.67 to 7.50, enhancing efficiency by 13.46%, and the model compression ratio increases from 37.26% to 40.26%.

For global slicing and category-wise slicing, we observe that in simpler models like MLP-MNIST and MLP-FMNIST, their performance is similar to NeuSemSlice, both effectively improving compression efficiency. However, as the model complexity increases, NeuSemSlice's ability to accurately identify critical neurons becomes more apparent. In the case of AlexNet-CIFAR10, with the same number of compression iterations and similar compression ratios, NeuSemSlice achieves an average improvement of 1.67% in model accuracy compared to global slicing and a 7.30% improvement compared to category-wise slicing. For more complex models, such as VGG16-CIFAR10, ResNet101-CIFAR10, and VGG19-ImageNet, global slicing and category-wise slicing often result in significant accuracy drops. For instance, global slicing reduces the accuracy of VGG16-CIFAR10 to around 47%, while category-wise slicing drops the accuracy of VGG19-ImageNet to about 58%, whereas NeuSemSlice does not experience such issues. These results demonstrate that the application of NeuSemSlice in model restructure boosts the performance of model compression and enhances the performance of compressed models.

Answer to RQ2: For model restructure, NeuSemSlice not only elevates the efficiency of compression technology—an average increase of 13.46%—but also achieves approximately a 3% enhancement in the compression rate and a slight increase in accuracy.

5.4 RQ3: Model Re-adaptation

Detailed Configuration. To answer RQ3, we conduct an experiment using two prevalently-employed adversarial sample generation techniques, Fast Gradient Sign Method (FGSM) [13] and Projected Gradient Descent (PGD) [41]. We generate 1,000 adversarial samples, with FGSM reducing the model's average accuracy to 18.53%, and PGD to 3.8%. We aim to compare the Training Parameter Ratio (**TPR**) and the Retrained Accuracy (**RA**) of existing approaches and our proposed method. Specifically, the experiment settings include fully training (i.e., comparison

Table 6. Results of Model Re-adaptation.

		MLP-MNIST			MLP-FMNIST			AlexNet-CIFAR10			VGG16-CIFAR10			ResNet101-CIFAR10			VGG19-ImageNet		
	Method	TPR ↓	RA ↑	RAA ↑	TPR ↓	RA ↑	RAA ↑	TPR ↓	RA ↑	RAA ↑	TPR ↓	RA ↑	RAA ↑	TPR ↓	RA ↑	RAA ↑	TPR ↓	RA ↑	RAA ↑
FGSM	Fully Training	100.00%	100.00%	14.20%	100.00%	**98.25%**	10.10%	100.00%	98.40%	43.90%	100.00%	97.83%	13.10%	100.00%	100.00%	49.50%	100.00%	98.54%	**48.59%**
	DeepArc	75.01%	100.00%	13.40%	75.01%	97.18%	11.20%	99.18%	99.43%	50.40%	67.60%	98.80%	13.70%	56.26%	100.00%	53.60%	95.82%	98.44%	42.66%
	Global Slicing	33.98%	100.00%	13.60%	54.06%	95.00%	11.30%	40.10%	98.60%	41.10%	89.85%	97.05%	16.20%	90.56%	90.40%	52.60%	89.97%	98.44%	**48.59%**
	Category-wise Slicing	37.57%	100.00%	15.90%	51.14%	95.85%	11.10%	46.65%	98.35%	45.30%	88.05%	95.20%	13.60%	87.22%	91.40%	49.30%	89.89%	95.47%	46.88%
	NeuSemSlice	26.74%	100.00%	18.10%	49.26%	97.72%	11.40%	33.95%	99.55%	46.10%	87.41%	99.60%	15.70%	86.35%	100.00%	54.20%	85.22%	98.78%	47.66%
	NeuSemSlice+DeepArc	**24.17%**	100.00%	**21.20%**	**40.29%**	98.00%	**11.50%**	**33.21%**	**100.00%**	**52.70%**	**60.39%**	**99.88%**	**37.30%**	**47.30%**	100.00%	**54.60%**	**81.12%**	**98.85%**	44.22%
PGD	Fully Training	100.00%	100.00%	**14.00%**	100.00%	97.90%	6.00%	100.00%	97.68%	1.30%	100.00%	95.80%	**2.90%**	100.00%	99.30%	0.90%	100.00%	99.38%	22.81%
	DeepArc	75.01%	100.00%	12.00%	75.01%	97.37%	7.20%	99.18%	**99.95%**	0.10%	67.60%	99.53%	0.00%	56.26%	95.30%	4.20%	95.82%	99.11%	20.78%
	Global Slicing	33.98%	100.00%	11.90%	54.06%	95.95%	8.60%	40.10%	96.70%	1.00%	89.85%	94.85%	0.00%	90.56%	92.90%	2.60%	89.97%	96.41%	17.50%
	Category-wise Slicing	37.57%	100.00%	11.30%	51.14%	95.65%	**10.50%**	46.65%	96.20%	0.90%	88.05%	96.40%	0.00%	87.22%	96.40%	3.10%	89.89%	95.63%	17.34%
	NeuSemSlice	26.74%	100.00%	12.90%	49.26%	97.90%	5.80%	33.95%	99.30%	**1.40%**	87.41%	99.75%	0.00%	86.35%	**99.40%**	1.40%	85.22%	99.18%	20.16%
	NeuSemSlice+DeepArc	**24.17%**	100.00%	12.60%	**40.29%**	**97.92%**	8.10%	**33.21%**	98.98%	0.30%	**60.39%**	**99.83%**	0.00%	**47.30%**	98.93%	**4.40%**	**81.12%**	**99.41%**	**24.06%**

baseline), DeepArc, global slicing, category-wise slicing, NeuSemSlice, and a combination of NeuSemSlice with DeepArc[4] under identical training epochs. Additionally, this RQ further assesses the resilience of each method against similar adversarial attacks. We generate another 1,000 adversarial samples after the model re-adaptation, and obtain the model accuracy against the samples (*i.e.*, shown as Re-Attack Accuracy, **RAA**).

Results and Findings. Table 6 presents our experimental results in the six settings. Compared to the traditional method of fully training and DeepArc, our method demonstrates a significant reduction in the required model parameters for training, averaging only 61.49%. This represents an average decrease of 38.51% in training parameters compared to the fully training strategy and an average reduction of 16.66% relative to DeepArc. Moreover, our method surpasses fully training by an average of 0.67% and DeepArc by an average of 0.50% in Retrained Accuracy, demonstrating an advantage in both training cost-efficiency and effectiveness. Compared to global slicing and category-wise slicing, with similar training parameters, our method shows an average improvement of 2.91% and 2.89% in Retrained Accuracy, respectively. This highlights that the inaccurate identification of critical neurons leads to reduced Re-adaptation effectiveness. Although fewer training parameters are used than the fully training method, the training accuracy also decreases accordingly. Among the six methods, our method combined with DeepArc achieves a 0.73% increase in Retrained Accuracy compared to traditional training methods, with the lowest training cost (averaging only 47.75% of the parameters trained).

Our method combined with DeepArc also shows the best performance in terms of re-attack performance, achieving up to a 17.8% improvement in RAA. The results further demonstrate the superior performance of our method in model re-adaptation scenarios. However, we observe that all of the methods experience notable performance loss against re-attack, alerting us to design better methods against adversarial attacks in the future, to be discussed in Section 5.7. Finally, NeuSemSlice is comparable to DeepArc in efficiency, requiring only 93.53% of the time needed for full training on average. When NeuSemSlice is combined with DeepArc, it requires only 89.80% of the full training time. The detailed data is available at [44].

Answer to RQ3: Our approach reduces the cost of model re-adaptation by maintaining critical neurons, requiring only an average of 61.49% of the parameters. Our method also exhibits slight advantages in terms of efficiency and model accuracy over baselines.

5.5 RQ4: Model Incremental Development

Detailed Configuration. To answer RQ4, we set up two experimental scenarios. In the first scenario, we split the datasets MNIST, FMNIST, CIFAR10, and ImageNet into two subtasks, each containing five categories, labeled as task *A* (previous task) and task *B* (new task). For example, in the CIFAR10 dataset, which consists of ten distinct image categories, task A includes the first five categories, while task B includes the remaining five categories. In

[4]The term "a combination of NeuSemSlice with DeepArc" refers to retraining the critical neurons identified by our method in the critical modules recognized by DeepArc.

Table 7. Results of Model Incremental Development(Scenario I)

	MLP-MNIST			MLP-FMNIST			AlexNet-CIFAR10			VGG16-CIFAR10			ResNet101-CIFAR10			VGG19-ImageNet		
Method	TaskA ↑	TaskB ↑	Avg ↑	TaskA ↑	TaskB ↑	Avg ↑	TaskA ↑	TaskB ↑	Avg ↑	TaskA ↑	TaskB ↑	Avg ↑	TaskA ↑	TaskB ↑	Avg ↑	TaskA ↑	TaskB ↑	Avg ↑
Original	98.92%	0.00%	49.46%	97.44%	0.00%	48.72%	89.14%	0.00%	44.57%	92.52%	0.00%	46.26%	96.90%	0.00%	48.45%	92.15%	0.00%	46.08%
Retrain	0.00%	98.50%	49.25%	2.78%	83.28%	43.03%	0.00%	89.40%	44.70%	0.00%	90.80%	45.40%	0.00%	85.86%	42.93%	5.37%	88.92%	47.15%
Replay 5%	87.54%	98.32%	92.93%	83.96%	83.14%	83.55%	46.32%	88.94%	67.63%	54.42%	90.76%	72.59%	26.54%	85.00%	55.77%	85.62%	87.77%	86.69%
Replay 10%	91.86%	98.13%	95.00%	87.90%	83.20%	85.55%	59.70%	88.16%	73.93%	62.80%	90.12%	76.46%	39.40%	84.68%	62.04%	87.23%	88.38%	87.81%
Replay 100%	97.20%	97.48%	97.34%	94.44%	81.78%	88.11%	82.96%	85.66%	84.31%	86.98%	88.30%	87.64%	84.12%	90.18%	87.15%	90.46%	89.38%	89.92%
DeepArc	/	/	/	/	/	/	0.00%	84.84%	42.42%	20.44%	0.00%	10.22%	46.68%	0.00%	23.34%	0.00%	84.92%	42.46%
Global Slicing	93.58%	97.75%	95.66%	82.10%	78.46%	80.28%	69.88%	87.26%	78.57%	65.74%	86.44%	76.09%	96.92%	79.20%	88.06%	88.38%	89.46%	88.92%
Category-wise Slicing	96.85%	97.44%	97.14%	82.12%	81.60%	81.86%	69.68%	87.18%	78.43%	72.96%	86.54%	79.75%	95.34%	78.98%	87.16%	86.62%	88.92%	87.77%
NeuSemSlice	98.89%	97.20%	98.04%	92.12%	82.08%	87.10%	84.65%	87.60%	86.13%	85.77%	86.98%	86.37%	97.12%	82.52%	89.82%	91.77%	90.54%	91.15%
NeuSemSlice+Replay 10%	98.94%	97.17%	**98.05%**	96.96%	81.90%	**89.43%**	87.22%	86.66%	**86.94%**	91.38%	86.02%	**88.70%**	98.52%	81.46%	**89.99%**	92.54%	90.38%	**91.46%**

Table 8. Results of Model Incremental Development(Scenario II)

		Retrain	Replay 5%	Replay 10%	Global	Category-wise	NeuSemSlice
MLP-MNIST	TaskA	0.00%	91.93%	94.85%	94.22%	93.55%	98.95%
	TaskB	0.14%	85.89%	90.71%	97.54%	98.79%	97.43%
	TaskC	98.11%	97.96%	97.88%	97.88%	97.98%	97.93%
	Avg	32.75%	91.93%	94.48%	96.55%	96.77%	**98.11%**
MLP-FMNIST	TaskA	0.00%	58.93%	66.40%	84.90%	90.13%	93.57%
	TaskB	0.00%	55.47%	68.63%	91.03%	96.10%	96.73%
	TaskC	96.80%	95.58%	94.63%	96.15%	96.25%	96.15%
	Avg	32.27%	69.99%	76.55%	90.69%	94.16%	**95.48%**
AlexNet-CIFAR10	TaskA	0.00%	37.57%	57.17%	75.57%	87.83%	93.37%
	TaskB	0.00%	29.30%	50.83%	59.07%	70.17%	76.03%
	TaskC	94.63%	94.60%	93.63%	92.70%	94.10%	92.93%
	Avg	31.54%	53.82%	67.21%	75.78%	84.03%	**87.44%**
VGG16-CIFAR10	TaskA	0.00%	38.63%	54.77%	86.90%	76.57%	89.17%
	TaskB	0.00%	33.10%	45.67%	54.33%	60.80%	65.93%
	TaskC	94.53%	92.30%	91.30%	91.40%	89.65%	91.55%
	Avg	31.51%	54.68%	63.91%	77.54%	75.67%	**82.22%**
ResNet101-CIFAR10	TaskA	10.03%	17.90%	38.37%	91.90%	92.20%	92.30%
	TaskB	6.37%	15.70%	27.63%	64.60%	57.07%	62.63%
	TaskC	89.73%	90.70%	87.13%	86.80%	89.73%	89.35%
	Avg	35.38%	41.43%	51.04%	81.10%	79.66%	**81.43%**
VGG19-ImageNet	TaskA	0.00%	31.28%	32.44%	93.33%	92.31%	98.21%
	TaskB	0.00%	28.08%	29.74%	80.13%	83.72%	85.13%
	TaskC	92.31%	92.02%	91.44%	93.08%	93.46%	92.60%
	Avg	30.77%	50.46%	51.21%	88.85%	89.83%	**91.98%**

the second scenario, we further divide the datasets into three subtasks. For instance, task *A* and task *B* include three categories each, while task *C* includes four categories.

To simulate the process of incremental model development, for the first scenario, we start by training models exclusively on task *A* in six different experimental setups following the division. Then, we proceed to train these models on task *B* while aiming to preserve their performance on task *A*. The evaluation focuses on the accuracy for both task *A* and task *B*. Here, the accuracy on task *A* evaluates how well the models retain knowledge of the original task without forgetting, while the accuracy on task *B* gauges the models' ability to learn new tasks. The second scenario follows a similar process (with an additional round of new task learning). Here, the model first learns task A, followed by task B, and finally task C.

We would like to note that *this setup aims to simulate the actual scenario of model incremental development, where we typically have limited access to the training data of previous tasks.* Therefore, we introduce 'replay' as a baseline for comparison, which refers to relaxing the restrictions on accessing the data of previous tasks while learning a new task. This is done by revisiting some of the task A data to aid the model in remembering and maintaining its performance on the old task.

In particular, we aim to compare the following methods: naive retraining (*i.e.*, directly training with new task data only), Replay 5% (i.e., training data includes 5% of previous task data along with all new task data), Replay 10%, Replay 100%, DeepArc (i.e., training the new task only on the redundant layers of the model), global slicing, category-wise slicing, NeuSemSlice, and NeuSemSlice combined with Replay.

Results and Findings. The results of Experiment Scenario 1, as shown in Table 7, indicate that direct retraining causes catastrophic forgetting in all settings, with the model completely forgetting knowledge related to task A in six settings. Replay methods outperform direct retraining, with 5% data replay resulting in an average 30.45% decrease in predictive accuracy for task A, 10% data replay leading to a 23.03% average decrease, and 100% data replay causing a 5.15% average decrease. Additionally, global slicing and category-wise slicing also achieve promising results, with global slicing reducing task A's accuracy by 11.75% and category-wise slicing by 10.58%, both outperforming 5% and 10% data replay.

Our method outperforms 5% and 10% data replay, as well as global slicing and category-wise slicing, and achieves comparable performance to 100% data replay in most scenarios, with an average precision loss of only 2.79% on task A. When combined with 10% data replay, our method consistently achieves the best results in all experimental settings, with an average precision loss for task A of only 0.25% and an average accuracy of 90.76%.

These results thoroughly demonstrate that our method effectively mitigates the issue of catastrophic forgetting in task A. Even with a fixed portion of the model's parameters, our method incurs only around a 1.64% loss in accuracy on task B compared to direct retraining. Notably, in terms of storage efficiency, replay methods require storing some training data from task A, typically needing several to tens of megabytes of storage space under our experimental settings. In contrast, our method only needs to retain index information of critical neurons in the model, significantly reducing storage requirements to just a few kilobytes, which is particularly crucial for applications with limited storage resources. Moreover, in terms of efficiency, our method is slightly superior to the 5% and 10% data replay and is twice as efficient as the 100% data replay, with more details provided in [44]. Overall, considering the combined performance on tasks A and B, our method demonstrates significant effectiveness in the model incremental development task.

Regarding DeepArc, we find that training on the identified redundant layers of the model does not effectively alleviate catastrophic forgetting. It maintains only partial capability to complete task A on VGG16 and ResNet101, and DeepArc leads to a substantial reduction in the model's ability to learn new tasks.

The results for the second scenario appear in Table 8. We further compare NeuSemSlice with naive retraining, replay-based methods, global slicing, and category-wise slicing [5]. In six experimental settings, direct retraining once again forgets knowledge related to task A and task B. Replay-based methods also decline compared to the first scenario. For example, replay that uses 5% of the data yields average accuracies of 91.93%, 69.99%, 53.82%, 54.68%, 41.43%, and 50.46% across the six settings, while replay that uses 10% of the data produces similar results and performs well only on MLP-MNIST. In contrast, NeuSemSlice achieves the highest performance, with an overall average accuracy of 89.44% in the six settings, which is 4.36% higher than global slicing and 2.75% higher than category-wise slicing. However, the introduction of more complex tasks leads to imbalanced performance across tasks, with forgetting in task B being the most severe in most settings. A possible explanation is that task

[5]Due to the significant decline in DeepArc's ability to learn new tasks observed in Experiment Scenario I, it is unable to proceed with training task C. Therefore, we do not include it in the comparison here.

A, as the base model, is deeply embedded in the model's parameters, while task C receives focused training as the newest target, leading to stronger performance on these two tasks.

To further validate the effectiveness of NeuSemSlice, we perform paired t-tests across all experimental settings to compare its performance with baseline methods, including replay-based methods, global slicing, and category-wise slicing. The results show that NeuSemSlice's improvements are statistically significant (p-values < 0.05, Cohen's d > 0.8) in all comparisons except for replay 10% (p-values = 0.06, Cohen's d = 0.98) and replay 100% (p-values = 0.33, Cohen's d = 0.44) in Experiment Scenario 1. These statistical findings confirm the robustness and generalizability of NeuSemSlice in incremental development scenarios, providing strong evidence of its practical utility in mitigating catastrophic forgetting while maintaining computational and storage efficiency.

Answer to RQ4: In Experiment Scenario 1, our method reduces the average accuracy loss for the original task to 2.79% and achieves 89.77% average accuracy in the incremental model development task. Combined with replay, it consistently excels across all settings, reducing the average accuracy loss for the original task to 0.25% and increasing average accuracy to 90.76%, effectively countering catastrophic forgetting. Similar trends are also observed in Experiment Scenario 2.

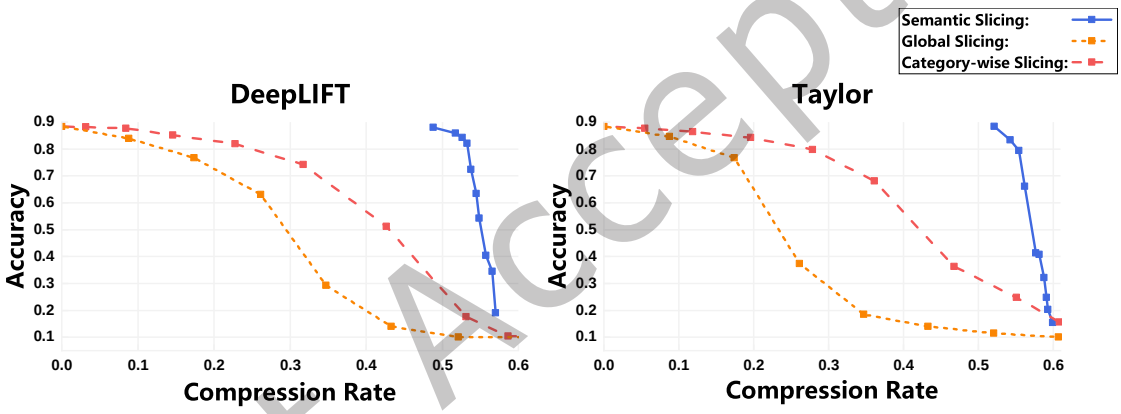

Fig. 6. Comparison Results of Performance for Semantic Slicing, Global Slicing, and Category-Wise Slicing on MobileNet (Left: Based on DeepLIFT, Right: Based on Taylor). Ideally, Dots (Lines) Located at the **Upper Right** Are Favored for Their Higher Compression Rates and Better Accuracy Performance, Which Also Suggest That the Corresponding Slicing Methods Better Identify Critical Neurons in the Model.

5.6 Applications in Software Engineering: A Case Study on MobileNet

To further demonstrate the applicability of our approach in software engineering, particularly in resource-constrained or embedded deployment scenarios, we extend our evaluation to the widely adopted MobileNet architecture [15]. MobileNet is extensively utilized in real-world environments, ranging from embedded systems to intelligent IoT devices, making it a strong representative for mission-critical deployments. Specifically, we continue using CIFAR-10 as the target dataset and adopt the latest MobileNet V3 as the target model, conducting a case study under the same experimental settings. Through this evaluation, we aim to assess whether NeuSemSlice can consistently facilitate effective model maintenance in architectures that closely align with real-world software systems.

Experimental Results. Figure 6 presents the effectiveness comparison of semantic slicing, global slicing, and category-wise slicing on MobileNet using DeepLIFT and Taylor. The results clearly indicate that, at comparable compression rates (i.e., when identifying a similar proportion of critical neurons), semantic slicing more accurately identifies critical neurons in the model due to its semantic-based approach. This leads to improved model accuracy under equivalent compression rates. Specifically, for Taylor, when the model compression rate is approximately 50%, semantic slicing achieves an accuracy of 88.41%, whereas global slicing and category-wise slicing yield accuracies of 13.90% and 36.25%, respectively. DeepLIFT exhibits a similar trend. Additionally, global slicing and category-wise slicing maintain reasonable accuracy only when the compression rate is reduced to 20% or lower.

Furthermore, the three slicing methods identified using Taylor are applied in Model Restructure and integrated with existing pruning techniques to evaluate their impact on model compression efficiency. Experimental results demonstrate that for Global and ERK, Model Restructure with semantic slicing enables the model to achieve a compression rate of approximately 60% within only two pruning iterations, while maintaining an accuracy of 88.41%. In contrast, directly applying Global or ERK requires approximately nine iterations to reach a similar compression rate and model accuracy. For the Uniform+ method, without semantic slicing, the model undergoes nine pruning iterations to achieve a compression rate of 61.26%, at which point accuracy drops to 19.34%. However, when assisted by semantic slicing in Model Restructure, only two pruning iterations are needed to reach a compression rate of 61.71%, while accuracy remains at 87.95%. In contrast, global slicing and category-wise slicing exhibit inaccuracies in identifying critical neurons, leading to substantial accuracy degradation during pruning (falling below 30%), ultimately compromising model usability.

Conclusion and Outlook. Our case study on MobileNet confirms that the proposed approach can be effectively adapted to resource-constrained or embedded software systems, enabling streamlined model maintenance without compromising performance. These promising results suggest broader applicability in other software engineering scenarios. In future work, we plan to extend our methodology to more diverse real-world applications to further demonstrate its utility.

5.7 Threats to Validity

5.7.1 External Threats. The threat to external validity arises from the particular settings we have chosen for our study, which raises concerns about the generalizability of our proposed approach. To mitigate these threats, we have selected diverse evaluation settings. In more detail, we select four widely utilized datasets-MNIST, Fashion-MNIST, CIFAR10, and ImageNet, and employ six classical DNN architectures-MLP, AlexNet, MobileNet,VGG16, ResNet101, and VGG19. The experiments are conducted under seven settings. This selection is intended to enhance the generalizability of our findings across common scenarios within the domain.

5.7.2 Internal Threats. Internal validity threats stem from the tools utilized in our research, which include captum [21] and cleverhans [46]. Additionally, there is a potential threat concerning the accurate reproduction of the contribution assessment metrics and the model modularization framework DeepArc in PyTorch [49]. The use of various deep learning frameworks, such as TensorFlow [1], may impact the results. Additionally, the inherent randomness in model training poses a potential threat to the internal validity. We addressed this issue by repeating key experiments more than five times and reporting the mean values. Finally, as mentioned in Section 5.4, the resilience of the model's re-adaptation to adversarial attacks requires enhancement. This issue, though orthogonal to the main focus of this paper, highlights the need for future work to tackle and reduce its impact.

6 CONCLUSION

This paper introduces NeuSemSlice, an innovative framework that integrates semantic slicing with semantic-aware model maintenance for deep neural networks. Semantic slicing identifies critical neurons at a granular

level, categorizing and merging them across layers based on semantic similarities. This approach markedly enhances the effectiveness of these neurons in model maintenance tasks, surpassing global slicing and category-wise slicing. NeuSemSlice further innovates in semantic-aware model maintenance by offering strategies for model restructure that preserve critical neurons, efficient re-adaptation through tuning only these neurons, and incremental development that maintains critical neurons while training non-critical ones. The effectiveness of NeuSemSlice is demonstrated through comprehensive evaluations, showing its superiority over baselines in all three aspects of model maintenance.

ACKNOWLEDGMENTS

We thank the editor and anonymous reviewers for their constructive comments. This work was supported by the National NSF of China (grants No.62302176, No.62072046, 62302181), the Key R&D Program of Hubei Province (2023BAB017, 2023BAB079), the Knowledge Innovation Program of Wuhan-Basic Research (2022010801010083)

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] AI & ML Model Maintenance. 2023. https://f8federal.com/ai-ml-model-maintenance.

[3] AI model maintenance: ensuring accuracy, scalability, and reliability. 2023. https://aiupbeat.com/ai-model-maintenance-ensuring-accuracy-scalability-and-reliability.

[4] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. 2019. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson (Eds.). https://doi.org/10.15607/RSS.2019.XV.031

[5] Árpád Beszédes, Tamás Gergely, Z Mihaly Szabo, János Csirik, and Tibor Gyimothy. 2001. Dynamic slicing method for maintenance of large C programs. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering*. IEEE, 105–113.

[6] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical Accuracy Estimation for Efficient Deep Neural Network Testing. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 30 (Oct. 2020), 35 pages. https://doi.org/10.1145/3394112

[7] Pin-Yu Chen and Payel Das. 2023. AI Maintenance: A Robustness Perspective. *Computer* 56, 2 (2023), 48–56. https://doi.org/10.1109/MC.2022.3218005

[8] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE international conference on computer vision*. 2857–2865.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[10] Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for interpretable machine learning. *Commun. ACM* 63, 1 (2019), 68–77.

[11] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the Lottery: Making All Tickets Winners. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 2943–2952. https://proceedings.mlr.press/v119/evci20a.html

[12] Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The State of Sparsity in Deep Neural Networks. *CoRR* abs/1902.09574 (2019). arXiv:1902.09574 http://arxiv.org/abs/1902.09574

[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2016.90

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[16] Dong Huang, Qingwen Bu, Yichao Fu, Yuhao Qing, Xiaofei Xie, Junjie Chen, and Heming Cui. 2024. Neuron Sensitivity-Guided Test Case Selection. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 188 (Sept. 2024), 32 pages. https://doi.org/10.1145/3672454

[17] Jonathan Ish-Horowicz, Dana Udwin, Seth R. Flaxman, Sarah Filippi, and Lorin Crawford. 2019. Interpreting Deep Neural Networks Through Variable Importance. *CoRR* abs/1901.09839 (2019). arXiv:1901.09839 http://arxiv.org/abs/1901.09839

[18] Jiajun Jiang, Junjie Yang, Yingyi Zhang, Zan Wang, Hanmo You, and Junjie Chen. 2024. A Post-training Framework for Improving the Performance of Deep Learning Models via Model Transformation. *ACM Trans. Softw. Eng. Methodol.* 33, 3, Article 61 (March 2024), 41 pages. https://doi.org/10.1145/3630011

[19] Siyuan Jiang, Raul Santelices, Mark Grechanik, and Haipeng Cai. 2014. On the accuracy of forward dynamic slicing and its effects on software maintenance. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 145–154.

[20] Sungmin Kang, Robert Feldt, and Shin Yoo. 2024. Deceiving Humans and Machines Alike: Search-based Test Input Generation for DNNs Using Variational Autoencoders. *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 103 (April 2024), 24 pages. https://doi.org/10.1145/3635706

[21] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for PyTorch. *CoRR* abs/2009.07896 (2020). arXiv:2009.07896 https://arxiv.org/abs/2009.07896

[22] Dimitrios Kollias, Athanasios Tagaris, Andreas Stafylopatis, Stefanos Kollias, and Georgios Tagaris. 2018. Deep neural architectures for prediction in healthcare. *Complex & Intelligent Systems* 4 (2018), 119–131.

[23] Raghavan Komondoor and Susan Horwitz. 2001. Using Slicing to Identify Duplication in Source Code. In *Static Analysis*, Patrick Cousot (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 40–56.

[24] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *International conference on machine learning*. PMLR, 3519–3529.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[27] Jasmine Latendresse, Samuel Abedu, Ahmad Abdellatif, and Emad Shihab. 2024. An Exploratory Study on Machine Learning Model Management. *ACM Trans. Softw. Eng. Methodol.* (Aug. 2024). https://doi.org/10.1145/3688841 Just Accepted.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[29] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. 2021. Layer-adaptive Sparsity for the Magnitude-based Pruning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H6ATjJ0TKdf

[30] Ding Li, Ziqi Zhang, Mengyu Yao, Yifeng Cai, Yao Guo, and Xiangqun Chen. 2024. TEESlice: Protecting Sensitive Neural Network Models in Trusted Execution Environments When Attackers have Pre-Trained Models. *ACM Trans. Softw. Eng. Methodol.* (Dec. 2024). https://doi.org/10.1145/3707453 Just Accepted.

[31] Ningke Li, Shenao Wang, Mingxi Feng, Kailong Wang, Meizhen Wang, and Haoyu Wang. 2024. MalWuKong: Towards Fast, Accurate, and Multilingual Detection of Malicious Code Poisoning in OSS Supply Chains. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering* (Echternach, Luxembourg) *(ASE '23)*. IEEE Press, 1993–2005. https://doi.org/10.1109/ASE56229.2023.00073

[32] Tianlin Li, Yue Cao, Jian Zhang, Shiqian Zhao, Yihao Huang, Aishan Liu, Qing Guo, and Yang Liu. 2024. RUNNER: Responsible UNfair NEuron Repair for Enhancing Deep Neural Network Fairness. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 9, 13 pages. https://doi.org/10.1145/3597503.3623334

[33] Tianlin Li, Qing Guo, Aishan Liu, Mengnan Du, Zhiming Li, and Yang Liu. 2023. FAIRER: fairness as decision rationale alignment. In *International Conference on Machine Learning*. PMLR, 19471–19489.

[34] Tianlin Li, Aishan Liu, Xianglong Liu, Yitao Xu, Chongzhi Zhang, and Xiaofei Xie. 2021. Understanding adversarial robustness via critical attacking route. *Information Sciences* 547 (2021), 568–578.

[35] Tianlin Li, Xiaofei Xie, Jian Wang, Qing Guo, Aishan Liu, Lei Ma, and Yang Liu. 2023. Faire: Repairing Fairness of Neural Networks via Neuron Condition Synthesis. *ACM Trans. Softw. Eng. Methodol.* (aug 2023). https://doi.org/10.1145/3617168 Just Accepted.

[36] Yi Li. 2017. Managing software evolution through semantic history slicing. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1014–1017. https://doi.org/10.1109/ASE.2017.8115722

[37] Yanzhou Li, Tianlin Li, Kangjie Chen, Jian Zhang, Shangqing Liu, Wenhan Wang, Tianwei Zhang, and Yang Liu. 2024. Badedit: Backdooring large language models by model editing. *arXiv preprint arXiv:2403.13355* (2024).

[38] Jiaxiang Liu, Yunhan Xing, Xiaomu Shi, Fu Song, Zhiwu Xu, and Zhong Ming. 2024. Abstraction and Refinement: Towards Scalable and Exact Verification of Neural Networks. *ACM Trans. Softw. Eng. Methodol.* 33, 5, Article 129 (June 2024), 35 pages. https://doi.org/10.1145/3644387

[39] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International symposium on research in attacks, intrusions, and defenses*. Springer, 273–294.

[40] Yongxin Liu, Jian Wang, Jianqiang Li, Shuteng Niu, and Houbing Song. 2021. Class-Incremental Learning for Wireless Device Identification in IoT. *IEEE Internet of Things Journal* 8, 23 (2021), 17227–17235. https://doi.org/10.1109/JIOT.2021.3078407

[41] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb

[42] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance Estimation for Neural Network Pruning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11256–11264. https://doi.org/10.1109/CVPR.2019.01152

[43] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. 2019. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/a4613e8d72a61b3b69b32d040f89ad81-Paper.pdf

[44] NeuSemSlice. 2025. NeuSemSlice: Towards Effective DNN Model Maintenance via Neuron-level Semantic Slicing. https://sites.google.com/view/neusemslice Accessed: 2025-03-21.

[45] Thao Nguyen, Maithra Raghu, and Simon Kornblith. 2021. Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth. In *International Conference on Learning Representations*. https://openreview.net/forum?id=KJNcAkY8tY4

[46] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768* (2018).

[47] Aurora Papotti, Fabio Massacci, and Katja Tuma. 2024. On the Effects of Program Slicing for Vulnerability Detection during Code Inspection: Extended Abstract. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24)*. Association for Computing Machinery, New York, NY, USA, 368–369. https://doi.org/10.1145/3639478.3643117

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[50] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[51] Xiaoning Ren, Yun Lin, Yinxing Xue, Ruofan Liu, Jun Sun, Zhiyong Feng, and Jin Song Dong. 2023. DeepArc: Modularizing Neural Networks for the Model Maintenance. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1008–1019.

[52] Francisco Servant and James A. Jones. 2012. History slicing: assisting code-evolution tasks. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (Cary, North Carolina) *(FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 43, 11 pages. https://doi.org/10.1145/2393596.2393646

[53] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International conference on machine learning*. PMLR, 3145–3153.

[54] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

[55] Mifta Sintaha, Noor Nashid, and Ali Mesbah. 2023. Katana: Dual Slicing Based Context for Learning Bug Fixes. *ACM Trans. Softw. Eng. Methodol.* 32, 4, Article 100 (May 2023), 27 pages. https://doi.org/10.1145/3579640

[56] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*. 338–349.

[57] Thibaut Théate and Damien Ernst. 2021. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications* 173 (2021), 114632.

[58] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*. IEEE, 1–5.

[59] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[60] Zhiyu Wang, Sihan Xu, Lingling Fan, Xiangrui Cai, Linyu Li, and Zheli Liu. 2024. Can Coverage Criteria Guide Failure Discovery for Image Classifiers? An Empirical Study. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 190 (Sept. 2024), 28 pages. https://doi.org/10.1145/3672446

[61] Zhengyuan Wei, Haipeng Wang, Imran Ashraf, and Wing-Kwong Chan. 2023. DeepPatch: Maintaining Deep Learning Model Programs to Retain Standard Accuracy with Substantial Robustness Improvement. *ACM Trans. Softw. Eng. Methodol.* 32, 6, Article 150 (Sept. 2023), 49 pages. https://doi.org/10.1145/3604609

[62] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017). arXiv:1708.07747 http://arxiv.org/abs/1708.07747

[63] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. 2022. Npc: N euron p ath c overage via characterizing decision logic of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27.

[64] Chongzhi Zhang, Aishan Liu, Xianglong Liu, Yitao Xu, Hang Yu, Yuqing Ma, and Tianlin Li. 2020. Interpreting and improving adversarial robustness of deep neural networks with neuron sensitivity. *IEEE Transactions on Image Processing* 30 (2020), 1291–1304.

[65] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic Slicing for Deep Neural Networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 838–850. https://doi.org/10.1145/3368089.3409676