ROBUSTNCL: ROBUSTIFYING NEUROMORPHIC CONTINUAL LEARNING UNDER TRANSIENT FAULTS WITH LIGHTWEIGHT DETECT-AND-CURE STRATEGY

Anonymous authors
Paper under double-blind review

ABSTRACT

In neural networks, continual learning (CL) is a prominent approach for enabling deployed systems to adapt to dynamic operational environments by effectively learning from new unseen data at runtime. Recently, the neuromorphic continual learning (NCL) concept has emerged as an energy-efficient alternative to CL by leveraging sparse event-based operations. However, NCL performance can be undermined by transient faults in the hardware platforms, which may come from high-energy particle strikes at any time. Currently, NCL reliability under transient faults has not been studied, and hence the corresponding fault-tolerance techniques remain unexplored. To address this, we propose a novel RobustNCL framework to improve NCL reliability in the presence of transient faults in the DRAM-based off-chip memory of hardware platforms through a lightweight detect-and-cure strategy. Specifically, it first monitors the presence of faults during weight data access from off-chip memory, and then cures the accessed weights if anomalous signatures are detected. It provides three curing variants: replacement with zero, min/max value, and most-probable value from the original/clean weight distribution. To support this, the weight curing values are stored in the radiation-hardened memory. Experimental results show that, in the presence of high fault rates (e.g., 5×10^{-4} for memory replay), RobustNCL maintains high accuracy close to the accuracy of fault-free models (i.e., within -0.5% accuracy for the best RobustNCL variant) and significantly improves accuracy from the baseline without fault mitigation across different NCL methods and datasets (i.e., Split MNIST, Split Fashion-MNIST, Split CIFAR-10, and Split CIFAR-100), while incurring negligible latency and energy overheads. These results highlight that our RobustNCL framework is a promising approach for enabling reliable and energy-efficient NCL systems.

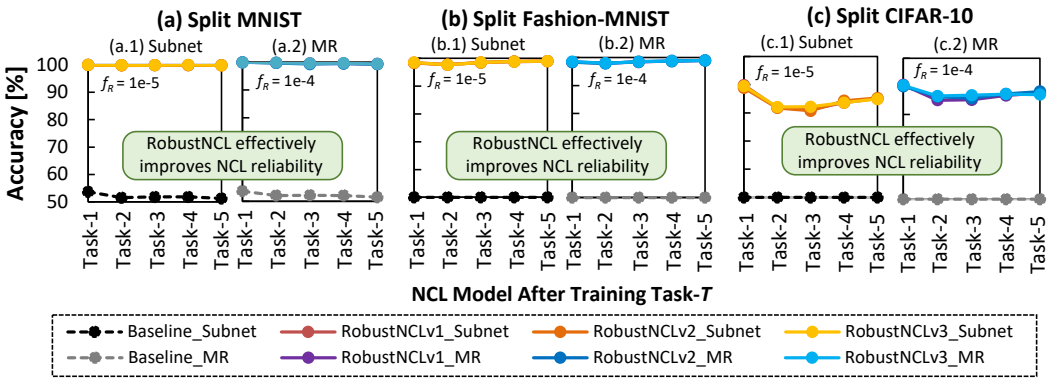

Figure 1: RobustNCL effectively improves NCL reliability againts transient faults in the DRAM off-chip/main memory under a task-incremental learning (TIL) scenario across different benchmarks: (a) Split MNIST, (b) Split Fashion-MNIST, and (c) Split CIFAR-10, across different NCL methods: task-specific subnetwork (Subnet) and memory replay (MR). The fault rate is denoted as f_R considering a DDR3-1600 2Gb DRAM chip (Micron, 2010), RobustNCL variants are with v1-v3 codes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Neural network (NN)-based systems typically rely on offline training to gain high performance (e.g., accuracy) during deployments (LeCun et al., 2015). However, the systems' knowledge may be outdated over time since they encounter new/unseen data at runtime, that were not learned during offline training and often represent a new task. This condition negatively affects system performance, such as significant accuracy degradation. Recently, emerging resource-constrained autonomous systems (e.g., mobile robots, UAVs, UGVs, etc.) increasingly demand that deployed systems can learn new tasks using new data gathered directly from dynamically changing environments under low computational and memory costs (Shaheen et al., 2022).

To address this, the continual learning (CL) concept has become a prominent approach. CL allows deployed NN-based systems to learn new tasks from data encountered at runtime without retraining on all past training data from previously-learned tasks, while minimizing catastrophic forgetting (CF) issues, where the NN models forget older tasks when learning a new task (De Lange et al., 2022; Wang et al., 2024; Minhas et al., 2025a). However, CL methods often demand high computational and memory costs, thereby posing non-trivial challenges for their deployments on resource-constrained autonomous systems. Recently, the neuromorphic continual learning (NCL) concept has been explored to offer an energy-efficient alternative to conventional CL methods by leveraging event-driven, sparse spiking operations from spiking neural network (SNN) computational models (Roy et al., 2019; Davies et al., 2018; Minhas et al., 2025a).

NCL accuracy can be undermined by *transient faults* in hardware platforms, especially in the DRAM off-chip/main memory that stores the trained weights; see Fig 1. Transient faults can occur at any time due to high-energy particle strikes, from either cosmic rays or packaging materials (Baumann, 2005; Putra et al., 2022). These faults manifest as bit-flips at the hardware layer, and these errors can propagate to the software layer (Putra et al., 2022). These errors persist until they are replaced with new data bits, hence also known as *soft errors*. Such errors change the weight values in the DRAM at any time, and may lead to significant NCL accuracy degradation. However, NCL reliability under transient faults has not been studied, and the corresponding fault-tolerance techniques also remain unexplored. *Therefore, **the targeted problem** in this paper is, how can we systematically study the negative impact of transient faults in the DRAM off-chip/main memory on NCL reliability, and develop an efficient fault-mitigation technique?* An efficient solution to this problem may enable reliable NCL processing with low memory, latency, and energy overheads, which is especially suitable for resource-constrained autonomous systems.

State-of-the-Art Fault Tolerance Methods: Standard fault-tolerance methods from integrated circuit (IC) design may be used to mitigate transient faults. They include a redundancy approach like Dual Modular Redundancy (DMR) (Vadlamani et al., 2010) and Triple Modular Redundancy (TMR) (Lyons & Vanderkulk, 1962), and an error correction approach like Error Correcting Codes (ECCs) (Sze, 2000). However, these techniques incur high overheads in terms of area, latency, and power/energy consumption. Additionally, fault-tolerance methods from offline-trained Deep Neural Networks (DNNs) Hoang et al. (2020) and Spiking Neural Networks (SNNs) (Putra et al., 2022) cannot be directly employed because they do not consider different states of the network model (i.e., updated models due to NCL process) that can be affected by transient faults. Furthermore, the state-of-the-art NCL works still focus on improving their performance for the targeted applications. *Therefore, NCL reliability against transient faults has not been studied, and consequently, the corresponding fault-mitigation techniques remain underexplored.*

Our Novel Contributions: To address the research problem, we propose ***RobustNCL**, a novel framework that aims at Robustifying NCL under transient faults in the DRAM off-chip/main memory with a lightweight detect-and-cure (D&C) strategy*; see Fig. 2. This study is also the first work to investigate NCL reliability in the presence of hardware faults and propose an efficient fault-mitigation solution. RobustNCL achieves this through the following novel contributions.

- **NCL Transient-Fault Modeling:** It introduces a fault model for injecting transient faults in the NCL scenario, which is important and required for systematic study and analysis. Its key idea is applying transient faults to the DRAM at different possible points in time.
- **NCL Fault-Tolerance Analysis:** It systematically studies NCL fault tolerance and its characteristics (e.g., weight distributions) through experiments that investigate NCL performance/accuracy under transient faults in the DRAM across different fault rates (f_R) and different NCL methods.

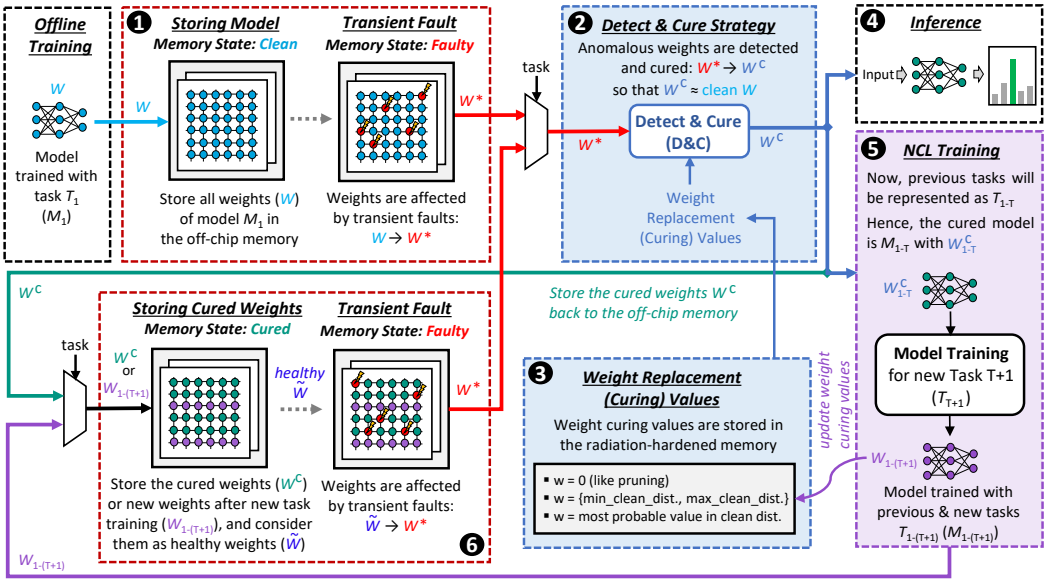

Figure 2: The RobustNCL framework, showing the transient fault exposure stage, our detect-and-cure (D&C) strategy stage supported by weight replacement (curing) value selection, and the NCL training stage. RobustNCL variants depend on how the faulty weights are replaced by selected curing weights.

- **Lightweight Detect-and-Cure (D&C) Strategy:** It continuously detects anomalous weights that are accessed from the DRAM based on their values, and replaces (cures) them with weight values that lead to safe spiking activities based on observations in the NCL training phase. It also provides appropriate curing weight variants, considering different possible model states due to NCL training.
- **Comprehensive Evaluation:** We evaluate the RobustNCL framework in diverse settings based on combinations of different fault rates f_R, different benchmarking datasets, and different NCL methods: task-specific subnetwork (Subnet) and memory replay (MR).

2 Related Work

Neuromorphic Continual Learning (NCL): NCL combines the event-driven computation of SNNs (Maass, 1997; Roy et al., 2019) with continual learning (Minhas et al., 2025a). In general, an NCL pipeline consists of three phases: (1) the offline training at design time, (2) the NCL training at runtime, and (3) the inference at runtime; see Fig. 3(a). At runtime, the deployed system will be in the NCL training phase when the network model learns a new task; otherwise, it is in the inference phase while employing the most updated model. NCL scenarios depend on the task identity. In this paper, we consider the task incremental learning (TIL) scenario, as it is widely used as the standard NCL evaluation in the SNN community. TIL has several characteristics: the task boundaries are explicitly defined, the task identity is known in both training and inference phases, and each task usually has multiple classes. In the TIL scenario, $\mathcal{T}_1, \ldots, \mathcal{T}_K$ arrive sequentially, and each task has its own classification head $\mathbf{h}_t$. Let $a_{t,j}$ denote the test accuracy on $\mathcal{T}_j$ after learning $\mathcal{T}_t$. We consider the final average accuracy as $\text{Acc} = \frac{1}{K}\sum_{j=1}^{K} a_{K,j}$ (Lopez-Paz & Ranzato, 2017).

NCL methods can be classified into several approaches, such as architecture methods (Han et al., 2023; Pes et al., 2024), memory replay methods (Dequino et al., 2024; Minhas et al., 2025b), and learning enhancement methods (Putra & Shafique, 2021; 2022; Skatchkovsky et al., 2022; Antonov et al., 2022; Xiao et al., 2024). In this paper, we consider two NCL methods from different approaches: *the task-specific subnetwork (Subnet) method* from the architecture-based approach, and *the memory replay (MR) method* to show the generality of our RobustNCL framework. In the Subnet methods, a specific subnetwork is trained for a specific task, allocating a pathway within the network to process that task; see Fig. 4(a). They typically achieve competitive accuracy with relatively small memory and computational overheads, but they may face limitations related to representation capacity.

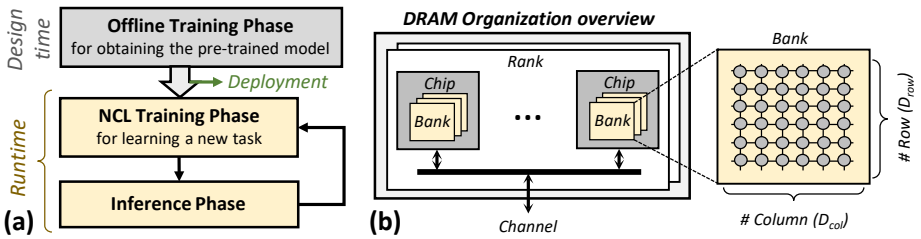

Figure 3: (a) The NCL pipeline consists of three phases: the offline training at design time, the NCL training at runtime, and the inference at runtime. (b) DRAM organization overview, including channels, ranks, chips, and banks.

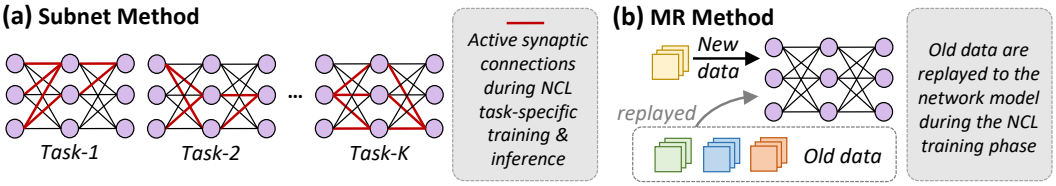

Figure 4: Overview of (a) the task-specific subnetwork *(Subnet)* method and (b) the memory replay *(MR)* method that are considered in this paper.

Meanwhile, MR methods replay previous inputs to the network model to reinforce old knowledge; see Fig. 4(b). Consequently, they often offer high accuracy but at the cost of a large replay buffer and extra forward/backward passes, indicating large memory and computational overheads (Proietti et al., 2023; Dequino et al., 2024; Minhas et al., 2025b).

DRAM Fundamentals: The top-down hierarchy of commodity DRAMs is organized into channels, ranks, chips, banks, rows, and columns, respectively (Ghose et al., 2019; Liu et al., 2012; Putra et al., 2020; 2021a); see Fig. 3(b). DRAM banks are the lowest-level units that can be accessed in parallel, known as bank-level parallelism Kim et al. (2010). In this paper, the number of DRAM channels-per-system, ranks-per-channel, chips-per-rank, banks-per-chip, rows-per-bank, and columns-per-bank are represented as D_{chn}, D_{rnk}, D_{chp}, D_{bnk}, D_{row}, and D_{col}, respectively. When a DRAM request is issued from the host CPU, a specific rank will respond, enabling parallel access to multiple chips in that rank whose data form a DRAM word. In each chip, the request is routed to a specific bank with a specific pair of row and column addresses. Then, the requested read or write can be performed.

3 The RobustNCL Framework

RobustNCL framework (Fig. 2) addresses the targeted problem through NCL transient-fault modeling, fault-tolerance analysis, detect-and-cure (D&C) strategy, and curing weight selection, as discussed in the following subsections.

3.1 NCL Transient-Fault Modeling

To properly investigate the impact of transient faults in the DRAM on NCL accuracy, *we develop a novel NCL transient-fault modeling*, as shown in Fig. 5. Its key idea is to consider fault generation and distribution for each DRAM chip, as each chip can be deployed independently on the system platform; see ①. Multiple banks in the chip are considered as a continuous memory space with the dimension of $D_{mem} = (D_{bnk} \cdot D_{row}) \times D_{col}$ memory cells, where each cell stores a single bit of data; see ②. Afterward, transient faults are randomly injected or distributed into the DRAM memory cells in the form of bit-flips based on the given fault rate f_R, as transient faults typically occur in random locations on a chip (Baumann, 2005; Putra et al., 2022); see ③. The fault rate f_R is defined as the ratio between the faulty bits and the total number of memory cells D_{mem}. Each faulty bit stays in the DRAM until it is overwritten by a new data bit; see ④. Furthermore, since transient fault injection can happen at any time, this fault model considers injecting transient faults into the

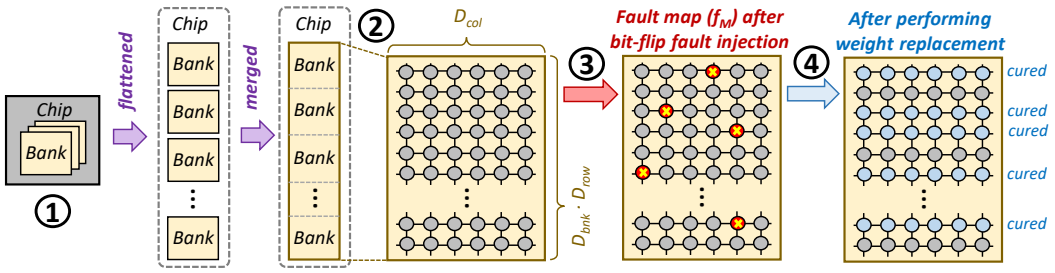

Figure 5: Transient-fault modeling in the DRAM with several key steps: flattening multiple banks into a continuous memory space, injecting transient faults, and curing anomalous weights. Transient fault injection can happen at any time, as illustrated in Fig. 2.

DRAM in different possible states of trained models. Consequently, the faults can affect the initial, offline-trained model (stage ❶ in Fig. 2) or the latest updated NCL-trained model (stage ❻ in Fig. 2).

3.2 NCL Fault-Tolerance Analysis

This step aims to understand NCL fault tolerance and its characteristics through experimental case studies that investigate NCL accuracy under different fault rates f_R of transient faults in the DRAM, while considering the following aspects.

- State-of-the-art NCL models often use floating-point 32-bit (FP32) precision for their weights; thus, we also use the same FP32 precision format and set the flattened $D_{col} = 32$.
- We consider a single DDR3-1600 2Gb DRAM chip to store the entire weights of the investigated NCL model, as this chip is widely used in many AI compute systems (Putra et al., 2021a;b). Since we set the flattened $D_{col} = 32$, then the flattened $D_{bnk} \cdot D_{row} = 2\text{Gb}/D_{col} = 2\text{Gb}/32$.
- Transient fault injection is performed after initial model deployment (stage ❶ in Fig. 2) and after every NCL training phase for each new task (stage ❻ in Fig. 2), while considering the fault modeling in Sec. 3.1.
- We apply the Subnet and MR methods separately to the spiking-based VGG-9 architecture (Apolinario et al., 2025) to provide comprehensive observations and insights across different NCL methods, while considering the Split MNIST benchmarking dataset.

Details of the experimental setup are further discussed in Sec. 4. Some of the results from the experimental case studies are presented in Fig. 6, from which we make the following key observations.

- In general, bit-flips may change the weight distribution of each layer (or even each subnetwork) since they can notably change the weight values, depending on the faulty bit position within the selected precision format; see ⑤.
- A bit-flip at the sign bit can switch the polarity of the weight value, hence affecting neural dynamics and output behavior. Specifically, a higher absolute value will lead to a more significant impact on neural dynamics and outputs.
- Some bit-flips may shift weights toward values that are very close to each other, so that they can be clustered in the same bin and notably increase the weight count; see ⑥.
- Bit-flips at significant bits of exponents may explode the weight values (i.e., either toward extreme positive or negative values), hence significantly deteriorate the neural dynamics; see ⑦.
- In the Subnet method, we observe a unique behavior. Transient faults in the DRAM may affect both the trained weights (from trained subnetworks) and the untrained weights (from untrained subnetworks). Here, we observe that bit errors in untrained weights may lead to bad weight initialization, which in turn makes the corresponding subnetworks difficult to train. Therefore, we should mitigate this issue as well.
- In this step, we also record several important profiles of the weight distribution from each layer of the clean, fault-free model, including the minimum value (w_{min}), the maximum value (w_{max}), and the most-frequent/most-probable value (w_{pro}).

All these key observations are then leveraged to develop fault mitigation techniques in Sec. 3.3.

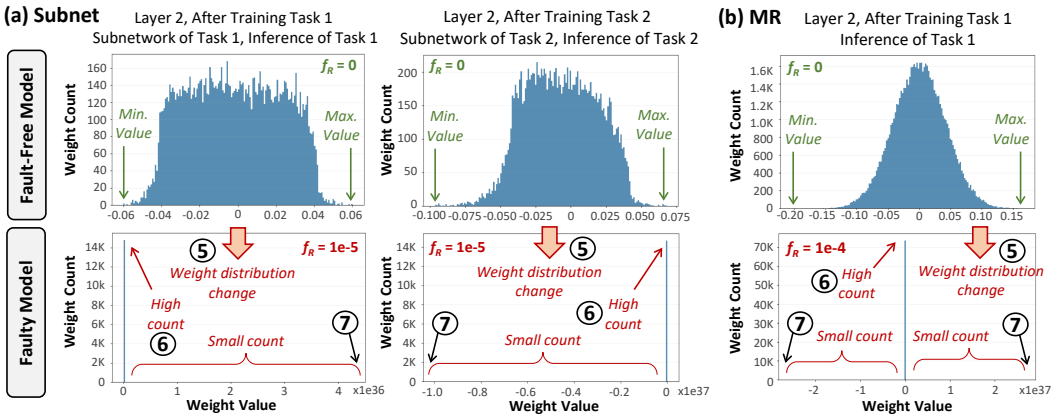

Figure 6: Experimental results from different NCL methods: (a) Subnet and (b) MR. These results show weight distributions of both the fault-free and faulty models from their layer 2 at certain model states (i.e., after training task 1 and task 2), inference tasks (i.e., task 1 and task 2), and fault rates ($f_R = 1{\times}10^{-5}$ for Subnet and $f_R = 1{\times}10^{-4}$ for MR).

3.3 Lightweight Detect-and-Cure (D&C) Strategy

To detect and mitigate transient faults at runtime, our RobustNCL framework employs the Detect-and-Cure (D&C) technique, which actively monitors whether anomalous weight values (W^*) are accessed from the DRAM, and proactively replaces (cures) them with safe values, thereby bounding the cured weight values (W^C) within a safe range that does not cause erroneous neural behavior.

For a clear explanation, we discuss the proposed D&C technique while describing the full stages of our RobustNCL framework below, based on Fig. 2.

Stage-❶ Initial Deployed Model: This stage represents the initial deployment of the offline-trained model (M_1). Its clean, trained weights (W) are fully stored in the DRAM. If transient faults appear, then the clean weights W will be corrupted into faulty weights (W^*). Here, the transient-fault modeling follows the discussion in Sec. 3.1. To address these faulty weights W^*, our D&C strategy in stage-❷ is employed.

Stage-❷ D&C Processing Strategy: This stage aims to cure faulty weights W^* accessed from the DRAM. Specifically, it performs our D&C technique on each weight accessed from the DRAM ($\bar{w}$). It first detects if $\bar{w}$ is outside the clean weight distribution (i.e., $\bar{w} > w_{max}$ or $\bar{w} < w_{min}$). If so, then it is considered a faulty weight ($\bar{w} \rightarrow w^*$) and will be replaced with the selected curing weight (w^c). As a result, all weights of the model are cured (W^C) and safe to use for both the inference and NCL training phases.

Stage-❸ Curing Weight Library: This stage provides weight candidates for each network layer to replace faulty weights, based on statistics (w_{min}, w_{max}, and w_{pro}) observed during the training phase. Here, there are two conditions.

- In the initial deployment, this library only records statistics from the offline-trained model. Consequently, the selected curing value w^c follows the initial statistics that are recorded in the offline fault-tolerance analysis in Sec. 3.2.
- As the model gets updated through NCL training at runtime, this library also updates its records based on updated statistics. Consequently, the selected curing value w^c follows the updated statistics in the NCL training phase in stage-❺.

***Weight Replacement Variants*:** To provide appropriate curing weight values, we propose different solution variants based on the recorded statistics of the per-layer weight distribution, including the minimum value w_{min}, the minimum value w_{max}, and the most-probable value w_{pro}. For the Subnet method in particular, we record statistics for each subnetwork per layer, as different subnetworks may have different weight distributions. The proposed weight replacement variants include the following.

- **RobustNCLv1 (Zero):** If the DRAM-accessed weight $\bar{w}$ is anomalous, then it is considered faulty ($\bar{w} \rightarrow w^*$). Then, we replace this faulty weight with zero, leading to $w^* \leftarrow w^c$ with $w^c = 0$.
- **RobustNCLv2 (MinMax):** If the DRAM-accessed weight $\bar{w}$ is anomalous, then it is considered faulty ($\bar{w} \rightarrow w^*$). Then, we clip this faulty weight based on its value relative to the corresponding weight distribution. If $w^* > w_{max}$, then we replace w^* with the maximum value from the distribution, leading to $w^* \leftarrow w^c$ with $w^c = w_{max}$. Meanwhile, if $w^* < w_{min}$, then we replace w^* with the minimum value from the distribution, leading to $w^* \leftarrow w^c$ with $w^c = w_{min}$.
- **RobustNCLv3 (Pro):** If the DRAM-accessed weight $\bar{w}$ is anomalous, then it is considered faulty ($\bar{w} \rightarrow w^*$). Then, we replace this faulty weight with the most-probable value, leading to $w^* \leftarrow w^c$ with $w^c = w_{pro}$.

Since these curing weight values can also be affected by transient faults, we consider storing them in a radiation-hardened memory to ensure resistance against high-energy particle strikes. To achieve this, the existing hardening techniques that enhance the fabrication process (Garg et al., 2009; Huang & Jiang, 2019) can be employed.

Stage-❹ Inference Phase: It performs inference using the cured weights W^C from the D&C stage.

Stage-❺ NCL Training Phase: This stage performs the NCL training to update the model with a new task (T+1). Specifically, the previously trained weights (W^C_{1-T}) are updated by training on a new task (T+1), becoming the latest updated weights ($W^C_{1-(T+1)}$). The recorded statistics from this phase are stored in the curing weight library, so they can be used to guide the D&C processing stage.

Stage-❻ NCL-Trained Model: This stage stores the updated weights ($W_{1-(T+1)}$) in the DRAM, called as healthy weights ($\tilde{W}$). If transient faults appear, then these healthy weights $\tilde{W}$ will be corrupted into faulty weights (W^*). Again, the transient-fault modeling follows the discussion in Sec. 3.1, and their faulty weights will be addressed by our D&C strategy in stage-❷.

4 Experimental Setup

To evaluate our RobustNCL framework, we implement it in Python and run it on a single Nvidia RTX 4090 GPU device. It considers the TIL scenario, each benchmarking dataset is divided into a sequence of tasks: task 1, task 2, ..., task K; where K denotes the maximum number of accommodated tasks. Task 1 is learned in the offline training phase to obtain the initial model to deploy for the first time; while the remaining tasks are learned sequentially in order during the NCL training phase.

NCL Methods and Benchmarking Datasets: To show the generality of our RobustNCL framework, we consider: (1) the task-specific subnetwork (Subnet) method to represent the architecture-based approach, and (2) the memory replay (MR) method (Minhas et al., 2025a). The evaluation considers diverse benchmarking datasets, encompassing Split MNIST, Split Fashion MNIST (FMNIST), Split CIFAR-10, and Split CIFAR-100. Each 5-task dataset (i.e., Split MNIST, Split FMNIST, and Split CIFAR-10) has 5 tasks, each having 2 classes-per-task. Meanwhile, the 10-task dataset (i.e., Split CIFAR-100) has 10 tasks with 10 classes-per-task. The evaluation outputs include accuracy, as well as processing time, memory, and power/energy consumption obtained from Nvidia NVML utilities.

Network, Hardware, and Fault Rate Settings: We use the same network architecture for all experiments: the VGG9 network with 8 convolutional and 1 fully-connected layers (Apolinario et al., 2025). To support the TIL scenario, the last layer is structured so that each task is associated with a specific FC head. For the 5-task datasets (Split MNIST, Split Fashion MNIST, and Split CIFAR-10), each task training considers 5 timesteps, 8 epochs-per-task, and a learning rate of $\eta = 0.005$. For the 10-task dataset (Split CIFAR-100), each task training considers 9 timesteps, 100 epochs-per-task, and a learning rate of $\eta = 0.005$. Both Subnet and MR use a batch size of 32. Additionally, MR uses a buffer of 2,000 samples and a replay batch of 48. Furthermore, we consider a single DDR3-1600 2Gb DRAM chip (Micron, 2010) and different fault rates, as mentioned below.

- For Subnet on 5-task datasets: $f_R \in \{5{\times}10^{-9}, 1{\times}10^{-8}, 5{\times}10^{-8}, 1{\times}10^{-7}, 5{\times}10^{-7}, 1{\times}10^{-6}, 5{\times}10^{-6}, 1{\times}10^{-5}\}$.
- For MR on 5-task datasets: $f_R \in \{1{\times}10^{-7}, 5{\times}10^{-7}, 1{\times}10^{-6}, 5{\times}10^{-6}, 1{\times}10^{-5}, 5{\times}10^{-5}, 1{\times}10^{-4}, 5{\times}10^{-4}\}$.
- For both Subnet and MR on Split CIFAR-100: $f_R \in \{1{\times}10^{-8}, 1{\times}10^{-7}, 1{\times}10^{-6}, 1{\times}10^{-5}\}$.

5 Results and Discussion

5.1 Maintaining High Accuracy of NCL under Transient Faults

Table 1, Table 2, and Table 3 present the experimental results on the final average accuracy (%) evaluated across all trained tasks and across different fault-mitigation techniques for: Subnet with 5-task datasets, MR with 5-task datasets, and both Subnet and MR with 10-task datasets, respectively.

In Table 1 (Subnet & 5-task datasets), the baseline suffers from more than -10% accuracy drop when encountering fault rates of $f_R \geq 1{\times}10^{-7}$ for Split MNIST and Split CIFAR-10, and $f_R \geq 5{\times}10^{-8}$ for Split FMNIST. In Table 2 (MR & 5-task datasets), the baseline again suffers from more than -10% accuracy drop when encountering fault rates of $f_R \geq 5{\times}10^{-5}$ for Split MNIST and Split CIFAR-10, and generally $f_R \geq 1{\times}10^{-5}$ for Split FMNIST. Similarly, in Table 3 (10-task datasets), the baseline suffers from more than -10% accuracy drop when encountering fault rates of $f_R \geq 1{\times}10^{-7}$ for Subnet, and $f_R \geq 1{\times}10^{-6}$ for MR. The reason is that, the baseline weights are very sensitive to bit-flips, especially at their significant bit locations, such as the sign and exponent bits, thereby leading to significant accuracy degradation if these bits are flipped, even at a low f_R.

In contrast, all of our RobustNCL variants successfully maintain NCL high accuracy of across different NCL methods, datasets, and fault rates. In Table 1 (Subnet & 5-task datasets), the best RobustNCL variant maintains no accuracy loss from the fault-free baseline, as they achieve final average accuracy between 99.79% – 99.82% (Split MNIST), 99.01% – 99.04% (Split FMNIST), and 84.88% – 85.25% (Split CIFAR-10) even at high fault rate $f_R = 1{\times}10^{-5}$, with RobustNCLv2 (MinMax) always achieves the highest accuracy. In Table 2 (MR & 5-task datasets), the best RobustNCL variant maintains accuracy within -0.5% from the fault-free baseline, as they achieve final average accuracy between 99.34% – 99.51% (Split MNIST), 98.31% – 98.92% (Split FMNIST), and 86.31% – 88.31% (Split CIFAR-10) even at high fault rate $f_R = 5{\times}10^{-4}$. Similarly, in Table 3 (10-task datasets), the best RobustNCL variant maintains no loss in accuracy from the fault-free baseline, as they achieve final average accuracy between 70.59% – 71.02% (Subnet), 65.90% – 66.82% (MR) at high fault rate $f_R = 1{\times}10^{-5}$. All these improvements obtained by our RobustNCL variants are due to the D&C strategy, which always monitors each DRAM-accessed weight to detect if it has an anomalous signature (i.e., too-small or too-big value) relative to the known clean/cured weight distributions for the corresponding layer or subnetwork-per-layer, and if so, it proactively cures this faulty weight accordingly. Moreover, our updated curing weight library ensures the replacement/curing process remains relevant and updated with the latest NCL-trained model, thereby adapting to the dynamic nature of NCL model. Furthermore, we also observe that RobustNCLv2 (MinMax) generally performs slightly better than the other variants for 5-task datasets, while RobustNCLv3 (Pro) generally performs slightly better than the other variants for the 10-task dataset, at high fault rate (e.g., $f_R = 1{\times}10^{-5}$) and across different NCL methods. These indicate that those variants provide a better representation of the weight distributions for the given workload/dataset.

Table 1: Final average accuracy (%) obtained by the Subnet method on the 5-task datasets evaluated across all trained tasks (task 1 – task 5) and across different fault-mitigation techniques.

Dataset	Technique	$5{\times}10^{-9}$	$1{\times}10^{-8}$	$5{\times}10^{-8}$	$1{\times}10^{-7}$	$5{\times}10^{-7}$	$1{\times}10^{-6}$	$5{\times}10^{-6}$	$1{\times}10^{-5}$
Split MNIST	Baseline	**99.84**	**99.83**	90.05	76.38	51.28	50.32	51.28	51.28
	RobustNCLv1	99.81	99.78	99.79	**99.66**	99.72	99.77	99.77	99.79
	RobustNCLv2	99.81	99.78	**99.80**	**99.66**	99.74	**99.79**	**99.80**	**99.82**
	RobustNCLv3	99.81	99.78	99.79	**99.66**	**99.76**	99.76	99.72	99.80
Split FMNIST	Baseline	**99.01**	89.36	66.79	57.54	50.00	50.00	50.00	50.00
	RobustNCLv1	**99.01**	99.05	99.03	**99.00**	98.98	99.06	**99.05**	99.01
	RobustNCLv2	**99.01**	**99.06**	99.03	98.95	99.02	**99.09**	99.04	**99.04**
	RobustNCLv3	**99.01**	99.05	**99.04**	98.96	**99.07**	**99.09**	98.99	99.01
Split CIFAR-10	Baseline	84.35	84.74	76.79	70.39	50.00	50.00	50.00	50.00
	RobustNCLv1	**84.85**	**85.17**	84.89	85.42	85.02	**85.17**	84.92	85.05
	RobustNCLv2	**84.85**	**85.17**	**84.92**	85.42	85.10	84.78	**85.71**	**85.25**
	RobustNCLv3	**84.85**	**85.17**	84.88	**85.43**	**85.23**	84.92	85.19	84.88

Table 2: Final average accuracy (%) obtained by the MR method on the 5-task datasets evaluated across all trained tasks (task 1 – task 5) and across different fault-mitigation techniques.

Dataset	Technique	1×10^{-7}	5×10^{-7}	1×10^{-6}	5×10^{-6}	1×10^{-5}	5×10^{-5}	1×10^{-4}	5×10^{-4}
Split MNIST	Baseline	**99.40**	**99.37**	99.38	**99.47**	98.61	51.50	51.50	51.50
	RobustNCLv1	99.33	99.31	99.41	99.22	99.40	**99.41**	99.37	99.34
	RobustNCLv2	99.39	99.35	99.19	99.44	**99.49**	99.30	99.28	**99.51**
	RobustNCLv3	99.29	99.32	**99.49**	99.38	99.40	99.23	**99.39**	99.37
Split FMNIST	Baseline	99.03	98.99	**99.08**	98.89	87.37	94.14	50.00	50.00
	RobustNCLv1	99.01	98.97	99.07	98.96	99.00	99.06	**99.01**	98.31
	RobustNCLv2	**99.07**	**99.06**	99.04	**99.06**	**99.01**	99.07	98.96	**98.92**
	RobustNCLv3	99.06	98.98	99.01	99.01	98.99	**99.08**	**99.01**	98.81
Split CIFAR-10	Baseline	**88.80**	88.39	88.49	86.53	87.85	50.00	50.00	50.00
	RobustNCLv1	88.31	**88.76**	88.10	88.60	87.31	88.37	87.64	86.31
	RobustNCLv2	88.03	88.59	**88.58**	88.53	88.32	**88.98**	**88.15**	86.37
	RobustNCLv3	87.61	88.12	88.47	**88.67**	**88.56**	87.85	87.07	**88.31**

Table 3: Final average accuracy (%) obtained by both the Subnet and MR methods on the 10-task dataset (Split CIFAR-100) evaluated across all trained tasks (task 1 – task 10).

Technique	Subnet				MR			
	1×10^{-8}	1×10^{-7}	1×10^{-6}	1×10^{-5}	1×10^{-8}	1×10^{-7}	1×10^{-6}	1×10^{-5}
Baseline	60.00	10.00	10.71	10.00	66.66	66.62	10.00	10.00
RobustNCLv1	**70.73**	70.61	70.61	70.82	**67.11**	**66.91**	**66.84**	66.35
RobustNCLv2	**70.73**	**70.66**	70.50	70.59	66.75	66.35	66.59	65.90
RobustNCLv3	**70.73**	70.57	**70.66**	**71.02**	66.65	66.63	66.25	**66.82**

5.2 Overheads of RobustNCL

RobustNCL incurs negligible processing time and memory overheads as compared to the baseline without any fault mitigation (i.e., up to 2.3% latency overheads and less than 1% memory overheads) across different NCL methods, datasets, and fault rates. This is because RobustNCL only requires small additional time and memory for performing the D&C process for each weight accessed from the DRAM, and for updating the curing weight library after the NCL training phase. RobustNCL also incurs a small overhead of energy consumption (i.e., 6.35% on average) across different NCL methods, datasets, and fault rates. The reason is that RobustNCL incurs moderate additional power to perform the D&C process and to update the curing weight library after the NCL training phase. In summary, these overhead results and analysis highlight that our RobustNCL framework successfully offers a low-cost fault-mitigation solution against hardware-level transient faults in the DRAM across different NCL methods, datasets, and fault rates, showcasing its good generality and scalability.

6 Conclusion

In this paper, we propose a novel RobustNCL framework to improve NCL reliability in the presence of transient faults in the DRAM off-chip/main memory through a lightweight detect-and-cure strategy. It detects faults during weight data access from the DRAM, then cures them through three curing variants: replacement with zero, min/max value, and most-probable value from the clean/cured weight distribution. RobustNCL successfully preserves high accuracy scores under high fault rates close to those of fault-free models (i.e., within -0.5% accuracy for the best RobustNCL variant) and significantly improves accuracy from the baseline without fault mitigation across different NCL methods and benchmarking datasets, while incurring negligible latency and energy overheads. These results showcase that our RobustNCL framework is a promising solution to realize reliable and energy-efficient NCL systems.

AI USE STATEMENT

AI tools were used solely for polishing writing. They did not contribute to the research content or scientific findings of this work.

ETHICS STATEMENT

The research conducted in the paper conforms to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We have provided implementation details in Sec. 4. We will release the code upon paper acceptance.

REFERENCES

Dmitry Antonov, Kirill Sviatov, and Sergey Sukhov. Continuous learning of spiking networks trained with local rules. *Neural Networks*, 155:512–522, 2022. doi: 10.1016/j.neunet.2022.09.003.

Marco P. E. Apolinario, Kaushik Roy, and Charlotte Frenkel. TESS: A scalable temporally and spatially local learning rule for spiking neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–9, 2025. doi: 10.1109/IJCNN64981.2025.11227652.

Robert C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005. doi: 10.1109/TDMR.2005.853449.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. doi: 10.1109/MM.2018.112130359.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. doi: 10.1109/TPAMI.2021.3057446.

Alberto Dequino, Alessio Carpegna, Davide Nadalini, Alessandro Savino, Luca Benini, Stefano Di Carlo, and Francesco Conti. Compressed latent replays for lightweight continual learning on spiking neural networks. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 240–245, 2024. doi: 10.1109/ISVLSI61997.2024.00052.

Rajesh Garg, Nikhil Jayakumar, Sunil P Khatri, and Gwan S Choi. Circuit-level design approaches for radiation-hard digital electronics. *IEEE transactions on very large scale integration (VLSI) systems*, 17(6):781–792, 2009.

Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. Demystifying complex workload-dram interactions: An experimental study. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–50, 2019.

Bing Han, Feifei Zhao, Yi Zeng, Wenxuan Pan, and Guobin Shen. Enhancing efficient continual learning with dynamic structure development of spiking neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2993–3001, 2023. doi: 10.24963/ijcai.2023/334.

Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. FT-ClipAct: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1241–1246, 2020. doi: 10.23919/DATE48585.2020.9116571.

Qiang Huang and Jin Jiang. An overview of radiation effects on electronic devices under severe accident conditions in npps, rad-hardened design techniques and simulation tools. *Progress in Nuclear Energy*, 114:105–120, 2019.

Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 65–76. IEEE, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. Raidr: Retention-aware intelligent dram refresh. *ACM SIGARCH Computer Architecture News*, 40(3):1–12, 2012.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 6467–6476, 2017.

Robert E Lyons and Wouter Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962. ISSN 0018-8646.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. doi: 10.1016/S0893-6080(97)00011-7.

Micron. Micron 2gb: x4, x8, x16 ddr3 sdram. datasheet mt41j128m16ha-12. 2010.

Mishal Fatima Minhas, Rachmad Vidya Wicaksana Putra, Falah Awwad, Osman Hasan, and Muhammad Shafique. Continual learning with neuromorphic computing: Foundations, methods, and emerging applications. *IEEE Access*, 13:124824–124873, 2025a. doi: 10.1109/ACCESS.2025.3588665.

Mishal Fatima Minhas, Rachmad Vidya Wicaksana Putra, Falah Awwad, Osman Hasan, and Muhammad Shafique. Replay4NCL: An efficient memory replay-based methodology for neuromorphic continual learning in embedded AI systems. In *62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025b. doi: 10.1109/DAC63849.2025.11132839.

Lorenzo Pes, Rick Luiken, Federico Corradi, and Charlotte Frenkel. Active dendrites enable efficient continual learning in time-to-first-spike neural networks. In *IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pp. 41–45, 2024. doi: 10.1109/AICAS59952.2024.10595872.

Michela Proietti, Alessio Ragno, and Roberto Capobianco. Memory replay for continual learning with spiking neural networks. In *IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, 2023. doi: 10.1109/MLSP55844.2023.10285911.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments. In *58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1057–1062, 2021. doi: 10.1109/DAC18074.2021.9586281.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. lpspikecon: Enabling low-precision spiking neural network processing for efficient unsupervised continual learning on autonomous agents. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Drmap: A generic dram data mapping policy for energy-efficient processing of convolutional neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Romanet: Fine-grained reuse-driven off-chip memory access management and data organization for deep neural network accelerators. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(4):702–715, 2021a. doi: 10.1109/TVLSI.2021.3060509.

Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram. In *58th ACM/IEEE Design Automation Conference (DAC)*, pp. 379–384, 2021b.

Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. SoftSNN: Low-cost fault tolerance for spiking neural network accelerators under soft errors. In *59th ACM/IEEE Design Automation Conference (DAC)*, pp. 151–156, 2022. doi: 10.1145/3489517.3530657.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. doi: 10.1038/s41586-019-1677-2.

Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *Journal of Intelligent & Robotic Systems*, 105(1):9, 2022.

Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. Bayesian continual learning via spiking neural networks. *Frontiers in Computational Neuroscience*, 16:1037976, 2022. doi: 10.3389/fncom.2022.1037976.

Hung Y. Sze. Circuit and method for rapid checking of error correction codes using cyclic redundancy check, July 18 2000. US Patent 6,092,231.

Ramakrishna Vadlamani, Jia Zhao, Wayne Burleson, and Russell Tessier. Multicore soft error rate stabilization using adaptive dual modular redundancy. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 27–32. IEEE, 2010.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(8):5362–5383, 2024. doi: 10.1109/TPAMI.2024.3367329.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Hebbian learning based orthogonal projection for continual learning of spiking neural networks. In *International Conference on Learning Representations (ICLR)*, 2024. URL `https://openreview.net/forum?id=MeB86edZ1P`.