Security of NVMe Offloaded Data in Large-Scale Machine Learning

Torsten Krauß[1] [0000−0003−0810−6646], Raphael Götz[1] [0009−0002−7208−738X], and Alexandra Dmitrienko[1] [0000−0001−5637−7016]

University of Würzburg, Sanderring 2, 97070 Würzburg, Germany
{torsten.krauss,raphael.goetz,alexandra.dmitrienko}@uni-wuerzburg.de

Abstract. Large-scale machine learning (LSML) models, such as the GPT-3.5 that powers the well-known ChatGPT chatbot, have revolutionized our perception of AI by enabling more natural, context-aware, and interactive experiences. Yet, training such large models nowadays requires multiple months of computation on expensive hardware, including GPUs, orchestrated by specialized software, so-called LSML frameworks. Due to the model size, neither the on-device memory of GPUs nor the RAM is capable of holding all parameters simultaneously during training. Therefore, LSML frameworks dynamically offload data to NVMe storage and reload the information just in time.

In this paper, we investigate the security of NVMe offloaded data in LSML against poisoning attacks and present *NVMevade*, the first *untargeted* poisoning attack on NVMe offloads. NVMevade allows the attacker to reduce the model performance, as well as slow down or even stall the training process. For instance, we demonstrate that an attacker can achieve a stealthy increase of 182% in training time, thus, inflating costs for model training. To address this vulnerability, we develop *NVMensure*, the first defense that guarantees the integrity and freshness of NVMe offloaded data in LSML. By conducting a large-scale study, we demonstrate the robustness of NVMensure against poisoning attacks and explore runtime efficiency and security trade-offs it can provide. We tested 22 different NVMensure configurations and report an overhead between 9.8% and 64.2%, depending on the selected security level. We also note that NVMensure is going to be effective against *targeted* poisoning attacks which do not exist yet but might be developed in the future.

Keywords: Large-Scale Machine Learning · NVMe Offload · Poisoning Attacks.

1 Introduction

Machine Learning (ML) enables the extraction of knowledge from datasets, which can then be utilized for prediction or classification tasks on unseen data. Usually, increased model sizes and larger datasets result in a greater amount of encapsulated knowledge in the ML model.

ML involving large Deep Neural Networks (DNNs), commonly known as Deep Learning (DL), produces models with multiple layers and numerous model parameters. With its capability to extract and encapsulate extensive knowledge, DL surpasses alternative methods across diverse domains, including image classification [22], speech recognition [15], text generation [5], language processing [9], as well as fraud and malware detection [17,11]. Scenarios, where the model and dataset exceed easily manageable sizes are referred to as large-scale machine learning (LSML).

Currently, there is a noticeable arms race focused on the size and capacity of large-scale machine learning (LSML) models. This trend can be observed from the development of models such as GPT [40] with 110 million parameters trained in 2018, followed by GPT-2 [41] (1.5 billion, 2019), Megatron-LM [53] (8.3 billion, 2019), Turing-NLG [25] (17.2 billion, 2020), and GPT-3 [5] (175 billion, 2020), which is currently utilized in the free version of ChatGPT [30]. However, the progression continues, with models like Megatron-Turing [54] (530 billion, 2022) and GPT-4 [31,6] (around 1 trillion, 2023) surpassing their predecessors and setting the trend for even larger models. Training such models is not feasible in standard ML environments like PyTorch [34]. Instead, specialized LSML frameworks like DeepSpeed [48] are employed. Additionally, optimized hardware such as NVIDIA DGX clusters [29] is necessary to provide the required computational power. Given the size of the models and the volume of data involved in training, LSML frameworks incorporate parallelism strategies to distribute the training process across multiple hardware instances and accelerators. This parallelism is crucial even in high-performance hardware setups. While performance is prioritized, the security aspects are neglected, which, as we show in this paper, leads to vulnerabilities to poisoning attacks [55].

Poisoning attacks come in two flavors: 1) *untargeted* attacks [19,62,61] strive to negatively impact the prediction performance of the model, while 2) *targeted* attacks [2,8,51] aim to embed trojan behavior into the model. Both attack versions can have visible effects on the final model or remain stealthy. As the size of models increases, it becomes easier to introduce stealthy poisonings since the inner workings of the model are often opaque. However, for those training LSML models, the trained models are valuable intellectual property, and the quality and security of the model predictions are of utmost importance. Improved performance translates to a competitive advantage, while reliable and secure predictions are vital for establishing trust in the technology's future use. Therefore, preventing or at least recognizing poisoning attacks is a necessary and critical step in LSML training, as such attacks can significantly impact both business value and end-user security. Ideally, countermeasures should focus on attack prevention rather than detection since eliminating poisonings may require costly model re-training on extensive hardware infrastructure [43].

Challenges and State-of-the-Art Solutions. The manipulation of the LSML training process by adversarial actors presents a significant challenge, leading to undesired model behavior and financial losses. These attacks encompass various vectors, including poisoned datasets, malicious code changes, and adversarial perturbations on communication channels. Efforts have been made to secure a significant portion of DNN training through technical measures. To ensure benign datasets, human experts or automatic filtering mechanisms [35,14,36] can eliminate potentially malicious data from the training set. Recent research, such as Slalom [57], Graviton [59], HETEE [64], and HIX [20], has proposed protection strategies for secure computation on GPUs and trustworthy code execution on CPUs [39,23]. These advancements help ensure that ML training, fueled by benign data, produces benign models. However, to generate models of maximal size, state-of-the-art LSML frameworks like DeepSpeed [48] employ techniques where idle portions of training data are temporarily offloaded to Nonvolatile Memory Express (NVMe) storage and reloaded onto the CPU or GPU for further training in a just-in-time manner [44,45,49]. While the offloaded data presents an attractive target for attacks on the training process, no previous work has explored the feasibility of such attacks or proposed defenses so far.

Contributions. To emphasize and address the aforementioned issue, this paper provides the following contributions:

- **Attack on NVMe Offloads:** We develop *NVMevade*, the first untargeted poisoning attack on NVMe offloaded data, highlighting the vulnerability of NVMe offload in LSML frameworks to such attacks. We present compelling evidence that our attack can effectively achieve the following outcomes, regardless of the structure of the offloaded data: i) Completely disrupt the training process, necessitating retraining. ii) Reduce prediction performance, resulting in a competitive disadvantage. iii) Slow down the training process without impacting prediction performance, leading to increased computational costs. We successfully implement NVMevade for the DeepSpeed framework, showcasing its ability to quickly terminate the training process, discreetly degrade model performance, and significantly prolong training time by 182%. These outcomes would inevitably lead to considerable financial losses within LSML setups.
- **Security for NVMe Offloads:** We present *NVMensure*, our defense mechanism designed to detect and mitigate poisoning attempts on NVMe offloaded data during training. This method exhibits resilience not only against untargeted poisoning attacks such as NVMevade but also against potential targeted attacks that may emerge in the future. NVMensure ensures the integrity and freshness of the data, providing protection against attacks such as NVMevade. Importantly, it allows the model creator to adjust the trade-off between security and training speed according to their specific requirements.
- **Large-Scale Study:** We have implemented NVMensure for the DeepSpeed framework and conducted a comprehensive evaluation of its runtime and storage efficiency. This involved assessing 22 different versions of the defense, allowing us to identify critical bottlenecks and propose enhancements. To mitigate runtime and memory overhead, we have leveraged space-efficient data structures and employed integrity mechanisms that provide various levels of collision resistance. By combining these methods with multi-core execution to utilize idle CPU resources, we have achieved an acceptable

overhead ranging from 9.8% to 64.2%, depending on the desired security level. Our optimizations effectively reduce the computational impact and storage requirements, while ensuring the defense remains highly effective.

NVMensure is orthogonal to previous works on securing CPU-based [39,23] and GPU-based ML workloads [57,59,64,20], and can be combined with them. Hence, NVMensure is a crucial component for enhancing the overall security of LSML.

2 Background

Below, we introduce the reader to DeepSpeed [48], the de facto framework for large-scale distributed ML training [24] before providing background information about poisoning attacks in ML.

Large-Scale Machine Learning with DeepSpeed. LSML models such as Megatron-Turing [54] would require more than ten terabytes of memory, surpassing the capacity of current GPUs.[1] Thus, LSML frameworks incorporate various forms of parallelism that enable the partitioning of data and computations across multiple hardware devices and accelerators. Thereby, data [44], pipeline [27], and model [53] parallelism strategies facilitate efficient computation during training by leveraging powerful hardware setups like NVIDIA DGX clusters [29]. However, with increasing model size, the limiting factor of training environments is not the computational power, but the GPU memory.

Addressing this problem, DeepSpeed [48], an open-source LSML framework developed by Microsoft in 2020, has become the industry standard for large-scale distributed ML [24]. DeepSpeed's pioneering NVMe offload capability has been key in achieving this milestone, enabling the training of large-scale models. Other frameworks focusing on one specific strategy, like model parallelism in Megatron-LM [53], lack the ability to train independently at such a scale. Hence, the project creators collaborated and integrated multiple methods, including Megatron-LM, into DeepSpeed. Offering ZeRO-Infinity [45], DeepSpeed resolves memory issues by offloading idle training data (model parameters, optimizer states, and gradients) from GPU and CPU to cost-efficient NVMe storage[2]. On-demand, the DeepNVMe [45] library prefetches and broadcasts the offloads to CPUs and GPUs as needed. In this paper, we propose NVMevade, an attack on NVMe offloads, and introduce NVMensure as a defense.

Poisoning attacks in Machine Learning. Poisoning attacks on ML models [56,60] can be classified into two categories based on the adversary's objectives: Untargeted attacks and targeted (or backdoor) attacks. Untargeted attacks aim to hinder the convergence of the global model. As an example, an untargeted attack can assign false labels to samples in the training dataset. Alternatively, the adversary can also manipulate the code running the training process itself, e.g., by manipulating model parameters or intermediate values like gradients.

[1] A NVIDIA A100 GPU [28] provides 80 GB of on-device memory.

[2] NVMe is an interface specification for PCIe attached Flash and SSD storage devices.

The attacker can choose to completely destroy the model or employ a stealthy approach that reduces the model's accuracy or slows down the training process, thereby increasing computational effort. On the other hand, targeted or backdoor attacks aim to introduce a backdoor trap into the model. These attacks typically consist of a trigger in the input data and a target prediction chosen by the adversary. The goal is to maintain the model's benign prediction performance while embedding the hidden malicious behavior.

Backdoor attacks often pose a higher risk since the resulting model surprisingly misbehaves, but in LSML high performance is a crucial competitive advantage making untargeted attacks also very attractive for adversaries. Both attacks can be cured by retraining or fine-tuning, but in LSML this involves high costs, making untargeted attacks a substantial danger. This paper conducts untargeted attacks in LSML by manipulating NVMe offloaded model parameters or intermediate values and proposes NVMensure as a defense method against both untargeted and targeted poisoning attacks.

3 NVMevade - Untargeted Poisoning Attack on NVMe

In this section, we introduce NVMevade, the first untargeted poisoning attack that manipulates NVMe offloaded data exposing vulnerabilities of current LSML frameworks. NVMevade offers three distinct operation modes that allow for the complete cessation of the training process or the discreet degradation of model performance and training efficiency through techniques such as bit-flips or replay attacks on NVMe-offloaded data.

3.1 System Model

Fig. 1 provides a visualization of an LSML setup, illustrating the four general steps of an LSML training process. It also demonstrates the scope of NVMevade (and NVMensure), as well as their integration with related works, offering comprehensive end-to-end security for LSML training.

Considered System. We consider a general LSML training process as depicted in Fig. 1 and consisting of four steps: 1) The training data is supplied to the computation device, typically a server equipped with CPUs and RAM. 2) During training, certain computations are subsequently offloaded to GPUs for accelerated processing. This involves copying model parameters to the GPUs' on-device memory. 3) When the GPU memory is fully utilized, causing it to become the system bottleneck, the LSML framework addresses this issue by offloading idle portions of the model parameters and training-internal intermediate values to separate files in NVMe storage so that upcoming computations can be conducted on the GPUs. 4) When needed for ongoing model training, the framework loads offloaded data back onto the GPU.

Attackers Goals and Capabilities. The attacker targets model creators that possess enough data to train a LSML model. Thereby, he strives to negatively impact the training process by manipulating the data offloaded to the NVMe.

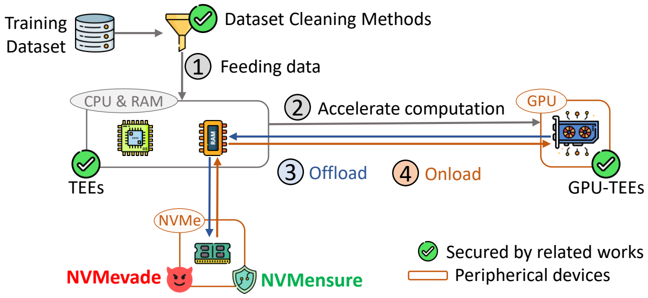

Fig. 1. LSML system overview depicting NVMevade's and NVMensure's scope. Parts of the system can be secured by related works (cf. Sect. 5).

Specifically, depending on the concrete goal, one of the following three effects should be triggered by conducting perturbations on the offloaded data: (i) destruction of the model performance or termination of the training process, (ii) stealthy model performance reduction, or (iii) stealthy slowdown of the training speed. These objectives should be accomplished solely through user-level access[3] to the NVMe offloaded data, without requiring access to the CPU, RAM, or other peripheral devices involved in the training process, such as GPUs. The attack needs to be conducted in the timing window between offload and onload of the respective data.

3.2 Design of NVMevade

In Fig. 2, we illustrate the design of NVMevade, which comprises four steps: 1) The attacker conducts a thorough scan of the NVMe storage to identify new offloads. Thereby, LSML frameworks store different parameters in separate files. 2) The adversary loads the newly offloaded data from the NVMe to his own RAM, then 3) poisons the data by using one of the three possible strategies, which we will describe below as *poisoning modes*. 4) Finally, the original data on the NVMe are replaced with the poisoned version.

Poisoning Modes. Depending on the effect that the attacker wants to trigger, NVMevade offers three modes, as can be retraced in Fig. 2:

- *Mode 1: Model Destruction.* This mode applies multiple random bit-flips to the NVMe offloaded data, aiming to transform the model into a naïve classifier or halt the learning process altogether. In mode 1, where all bytes are aggressively poisoned, NVMevade is agnostic to the particular LSML framework implementation and does not consider any knowledge of data types or formats.

[3] We posit that the attacker has effectively circumvented OS access controls, thus obtaining necessary user-level permissions to access and manipulate NVMe files.

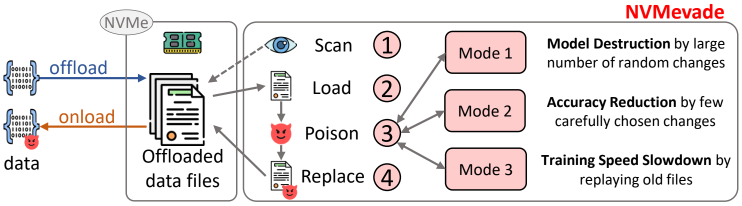
NVMevade
NVMe
offload
onload
data
Offloaded data files
Scan 1
Load 2
Poison 3
Replace 4
Mode 1
Mode 2
Mode 3
Model Destruction by large number of random changes
Accuracy Reduction by few carefully chosen changes
Training Speed Slowdown by replaying old files

Fig. 2. Overview of NVMevade.

- *Mode 2: Accuracy Reduction.* To reduce the prediction performance without interrupting the training process or producing any noticeable traces for the model creator, mode 2 applies a reduced (in comparison to mode 1) number of bit-flips at the byte level. To achieve a trade-off between reduction performance and stealthiness, the attacker parameterizes the attack by defining parameters (cf. App. Tab. 6) that specify the level of poisoning. Thereby, the type of offloaded parameters is considered, making it the only part of the paper which is not agnostic to DeepSpeed.
- *Mode 3: Training Speed Slowdown.* Conducting a replay attack allows an attacker to slow down the training process while remaining stealthy to the model creator. By capturing offloaded data and replaying them later on, more realistic poisoned data are loaded from the NVMe, exerting a negligible effect on the model performance.[4] The data is replaced at the granularity of files and without consideration of exact file contents, thus remaining agnostic to the implementation of any particular LSML framework.

Among the three modes, mode 1 is very effective but can be detected by monitoring. Further, countermeasures like backups can minimize negative effects. While mode 1 already highlights the vulnerability of NVMe offloads, modes 2 and 3 pose a higher risk in real-world scenarios, due to their stealthiness.

Parameter Configurations. To poison the offloads in mode 1 and 2, we apply bit-flips using a random mask and XOR operation. Six parameters listed in App. Tab. 6 control the extent of poisoning. Namely, for each file, we traverse all bytes considering only a portion of those bytes with respect to P_1. Then, since DeepSpeed is configured with float16 parameters, we check if the actual byte is the upper or lower half of the parameter. For each of the bytes (upper or lower), we decide on the amount of poisoning based on the probability in P_2 and P_3 and start by poisoning the least significant bits (LSBs) of the byte to reduce the influence in the final parameter. For example, in the case of 25% for P_3, the two LSBs of the upper byte are combined with a random mask. When poisoning a byte, we increase a respective counter for upper or lower bytes and stop poisoning upper or lower bytes if a respective threshold ($P4$ and $P5$) is surpassed. Finally, P_6 limits the time window of poisoning data at a stretch.

[4] The replay attack naturally also affects the model performance, but the effect was very marginal and not recognizable in our experiments.

These parameters enable precise configuration of the poisoning level at the lowest granularity, effectively controlling the intensity and stealthiness of the attack.

Instantiation of NVMevade for DeepSpeed. We implemented NVMevade for DeepSpeed [48], the de facto LSML framework. We present our observations and challenges in detail in App. A.1. Here, we want to highlight, that the timing window between offload and onload of data, ranging from 0.33 to 0.95 seconds (average of 0.37 seconds) in our experiments, was sufficient to conduct the attacks and hence did not pose a technical obstacle.

3.3 Evaluation

To efficiently test our approaches, we simulate the NVMe offload of LSML training in a small-scale setup. Specifically, we used a server with 32GB RAM running Ubuntu 20.04 equipped with a single Nvidia Quadro P2200 GPU with 5GB GPU memory and a Western Digital 256 GB NVMe. As LSML framework, DeepSpeed [48] is leveraged to train a T5-small [42] model on translation tasks using the wmt16 (ro-en) dataset [4]. The AdamW optimizer and DeepSpeed's WarmupLR learning rate scheduler were used together with float16 mixed precision training and ZeRO-Infinity [45], which enables the NVMe offload.

Metrics. LSML models are predominantly language models in present times. To evaluate the performance of these models, we assess their prediction accuracy on a test set, indicated by the loss (e.g., cross-entropy) computed through the forward path on the test data. A lower loss value signifies better model performance and quality. The sacreBLEU score [38] is a computationally efficient and highly comparable version of BLEU, widely used for automatic machine translation evaluation. It demonstrates a strong correlation with human evaluations, where a score of 100 represents a perfect prediction and a score of 0 indicates a complete mismatch. Regarding the training speed, one can analyze the training samples per second (SPS), that are processed by the ML system. Further, the seconds per epoch (SPE) indicates the duration for one iteration over the training dataset.

Mode 1: Model Destruction. To conduct mode 1 attacks, we set all parameters to a maximum as listed in App. Tab. 6, resulting in the highest level of poisoning, meaning that all offloaded bytes were randomly changed. Once the attack started, DeepSpeed was faced with overflows resulting in skipping of the optimizer step and a complete stop within a few skips, in our case 15. Therefore, all previously unsaved training steps are lost. If we switch off DeepSpeed's internal overflow skipping procedure, the model becomes naïve within a few optimizer steps. Hence, depending on the internals of the LSML framework, NVMevade's mode 1 destroys the model or, in the case of DeepSpeed, stops the entire training procedure, showing the vulnerability of NVMe offloaded data.

Mode 2: Accuracy Reduction. To achieve a stealthy model performance reduction, extensive fine-tuning of the parameters is required, which we list in App. Tab. 6. The parameters allow only minimal poisoning since even a small number of bit flips can cause massive damage to a model [63]. By executing the attack, we achieved a substantial increase in loss on the test set from 3.3 to 6.8615 and a drastic reduction in the sacreBLEU score [38] from 23.9 to 0.0199, indicating a near-complete mismatch in all predictions. However, by modifying random seed of our test setup, which impacts the ML process and NVMevade, we observed a reduced influence of the attack. This resulted in a loss of 3.9932 and a sacreBLEU score of 5.8927 and shows, that the manual fine-tuning of the parameters is very sensitive as no continuous schema could be identified so far. Simultaneously to the attack, no recognizable warning being a sign for an attack was produced by DeepSpeed[5], thus increasing the stealthiness compared to mode 1, where DeepSpeed stopped the training process. However, since the model performance is reduced, loss and accuracy values calculated during training would be conspicuous. Our findings illustrate the potential to degrade model performance during training without interrupting the process. However, selecting suitable parameters is nontrivial. It necessitated 35 fine-tuning steps to identify optimal parameters for a successful stealthy attack. Merely reducing noise in mode 1 proved insufficient and resulted either in model destruction or had no impact making a more complex strategy necessary. While future research might propose advanced poisoning strategies to prevent expensive parameter fine-tuning, our defense, NVMensure, guards against all such variations. Consequently, further exploration in this area is not pursued.

Mode 3: Training Speed Slowdown. To conduct this attack, NVMevade caches offloaded files and as soon as the scanning procedure of NVMevade reports a subsequent NVMe offload with the same file name, the data are replaced with the historical file, thus resulting in a replay attack. Thereby, we could reduce the samples per second (SPS) from 11.802 to 8.910, leading to a total training time for one epoch being 03:44 minutes instead of 02:49. As visualized in Tab. 1, the training efficiency was reduced by 24.5% and the resources for training are leveraged for 32.5% longer. We discovered that our attack faced challenges with the timing constraint, specifically the time between offload and onload. In certain cases, the replay attack was not executed quickly enough. Consequently, we shifted our focus to model parameters based on the offloaded file name, as intermediate training parameters with the same name were often not offloaded twice, leaving no opportunity for a replay attack. Thus, we could worsen those rates by only attacking model parameters to a 69.4% decrease in SPS and 182% increased training time, which could lead to substantial financial losses for model creators. Further, mode 3 addresses the downsides of mode 2, since neither loss nor sacreBLEU score were affected significantly and no visible clues could be detected in DeepSpeed's console output making the attack completely stealthy.

In summary, NVMevade successfully poisons NVMe offloaded data, enabling aggressive or stealthy attacks that reduce training time and model performance. These outcomes lead to financial losses for the model creator, underscoring the importance of prevention in real-world setups.

[5] For stealthiness evaluation, we compared the execution log (console output) with and without attack. We only found different timestamps and minimal loss value changes, which is normal in different runs.

Table 1. Effectiveness of NVMevade's mode 3.

	SPS	SPE (in minutes)	Loss	sacreBLEU
Without attack	11.802	02:49	3.3812	23.9366
NVMevade mode 3	8.910 (**-24.5%**)	03:44 (+**32.5%**)	3.3812 (+/- **0%**)	23.9366 (+/- **0%**)
NVMevade mode 3 (only model parameters)	3.607 (**-69.4%**)	09:14 (+**182%**)	3.3812 (+/- **0%**)	23.9366 (+/- **0%**)

4 NVMensure - Preventing Poisoning Attacks on NVMe

In this section, we present NVMensure, a new defense mechanism designed to counteract poisoning attacks (both targeted and untargeted) on NVMe offloads in LSML. This defense adopts the same system model as described in Sect. 3.1. NVMensure guarantees data integrity by performing checksum calculations during offload and subsequent validation during onload. Additionally, NVMensure ensures freshness by promptly invalidating checksums after validation. To accommodate diverse requirements, NVMensure offers the flexibility to balance security with storage and runtime efficiency. This is achieved through the implementation of various integrity mechanisms for checksum calculation and the utilization of space-efficient data structures.

4.1 Design of NVMensure

As depicted in Fig. 3, NVMensure is comprised of eight distinct steps, which are divided between offload and onload: 1) When DeepSpeed initiates an offload, NVMensure carries out a data integrity mechanism (DIM) to compute a checksum (CS). 2) This checksum is then inserted into a data structure (DS) residing in the RAM. 3) The regular offload to the NVMe continues thereafter. 4) Once DeepSpeed triggers an onload, a second checksum is created from the data loaded from the NVMe. 5) The checksum associated with the respective offload is retrieved from the data structure. 6) Both checksums are compared,

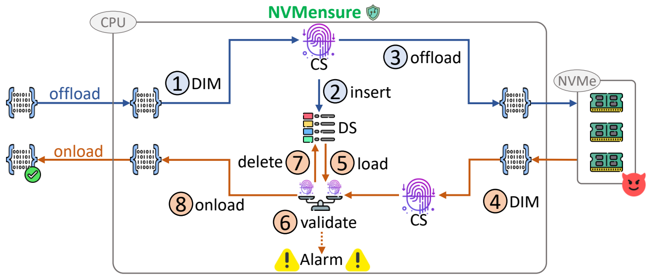

Fig. 3. Overview of NVMensure. A data integrity mechanism (DIM) computes checksums (CSs) which are stored in a data structure (DS) on the RAM during offload and validated during onload.

Table 2. NVMensure's data structures.

	Advantages	Disadvantages
DS_1: List	- No False Positives **- Most secure**	- Linear increase of size - Potential bottleneck when scaling up
DS_2: Bloom Filter [3]	- Constant size - Simple implementation - Most size efficient	- No element deletion resulting in a high false positive rate (FPR)
DS_3: Counting Bloom Filter [16]	- Constant size - Allows deletion of elements - Lower FPR than DS_2 due to deletion of elements	- Less size efficient than DS_2 - Low FPR
DS_4 Cuckoo Filter [12]	- Constant size **- More size efficient than DS_3** - Allows deletion of elements - Lower FPR than DS_2 (deletion)	- Less size efficient than DS_2 - Low FPR

and if they differ, an alarm is raised.[6] This ensures integrity, as only valid checksums are present within the data structure. 7) The checksum corresponding to the offload is immediately removed from the data structure, ensuring freshness, as the utilized checksum becomes invalid on removal. 8) The validated data is utilized for training purposes.

Storage-Efficient Data Structures. In LSML setups, where multiple gigabytes of data are offloaded, the size of the data structure holding the checksums can present challenges and become a bottleneck for the system. To mitigate this issue and to optimize the memory usage of NVMensure, the defender has the option to select from four different data structures, which are listed with advantages and disadvantages in Tab. 2. Our baseline, offering the highest level of security, is a straightforward list (DS_1). The remaining three alternatives consist of space-efficient probabilistic data structures, which reduce the memory requirements, but sacrifice some security since they introduce a false positive rate (FPR) dependent on the number of elements they store. Thereby, the Cuckoo Filter [12] (DS_4) poses the fastest alternative with minimal security sacrifices.

Runtime-Efficient Integrity Mechanisms. Like any other security mechanism, NVMensure introduces additional overhead. To balance runtime efficiency and security, the defender is provided with a choice of five different integrity mechanisms, each offering varying levels of collision resistance and computational complexity. All of these considered mechanisms require the attacker to adapt their perturbations in a manner that results in a checksum collision with the benign data, to execute a successful and covert attack. The feasibility of achieving this task depends on the chosen mechanism and the time window the offloads reside on the NVMe, typically spanning only a few seconds. Therefore, to encompass the full spectrum of the trade-off, it is reasonable to also consider algorithms that do not provide formal collision resistance. Such methods can still effectively detect poisoning attacks like NVMensure and offer the potential for significant performance advantages over more intricate approaches. We present a comprehensive overview of integrity mechanisms, along with their

[6] In our experiments the training then stopped. Certainly, real-world scenarios can roll back to a certain checkpoint and continue training automatically.

Table 3. NVMensure's data integrity mechanisms.

	Advantages	Disadvantages
DIM_1 BLAKE2bp [1]	**- SHA-3 security standard** - High-performance (multi-threaded)	- New and unestablished
DIM_2 SHA-256 [13]	- Collision resistant hash function - More established than DIM_1	- Computationally complex - Some vulnerabilities known
DIM_3 MD5 [50]	- Simplicity & wide adaption - Runtime efficiency	- Less secure than DIM_1 & DIM_2 - Collision & preimage attacks
DIM_4 CRC-32 [37]	- Simplicity & cross-domain adaption - Faster than DIM_1 - DIM_3	- Limited error detection - Not designed for intentional attacks
DIM_5 LRC [18]	- Most simple solution **- Fastest solution**	- Designed for consecutive errors - Lowest security level

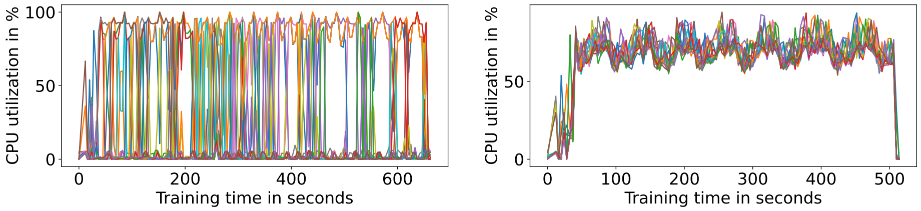

Fig. 4. Comparison of the CPU utilization per core for BLAKE2bp. One color represents one CPU core. On the left, one BLAKE2bp instance runs on a single thread, on the right eight instances are processed by eight threads. The multi-threaded version improves resource utilization by leveraging idle CPU cores.

respective advantages and disadvantages, in Tab. 3. Among these mechanisms, BLAKE2bp [1] (DIM_1) stands out as the fastest collision-resistant option. On the other hand, while the Longitudinal Redundancy Check [18] (LRC, DIM_5) is the overall fastest version, it sacrifices a significant degree of security.

4.2 Instantiation of NVMevade for DeepSpeed

We implemented NVMensure inside DeepSpeed's DeepNVMe [45] library, utilizing efficient algorithm implementations for optimal performance. We present details in App. A.2. During our experiments, we found that the runtime is mainly affected by single-core CPU utilization. To address this, we implemented LRC as the fastest and BLAKE2bp as the most secure integrity mechanisms, leveraging multiple CPU cores. By splitting the offloaded data into equal segments and calculating checksums concurrently on different cores in different threads, we optimize the utilization of idle CPU resources and greatly enhance the performance of these versions. This positive effect can be observed in Fig. 4.

4.3 Evaluation

To assess the effectiveness of our defense, we utilize the identical experimental setup as in Sect. 3.3. To assess the performance of NVMensure, we conducted extensive benchmarking on all 20 combinations of data structures and integrity

Table 4. Comparison of SPS and SPE to the baseline without defense, categorized by data structures and integrity mechanism, including the BLAKE2bp and LRC multi-core versions (M-C).

Data Structure	Integrity Mechanism	SPS	SPE (in minutes)	SPS decrease in %
Without NVMensure		**11.802**	**02:49.45**	**0%**
List	BLAKE2bp [1]	3.233	10:18.55	72.6%
	SHA-256 [13]	2.457	13:22.58	79.1%
	MD5 [50]	1.160	29:17.00	90.1%
	CRC-32 [37]	3.826	08:42.67	67.5%
	LRC [18]	6.134	05:26.05	48.0%
	(M-C) BLAKE2bp [1]	**4.224**	**07:53.50**	**64.2%**
	(M-C) LRC [18]	**10.642**	**03:07.92**	**9.8%**
Bloom Filter [3]	BLAKE2bp [1]	3.128	10:30.88	73.4%
	SHA-256 [13]	2.491	13:22.85	78.9%
	MD5 [50]	1.124	30:14.00	90.4%
	CRC-32 [37]	3.816	08:44.07	67.6%
	LRC [18]	6.163	05:24.53	47.7%
Counting Bloom Filter [16]	BLAKE2bp [1]	3.257	10:05.78	72.4%
	SHA-256 [13]	2.461	13:21.20	79.1%
	MD5 [50]	1.136	30:14.00	90.3%
	CRC-32 [37]	3.817	08:44.02	67.6%
	LRC [18]	6.175	05:23.90	47.6%
Cuckoo Filter [12]	BLAKE2bp [1]	3.254	10:14.69	72.4%
	SHA-256 [13]	2.460	13:21.72	79.1%
	MD5 [50]	1.115	30:42.00	90.5%
	CRC-32 [37]	3.806	08:45.42	67.7%
	LRC [18]	6.153	05:25.04	47.8%

mechanisms. Additionally, we evaluated the two additional versions that leverage multi-core execution. During evaluation, we measure the runtime efficiency by analyzing the samples processed per second (SPS) and the duration of one epoch (SPE). The results are presented in Tab. 4 and discussed below. Additionally, we track the utilization statistics of the GPU, RAM, and CPU to identify any bottlenecks and to illustrate the impact of NVMensure on the system.

Storage Efficiency. We did not observe significant differences in RAM utilization, hence the choice of data structure should be based primarily on its security. The reason for this is that in our small-scale setup, a relatively small amount of files is created on the NVMe and reused for subsequent offloads of the same parameter. We, anticipate that the data structure choice will significantly impact RAM memory usage when scaling up, potentially becoming a bottleneck.

Runtime Efficiency. Analyzing the measurements presented in Tab. 4, we observe that the runtime impact of the data structure used is negligible. For instance, the decrease in SPS for BLAKE2bp across different data structures is consistently around 3.2, with values of 3.233, 3.128, 3.257, and 3.254 for DIM_1, DIM_2, DIM_3, and DIM_4, respectively. This pattern is similarly observed for all other integrity mechanisms. However, within each data structure, significant efficiency differences can be observed among different integrity mechanisms. For instance, in the case of the list data structure, LRC achieved a speed of 6.134 SPS, making it 5.28 times faster than MD5 with 1.160 SPS and 1.98 times faster than BLAKE2bp with 3.233 SPS. LRC emerged as the fastest integrity mechanism, while MD5 exhibited the slowest performance. Notably, MD5 was even slower than SHA-256 due to our utilization of a kernel implementation for SHA-256. These findings highlight that, depending on the desired balance between security and runtime efficiency, BLAKE2bp is the most sensible choice for prioritizing security, whereas LRC is preferable for maximizing performance. Furthermore, by introducing a multi-core execution of the integrity mechanism, we achieved significant runtime improvements for both algorithms. As a result, we observed performance enhancements ranging from 9.8% to 64.2%, depending on the desired level of security.

Security Considerations. To bypass NVMensure, an attacker must generate a collision on the integrity mechanism by providing modified NVMe offloaded data that produces the same checksum as the unmodified data. SHA-256 and BLAKE2bp are both highly collision-resistant cryptographic hash functions, rendering collisions infeasible, especially with regard to the tight time window. Our experiments reveal that data is typically residing on the NVMe between 0.33 and 0.95 seconds (with an average of 0.37 seconds). Therefore, BLAKE2bp is the optimal choice regarding security due to its exceptional collision resistance. Concerning LRC and CRC-32, generating collisions is achievable within polynomial time, which makes these integrity mechanisms only effective against unsophisticated attacks. Notably, within our experiments, the LRC version detected all NVMevade attacks, demonstrating its efficacy against unadapted attacks.

Probabilistic data structures involve a trade-off between storage efficiency and security, as indicated by the false positive rate (FPR). In our experiments, with a total of 12,548 NVMe offloads in one epoch, the Counting Bloom and Cuckoo Filters, capable of deleting outdated entries, stored up to 127 entries concurrently, resulting in FPRs of 0.02% and 0.0002%, respectively. However, the Bloom Filter lacks element deletion functionality and saved all 12,548 entries simultaneously, leading to a high FPR of 99.7% despite its 19,172-bit capacity. These FPRs show that using probabilistic data structures in NVMensure is acceptable, as long as they support element deletion, without significantly compromising security.

The data's freshness is guaranteed as each checksum remains valid only for a single validation process and is promptly invalidated by removing it from the data structure. Consequently, attempts to substitute new offloads with previously copied data are rendered ineffective, making replay attacks ineffective.

Scaling to Large-Scale Environments. Our experiments were conducted on a small-scale setup, as described in Sect. 3.3, due to limited access to expensive large-scale hardware. However, we anticipate that larger deployments will be more beneficial for our defense, as its runtime overhead will be reduced in larger systems. In large-scale setups, CPU power is typically not a limiting factor, unlike in our small-scale setup. Consequently, idle CPU resources in larger setups can be utilized to handle the additional computational effort required by our defense. This expectation is supported by observations we made when comparing DeepSpeed [48] with and without activated NVMe offload in our small-scale setup. In large-scale setups, NVMe offload improves computational efficiency and enables the use of larger model architectures. However, in our small-scale system,

Table 5. Small-scale setup comparison of DeepSpeed with and without offload.

	SPS	SPE (in minutes)
DeepSpeed without NVMe offload	71.239	00:28.07
DeepSpeed with NVMe offload	11.802	02:49.45
Impact	**-83.43%**	**+5926%**

the runtime efficiency of NVMe-enabled version is reduced by 83.43%, as shown in Tab. 5, due to increased communication overhead and CPU load. Similarly, our defense introduces additional CPU load, suggesting a similar positive effect when scaling up. Regarding memory consumption, the choice of data structure is not critical in small-scale systems. However, when scaling up, the number of offloads and corresponding checksums within the data structure increases. This can become a bottleneck if using a data structure that grows linearly with the number of offloads, such as a list. Then, the Cuckoo Filter [12] is a suitable alternative as it reduces the memory footprint depending on the configuration. Generally, Cuckoo Filters aim to achieve false positive rates on the order of 1% or less. Moreover, the computational complexity of a list is O(n) while that of a Cuckoo Filter is O(1), demonstrating even better runtime efficiency.

5 Related Work

Below, we examine related works, including targeted poisoning attacks against as well as LSML frameworks and existing security methods for various components of an LSML system (cf. Fig. 1), outlining potential synergies with our research.

Targeted Poisoning attacks. NVMensure has been specifically designed to ensure robust security against all types of poisoning attacks, including sophisticated targeted poisoning. Although no instances of such attacks on NVMe offloaded data have been demonstrated thus far, the potential for their development in future work exists. Techniques employed in cutting-edge methods like TBT [46], ProFlip [7], and T-BFA [47], which utilize bit-flips via Rowhammer [26], may be considered for adaptation. However, these white-box attacks require full model access and intricate architectural understanding. They are also offline attacks, which poison a DNN post-training by flipping key bits based on factors such as the model's loss. Adapting these methods for NVMe offloaded data is challenging due to the tight timing constraint and limited model access.

In contrast, NVMevade offers three variations of untargeted poisoning attacks during training, requiring control solely over offloaded model portions or training parameters, and demanding minimal ML expertise. Meanwhile, NVMensure effectively detects any modifications made to the NVMe offloaded data, offering protection against potential future targeted poisoning attacks that may emerge

Frameworks powering LSML. DeepSpeed [48] has emerged as the leading LSML framework, thanks to its unique NVMe offload feature enabling training of giant models. Other existing projects and frameworks for LSML focus on parallelism strategies to distribute data and computation across hardware instances [24] but lack the NVMe offload feature. Many of these projects, such as NVIDIA's Megatron-LM [53] for model, PipeDream [27] for pipeline, and ZeRO [44] for data parallelism, have been integrated into DeepSpeed.

Dataset Cleaning Methods. Benign models require poison-free datasets, leading to the use of filtering-based approaches for reactive defense against data poisoning attacks.[7] Outlier detection in the input space [35] is one such method, but also classification algorithms can be used to filter malicious samples [36]. For additional information, we direct the reader to [14].

Security of CPU and RAM. Several frameworks [39,58,23,32] secure the CPU-based training process by using a Trusted Execution Environment (TEE) like Intel SGX [10]. Another proposed solution is Perun [33], which relies on TMP [21] to attest the entire OS, ensuring trust also for accelerators. Albeit those approaches introduce overhead, advancements in CPU power and TEE memory capacity in next-generation hardware can address this bottleneck.

Secure execution on GPUs. Several works have addressed secure ML computation offloading to GPUs. Slalom [57], enables outsourcing of linear model layer computation from a TEE to GPUs. Graviton [59], proposed for single-GPU setups, extends TEE security guarantees to GPUs, although some technical effort is needed to support multiple GPUs. HETEE [64] introduces a container running on separate trusted hardware to control access to untrusted accelerators, while HIX [20] modifies the GPU driver within a TEE to ensure trusted code usage.

Overall, the aforementioned methods are orthogonal to NVMensure and can be combined to create a comprehensive security architecture that protects all components of an LSML system. This technical effort exceeds the scope of this work but is possible for real-world scenarios.

6 Conclusion

In this paper, we analyzed the security of large-scale machine learning (LSML) and recent advancements in securing distributed ML training. While security frameworks for CPU and GPU training exist, the NVMe offload mechanism introduced in DeepSpeed, which enables efficient GPU memory utilization and training of billion parameter models, lacked attention in terms of security.

In this work, we are the first to examine the security of NVMe offloads pointing out the vulnerable to untargeted poisoning attacks. To demonstrate this weakness, we propose NVMevade, which uses three distinct methods to exploit NVMe offloads and negatively affect model performance and training speed. To close the vulnerability, we propose NVMensure, the first defense against poisoning attack, targeted or untargeted, on NVMe offloads in LSML. We implement attack and defense for the DeepSpeed library, demonstrate the NVMensure's effectiveness, and explore security-performance trade-offs one can achieve when opting for various integrity protection methods and data structures.

[7] Since LSML models are normally trained on publicly available data that can be scrutinized by experts, dataset privacy is not a concern.

Acknowledgment

We thank the Private AI Collaborate Research Institute which is co-sponsored by Intel Labs(www.private-ai.org) for partially supporting this research.

References

1. Aumasson, J.P., Neves, S., Wilcox-O'Hearn, Z., Winnerlein, C.: BLAKE2: Simpler, Smaller, Fast as MD5. ACNS (2013)
2. Bagdasaryan, E., Shmatikov, V.: Blind Backdoors in Deep Learning Models. USENIX Security (2021)
3. Bloom, B.H.: Space/Time Trade-Offs in Hash Coding with Allowable Errors. Commun. ACM (1970)
4. Bojar, et al.: Findings of the 2016 Conference on Machine Translation. Proceedings of the First Conference on Machine Translation (2016)
5. Brown, et al.: Language Models are Few-Shot Learners. NeurIPS (2020)
6. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M.T., Zhang, Y.: Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv preprint arXiv:2303.12712 (2023)
7. Chen, H., Fu, C., Zhao, J., Koushanfar, F.: ProFlip: Targeted Trojan Attack with Progressive Bit Flips. IEEE/CVF ICCV (2021)
8. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. arXiv preprint arXiv:1712.05526 (2017)
9. Collobert, R., Weston, J.: A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. ICML (2008)
10. Costan, V., Devadas, S.: Intel SGX explained. Cryptology ePrint Archive (2016)
11. El Merabet, H., Hajraoui, A.: A Survey of Malware Detection Techniques based on Machine Learning. IJACSA (2019)
12. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo Filter: Practically Better Than Bloom. CoNEXT (2014)
13. Gallagher, P., Director, A.: Secure Hash Standard (shs). FIPS PUB (1995)
14. Goldblum, M., Tsipras, D., Xie, C., Chen, X., Schwarzschild, A., Song, D., Mądry, A., Li, B., Goldstein, T.: Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses. IEEE PAMI (2022)
15. Graves, A., Mohamed, A.r., Hinton, G.: Speech Recognition with Deep Recurrent Neural Networks. ICASSP (2013)
16. Guo, D., Liu, Y., Li, X., Yang, P.: False Negative Problem of Counting Bloom Filter. IEEE Transactions on Knowledge and Data Engineering (2010)
17. Hilal, W., Gadsden, S.A., Yawney, J.: Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances. Expert Syst. Appl. (2022)
18. International Organization for Standardization: Information processing — Use of longitudinal parity to detect errors in information messages. ISO Standard ISO 1155, ISO (2001)
19. Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., Li, B.: Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. IEEE S&P (2018)
20. Jang, I., Tang, A., Kim, T., Sethumadhavan, S., Huh, J.: Heterogeneous Isolated Execution for Commodity GPUs. ASPLOS (2019)

21. Kinney, S.L.: Trusted Platform Module Basics: Using TPM in Embedded Systems. Elsevier (2006)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS (2017)
23. Le Quoc, D., Gregor, F., Singh, J., Fetzer, C.: SGX-PySpark: Secure Distributed Data Analytics. WWW (2019)
24. Mechanics, M.: What runs ChatGPT? Inside Microsoft's AI supercomputer | Featuring Mark Russinovich. https://youtu.be/Rk3nTUfRZmo (2023)
25. Microsoft Research: Turing NLG: A 17 Billion Parameter Language Model by Microsoft. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/ (2021)
26. Mutlu, O., Kim, J.S.: RowHammer: A Retrospective. IEEE TCAD (2020)
27. Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N., Granger, G., Gibbons, P., Zaharia, M.: PipeDream: Generalized Pipeline Parallelism for DNN Training. ACM SOSP (2019)
28. Nvidia: A100 GPU. https://www.nvidia.com/en-us/data-center/a100/ (2023)
29. Nvidia: DGX Systems. https://www.nvidia.com/de-de/data-center/dgx-systems/ (2023)
30. OpenAI: Chatgpt. https://openai.com/research/chatgpt (2023)
31. OpenAI: GPT-4 Technical Report. arXiv preprint arXiv:2303.08774 (2023)
32. Orenbach, M., Lifshits, P., Minkin, M., Silberstein, M.: Eleos: ExitLess OS Services for SGX Enclaves. EuroSys (2017)
33. Ozga, W., Quoc, D.L., Fetzer, C.: Perun: Secure Multi-Stakeholder Machine Learning Framework with GPU Support. arXiv preprint arXiv:2103.16898 (2021)
34. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS (2019)
35. Paudice, A., Muñoz-González, L., Gyorgy, A., Lupu, E.C.: Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection. arXiv preprint arXiv:1802.03041 (2018)
36. Peri, N., Gupta, N., Huang, W.R., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., Dickerson, J.P.: Deep k-NN Defense against Clean-label Data Poisoning Attacks. ECCV (2020)
37. Peterson, W.W., Brown, D.T.: Cyclic Codes for Error Detection. Proceedings of the IRE (1961)
38. Post, M.: A Call for Clarity in Reporting BLEU Scores. In: Proceedings of the Third Conference on Machine Translation: Research Papers (2018)
39. Quoc, D.L., Gregor, F., Arnautov, S., Kunkel, R., Bhatotia, P., Fetzer, C.: SecureTF: A Secure TensorFlow Framework. ACM/IFIP Middleware (2020)
40. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving Language Understanding by Generative Pre-Training. OpenAI (2018)
41. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language Models are Unsupervised Multitask Learners. OpenAI blog (2019)
42. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. JMLR (2020)
43. Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R.Y., Awan, A.A., Rasley, J., He, Y.: DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. arXiv preprint arXiv:2201.05596 (2022)
44. Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: ZeRO: Memory Optimizations toward Training Trillion Parameter Models. SC20 (2020)

45. Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., He, Y.: ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. arXiv preprint arXiv:2104.07857 (2021)
46. Rakin, A.S., He, Z., Fan, D.: TBT: Targeted Neural Network Attack with Bit Trojan. IEEE/CVF CVPR (2020)
47. Rakin, A.S., He, Z., Li, J., Yao, F., Chakrabarti, C., Fan, D.: T-BFA: Targeted Bit-Flip Adversarial Weight Attack. IEEE PAMI (2022)
48. Rasley, J., Rajbhandari, S., Ruwase, O., He, Y.: DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. SIGKDD (2020)
49. Ren, J., Rajbhandari, S., Aminabadi, R.Y., Ruwase, O., Yang, S., Zhang, M., Li, D., He, Y.: ZeRO-Offload: Democratizing Billion-Scale Model Training. USENIX ATC (2021)
50. Rivest, R.: The MD5 Message-Digest Algorithm. IETF (1992)
51. Saha, A., Subramanya, A., Pirsiavash, H.: Hidden Trigger Backdoor Attacks. AAAI (2020)
52. Sarwate, D.V.: Computation of Cyclic Redundancy Checks via Table Look-Up. Commun. ACM (1988)
53. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv preprint arXiv:1909.08053 (2020)
54. Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., Zhang, E., Child, R., Aminabadi, R.Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., Catanzaro, B.: Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model (2022)
55. Tian, Z., Cui, L., Liang, J., Yu, S.: A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning. ACM Comput. Surv. (2022)
56. Tian, Z., Cui, L., Liang, J., Yu, S.: A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning. ACM CSUR (2022)
57. Tramèr, F., Boneh, D.: Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. ICLR (2018)
58. Tsai, C.C., Porter, D.E., Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. USENIX ATC (2017)
59. Volos, S., Vaswani, K., Bruno, R.: Graviton: Trusted Execution Environments on GPUs. USENIX OSDI (2018)
60. Xia, G., Chen, J., Yu, C., Ma, J.: Poisoning Attacks in Federated Learning: A Survey. IEEE Access (2023)
61. Xiao, H., Biggio, B., Brown, G., Fumera, G., Eckert, C., Roli, F.: Is Feature Selection Secure against Training Data Poisoning? PMLR (2015)
62. Yang, C., Wu, Q., Li, H., Chen, Y.: Generative Poisoning Attack Method Against Neural Networks. arXiv preprint arXiv:1703.01340 (2017)
63. Yao, F., Rakin, A.S., Fan, D.: DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. USENIX Security (2020)
64. Zhu, J., Hou, R., Wang, X., Wang, W., Cao, J., Zhao, B., Wang, Z., Zhang, Y., Ying, J., Zhang, L., et al.: Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. IEEE S&P (2020)

A Instantiation for DeepSpeed

Below, we provide implementation details of our approaches for DeepSpeed [48].

A.1 NVMevade

Tab. 6 depicts the parameters of NVMevade for attack modes 1 and 2. Further, we want to report, that our scanning process confirmed that DeepSpeed offloads three types of data: model parameters, optimizer states, and gradients. Following, we provide additional information for the different attack modes.

Table 6. NVMevade's parameters for bit-flips in attack mode 1 and 2.

NVMevade's Parameters		Mode 1	Mode 2
P_1:	Percent of poisoned bytes within each offloaded file	100%	0.0016%
P_2:	Percent of poisonings within lower bytes of float16 parameters	100%	100%
P_3:	Percent of poisonings within upper bytes of float16 parameters	100%	25%
P_4:	Threshold of poisonings for lower bytes in percentage of the file	100%	0.0002%
P_5:	Threshold of poisonings for upper bytes in percentage of the file	100%	0.0027%
P_6:	Continuous poisoning time window	100%	100%

Mode 1 and 2. During the poisoning process via bit-flips, we simultaneously attacked all offloaded data. The most impactful perturbations were made in intermediate training parameters, namely gradients and optimizer states, leading to immediate gradient overflows. In benign training, DeepSpeed scales gradients using a scaling factor to address gradient underflows. However, in NVMevade's implanted overflow scenario, the current training step is skipped, the scaling factor is halved, and the training is halted. When the scaling factor reaches a configured minimum, the entire DeepSpeed process is interrupted with an exception, typically occurring within a few steps of malicious overflows. When adjusting the parameters for mode 2, the objective is to minimize the occurrence of overflows while ensuring a significant level of detrimental modifications, thereby preventing the training process from abruptly terminating.

Mode 3. Due to DeepSpeed's implementation of a basic sanity check on file sizes, adjusting file size might be necessary if a newly offloaded file with the same name but different size appears during replay attacks. The adjustment can be done via Python's truncate function to reduce size or zero-padding to increase it. Yet, our experiments didn't encounter this scenario.

A.2 NVMensure

NVMensure is implemented in C++ within the DeepNVMe [45] library. The runtime of integrity mechanisms varies depending on the implementation. MD5 and BLAKE2bp employ C++ reference implementations, while SHA-256 utilizes a highly efficient kernel implementation accessed through the kernel's crypto API. We implemented CRC-32 [52] and LRC [18] by ourselves in C++.