ITC 2/55
Information Technology and Control
Vol. 55 / No. 2/ 2026
pp. 469-488
DOI 10.5755/j01.itc.55.2.43582

AdapFuzzer: An Adaptive Fuzzing Framework for DNNS with Diversity-Aware Seed Selection and Gan-Based Mutation

Received 2025/11/12 | Accepted after revision 2026/03/19

HOW TO CITE: Dan, N., Admodisastro, N., Murad, A. A. M., Osman, H. M. (2026) AdapFuzzer: An Adaptive Fuzzing Framework for DNNS with Diversity-Aware Seed Selection and Gan-Based Mutation. *Information Technology and Control*, 55(2), 469-488. https://doi.org/10.5755/j01.itc.55.2.43582

AdapFuzzer: An Adaptive Fuzzing Framework for DNNS with Diversity-Aware Seed Selection and Gan-Based Mutation

Ningyun Dan

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia

College of Physics and Information Engineering, Zhaotong University, Zhaotong, 657000, China.

Novia Admodisastro*, Masrah Azrifah Azmi Murad, Mohd Hafeez Osman

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia; emails: GS65715@student.upm.edu.my; masrah@upm.edu.my; hafeez@upm.edu.my

Corresponding author: novia@upm.edu.my

Deep learning systems, built on data-driven learning paradigms, are highly sensitive to input perturbations, where even slight variations can lead to substantial output deviations or incorrect decisions, posing serious reliability and safety concerns. To address these challenges, we propose AdapFuzzer, an adaptive fuzzing framework that incorporates feedback-driven control principles into the testing process. AdapFuzzer consists of three core components: (1) NFuzzer, a diversity-driven seed selection module that iteratively constructs a representative seed pool using deep feature-based dissimilarity measurement; (2) FAGAN, a mutation engine that leverages generative adversarial learning to produce high-quality and diverse adversarial variants; and (3) a test management module that continuously analyzes coverage and fault-triggering feedback to refine both seed prioritization and mutation strategies. Through this closed-loop optimization, AdapFuzzer dynamically adapts its testing behavior to the evolving detection capability of seeds and mutation operators. Experimental results on multiple deep learning models demonstrate that AdapFuzzer significantly improves testing effectiveness, achieving higher coverage and uncovering more erroneous behaviors than state-of-the-art fuzzing approaches, thereby enhancing the robustness and security of deep learning systems.

KEYWORDS: Fuzz testing, deep learning, Deep Learning Security, Deep Learning Systems, Adversarial examples

1. Introduction

With the widespread application of deep learning technology across various domains [25], from autonomous driving to medical diagnosis and natural language processing, these models Have permeated every corner of modern society. However, current deep learning fuzzing techniques face two key bottlenecks that limit testing efficiency: (1) insufficient seed diversity, which leads to inadequate exploration of the model's state space; and (2) the lack of adaptive mutation strategies, which cannot dynamically adjust based on real-time feedback during testing, resulting in resource waste. This paper aims to address both issues through a unified closed-loop feedback framework. Therefore, ensuring that these models operate stably and accurately is of paramount importance.

Fuzzing, an effective software security evaluation technique, involves injecting large amounts of random or semi-random data into a system to probe its boundary conditions and abnormal behaviors [2]. It has been proven to be an effective method for identifying potential defects [15]. Applying fuzzing to deep learning can help researchers identify unforeseen yet critical vulnerabilities and promote the development of more robust and reliable artificial intelligence systems. Existing studies have proposed and validated multiple fuzzing techniques for deep learning models, demonstrating their effectiveness in revealing faults (i.e., generating adversarial examples) and exploring the internal states of the models. Typically, these techniques use a random selection method to choose a subset of test cases as the seed queue. While this approach is simple and easy to implement, it often fails to ensure the diversity of the seed queue, limiting the effectiveness of fuzzing.

In fact, the diversity of the seed queue is crucial for enhancing the performance of fuzzing. Theoretically, a diverse seed queue helps to uncover more potential issues and thoroughly explore the state space of the model. When applied to deep learning models, such a queue, after mutation, can activate more neurons, providing a more comprehensive examination of the rules learned by the model. In the context of traditional software fuzzing, researchers have attempted to create highly diverse seed queues using distance concepts. However, for deep learning, most metrics for measuring differences are applicable only to basic data types (e.g., numbers, strings). In contrast, fuzzing strategies for deep learning models are mainly categorized into four types: Random selection, recency-aware, frequency-aware, and clustering-based. Random selection ignores all available information, potentially reducing fault detection efficiency [26]. Recency-aware and frequency-aware methods rely on single-dimensional information (such as the time a seed was added to the queue or the number of times it was used), failing to utilize other valuable data (e.g., fault exposure information). Although clustering algorithms can be used to classify seeds and then select from the clusters, their effectiveness remains to be further validated.

Moreover, the choice of mutation strategy is another key factor affecting the effectiveness and efficiency of fuzzing. Current research often focuses on designing new mutation schemes, with less attention paid to how to select the optimal strategy for specific seeds. Studies by Tian et al. have shown that different mutation strategies tend to activate different neurons, indicating that there is no one-size-fits-all best practice. Nevertheless, common practices still rely on random or greedy selection, making it difficult to find the most suitable mutation path for each case [21].

To address these issues, we propose AdapFuzzer, an adaptive fuzzing framework that incorporates a closed-loop feedback mechanism to jointly optimize seed diversity and mutation strategy. The main contributions of this work are threefold.

1. We propose a novel fuzz testing method, AdapFuzzer, which features an innovative seed selection module and seed mutation strategies.
2. Compared to existing methods, AdapFuzzer achieves significantly better results. In contrast to common testing techniques, AdapFuzzer generates more adversarial inputs, achieves higher neuron coverage, and exhibits a stronger capability to explore the internal states of the model.
3. We conducted a series of ablation studies to analyze the effectiveness of the different components in our proposed method. These studies validated the contribution of each component to the overall performance, further demonstrating the advantages of AdapFuzzer.

2. Related Work

2.1. Fuzz Testing

Fuzz testing techniques for deep learning systems have garnered significant attention in recent years [3], as these methods aim to systematically uncover potential vulnerabilities and errors within neural networks [10]. One pioneering work is Deepxplore, which introduces an automated whitebox testing framework that employs the multiple input cross-coverage (micc) criterion to generate test cases capable of triggering different combinations of neurons. This approach not only detects logical errors within the model but also identifies inconsistencies by comparing differences among multiple DNNS without requiring access to the original training data or labels [19]. This provides an effective means for evaluating and enhancing the security of deep learning models. Subsequent research has further expanded this field. For instance, DLFuzz [24] proposes a differential fuzzing method based on genetic algorithms, aiming to maximize neuron coverage while minimizing misclassification rates, thereby effectively finding adversarial samples that may cause model misclassification. Through this approach, DLFuzz can maintain efficiency while ensuring the quality of generated adversarial samples. Tensorfuzz [17] brings a novel method that combines traditional software fuzzing with neural network analysis. It generates new test inputs using mutation strategies and ensures that these inputs retain certain semantic characteristics of the original data. This method is particularly suitable for application scenarios where input data must adhere to specific formats, such as image processing tasks. Furthermore, Deephunter [18] offers a more general framework for guided fuzz testing. It integrates various seed selection and mutation strategies to comprehensively explore the behavior boundaries of deep learning models. By adopting a metric known as "Neuron-level coverage," Deephunter can more precisely control the testing process and increase the probability of discovering new types of errors.recent studies have also begun to focus on integrating fuzz testing with other security mechanisms, such as formal verification methods or static analysis techniques, to form a more comprehensive security assurance system. Tensorfuzz [6] focuses on designing fuzzing methods for deep learning systems specifically to locate inputs that cause unexpected behaviors. Unlike traditional software fuzzing, Tensorfuzz is particularly suited for continuous input spaces and can handle deep learning models lacking clear logic or control flow.

A systematic comparison of the core mechanisms among existing DNN fuzzing techniques is instrumental in understanding the evolution of the field and in highlighting the technical contributions of this work. To this end, we summarize and analyze representative approaches from five key dimensions: seed selection strategy, mutation generation method, feedback mechanism, coverage criterion, and primary objective, as presented in Table 1.

The comparative analysis in Table 1 reveals that while prior works have advanced mutation methods and coverage criteria, limitations persist in their seed selection strategies – often reliant on randomness or simple heuristics –and their feedback mechanisms – typically unidirectional or static. In contrast, the technical trajectory of AdapFuzzer is distinguished by three fundamental advancements:

Seed Selection: Moving beyond random or single-metric approaches, NFuzzer introduces a systematic, diversity-driven strategy underpinned by a deep difference metric (DMSNet/DMTNet). This aims to fundamentally address the lack of structured exploration of the input space.

Mutation Generation: Replacing discrete, preset mutation operators or single-gradient perturbations, FAGAN leverages the generative capacity of GANs to produce more coherent, imperceptible, and diverse adversarial variants.

Feedback & Adaptation: This constitutes the core differentiator of our work. Unlike the often simple or linear feedback loops in existing frameworks (e.g., updating seed priorities based solely on coverage), AdapFuzzer incorporates a Monitoring and Control Module. This module establishes a multi-level, closed-loop adaptive control system encompassing seed scheduling, mutation strategy tuning, and execution feedback analysis. Consequently, the entire fuzzing campaign can dynamically refine its focus based on real-time feedback.

Therefore, the paradigm shift offered by AdapFuzzer can be summarized as a transition from depen-

Table 1
Comparison of core mechanisms in DNN fuzzing techniques.

Method	Seed Selection Strategy	Mutation Generation Method	Feedback Mechanism	Coverage Criterion	Primary Objective
DeepXplore [18]	Random selection	Gradient-guided pixel-level perturbation	Neuron activation states	Neuron Coverage (NC)	Generate divergent inputs to explore neuron combinations
DLFuzz [6]	Fitness-based selection (Genetic Algorithm)	Gradient-based + Genetic Algorithm mutations	Neuron coverage & prediction change	Neuron Coverage (NC)	Maximize coverage while finding misclassifications
Tensorfuzz [17]	Coverage-guided random selection	Random application of traditional mutation operators	Coverage of new code/states (for APIs)	Custom (e.g., code coverage)	Locate inputs causing unexpected behaviors in continuous spaces
Deephunter [18]	Multiple strategies (e.g., round-robin, random)	Predefined mutation operators (e.g., noise, blur)	Multi-granularity coverage feedback	Neuron-level Coverage	General framework for comprehensive behavioral exploration
AdapFuzzer (Ours)	NFuzzer: Diversity-driven selection based on deep feature dissimilarity	FAGAN: Guided mutation via Generative Adversarial Networks	Adaptive Control Module: Multi-level (seed, mutation, execution) closed-loop feedback	Neuron Coverage (NC)	High-efficiency, diverse fault discovery via closed-loop adaptive optimization

dency on static or stochastic strategies to systematic seed selection based on deep dissimilarity metrics; from the application of discrete, predefined mutation operators to directed, high-quality mutation via generative models; and most critically, from open-loop or simplistic feedback processes to an intelligent, adaptive closed-loop control system capable of multi-level decision-making. This holistic design strives to achieve more efficient and exhaustive exploration of the state space of deep learning models.

2.2. Adversarial Examples

Adversarial examples refer to the phenomenon where machine learning models are made to produce incorrect predictions by applying small but carefully crafted perturbations to the input data. This concept was First introduced in 2013 by Goodfellow et al., who demonstrated a simple and effective method for generating adversarial examples using the fast gradient sign method (FGSM) [22]. Since then, research on adversarial examples has rapidly evolved, encompassing their generation, detection, and defense [1]. In terms of generation, techniques such as iterative FGSM, basic iterative method (bim) [17], projected gradient descent (PGD) [14], and the carlini-wagner [1] attack have been developed to create more deceptive adversarial examples.

3. AdapFuzzer

3.1. Overview of AdapFuzzer

Based on the fundamental principles of control theory and the fuzzing process, this paper introduces an adaptive fuzzing framework, AdapFuzzer. As illustrated in the Figure 1, AdapFuzzer is a system composed of Four main Modules: The fuzzing process module (which includes seeds, mutation strategies, test cases, and test results), the storage module (used for storing testing process information such as test result information and seed coverage information), and the monitoring module (which includes seed selection algorithms and mutation strategy selection algorithms). The testing process information not only guides the controller in selecting seeds and mutation strategies but also continuously improves the controller's ability to adapt to the evolving fault-revealing capabilities of the seeds and mutation strategies.

AdapFuzzer primarily consists of the following four key modules:

1. **Seed selection module:** This module selects the optimal seed based on the priority of the seeds.
2. **FAGAN module:** Utilizing FAGAN-based mutation strategies, this module guides the mutation of seeds to generate a large number of test cases.
3. **Test management module:** This module monitors the generated test cases and records the execution information, including test coverage and seed information, and monitors, trains, and adjusts FAGAN. The architecture of AdapFuzzer is conceptually divided into two interconnected parts: the core execution workflow and the adaptive control loop, illustrated in Figures 1-2, respectively.
4. **Monitoring and Control Module:** This module serves as the "brain" of the adaptive loop. It analyzes all recorded information from the Test Management Module, and then dynamically adjusts the strategies of the Seed Selection and FAGAN Mutation modules. This feedback loop enables AdapFuzzer to continuously self-optimize based on the evolving testing state.

The architecture of AdapFuzzer, driven by these four modules, is conceptually divided into two interconnected parts: the core execution workflow (the forward path of generating and running tests) and the adaptive control loop (the feedback path that optimizes the process), illustrated in Figures 1-2, respectively. Figure 1 outlines the forward path of the fuzzing process. It begins with the NFuzzer component selecting the most promising seed from the queue. This seed is then mutated by the FAGAN engine to generate a batch of adversarial test cases. These tests are executed against the target DNN, and their raw outcomes (e.g., coverage, classification results) are logged to the Storage Module. Figure 2 details the backward adaptive loop that enables continuous optimization. The Management Core analyzes the aggregated data from the Storage Module. Based on this analysis, it dispatches refined strategies to the controllers for seed selection (guiding NFuzzer), mutation (tuning FAGAN), and test evaluation. This (closed-loop) feedback ensures that the fuzzing process intelligently adapts over time, focusing its efforts on the most effective seeds and mutation strategies.

AdapFuzzer records all valuable test execution information enabling adaptive optimization of both the seed selection strategy and mutation strategy.

Figure 1

The core execution workflow of AdapFuzzer.

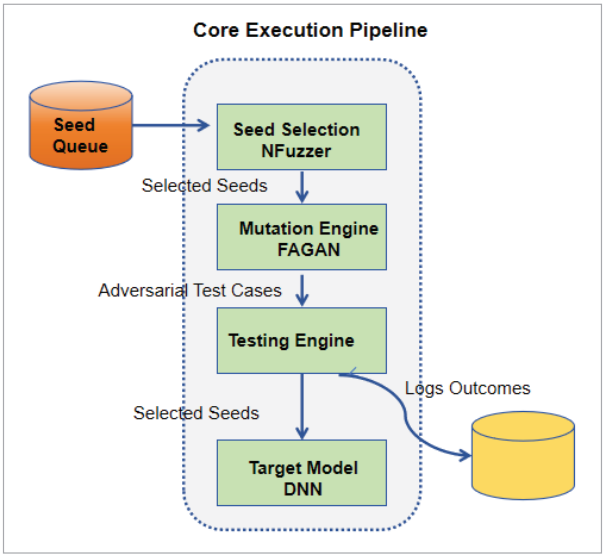
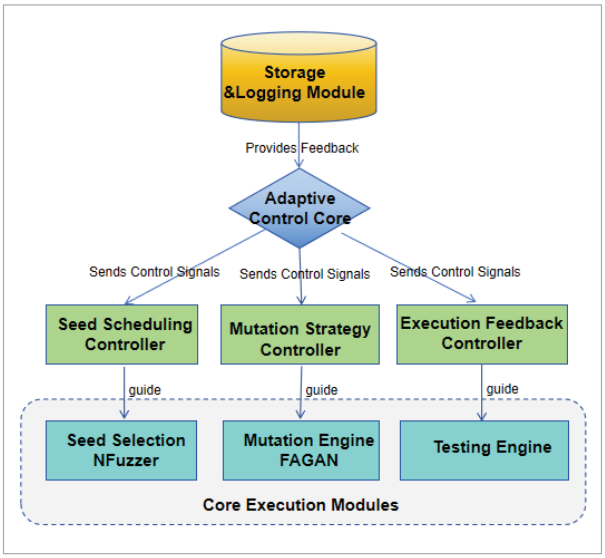


Figure 2

The adaptive feedback and control loop of AdapFuzzer.



To clearly describe how the three core modules cooperate during the fuzzing process, we summarize the overall workflow of AdapFuzzer in Algorithm 1. The algorithm 1 llustrates the iterative testing loop, where NFuzzer selects diverse and high-priority seeds, FAGAN generates mutated test inputs, and the test management module evaluates coverage feedback and updates seed priorities. This closed-loop interaction enables AdapFuzzer to continuously adapt its seed selection and mutation strategies throughout the testing process.

Algorithm 1 Testing phase of AdapFuzzer.

Input: D: Datasets of testing seeds;
Dnn: Deep neural network;
M: Maximum number of iterations;
N: Number of new examples generated;
P: FAGAN parameters;
K: Top-k parameter
Output: Test example set for increasing coverage train FAGAN;
T = divide(d);
while number of iterations <M do
S= SeedSelect (T) ;
Instance = Sample (S);
while number of generations <N do
data = FAGan(Instance);
Ip, Id=InformationExtraction(Instance, data);
Sim=compare(Ip, Id) ;
end
Select top- k examples;
while number of calculations <K do
Cover = DNNFeed(data);
if isNewCover(Cover) then
Add data to the processing pool;
Set selection priority for data;
end
end
end
Output examples set;

3.2. NFuzzer Component

3.2.1. Algorithm Description

The NFuzzer module is the core of the adaptive seed selection strategy in AdapFuzzer. Its primary function is to construct and maintain a diversity-aware seed queue. Since deep learning models typically come with a test dataset, obtaining an initial test case set is straightforward. NFuzzer operates on a simple yet effective principle: to iteratively select the seed that is most dissimilar to those already in the queue.

To achieve this efficiently, NFuzzer maintains two disjoint sets: a candidate seed set and the seed queue itself. The algorithm, detailed in Algorithm 2, starts by initializing as empty. Seeds are then incrementally added to from until a termination condition (e.g., |L|=n) is met. In each iteration, a candidate set S of size k is randomly sampled from. The key step is to identify the seed in that has the largest minimum distance to any seed in, which ensures it is the most diverse addition to the queue. The distance between seeds is quantified by a metric, which in our implementation is provided by the DMSNet or DMTNet models, leveraging their ability to perceive perceptual differences between inputs.

Algorithm 2 NFuzzer algorithm for building a diverse seed queue.

Input: Test case set t, seed queue size n, candidate seed set size k, distance metric d
Output: Constructed seed queue l
Initialize seed queue l as empty;
Initialize candidate seed set s as empty; while |l| < n do
while|L|<ndo
if|L|==0then
Select a test case ts0 from T and add it to L ;
end
for i ← 0$ tok-1do
Select a test case t s1 from T and add it to S ;
end
top_item ← S[0];
top_dist ← 0;
foreach si ∈ S do
bot_item ← S[0];
bot_dist ← +∞;
foreach l j ∈ Ldo
if bot_dist >D(s i, l j) then
bot_item ← si;
bot_dist ← D(s i, l j);
end
end
if top_dist < bot_dist then
top_item ← bot_item;
top_dist ← bot_dist;
end
end
Add top_item to L;
Clear S;
end
return L;

The distance between seeds is quantified by a metric D, which in our implementation is provided by the DMSNet or DMTNet models. These models are trained to map inputs into a feature space where a triplet loss function is employed. The triplet [7] loss aims to enhance feature diversity by structuring the feature space such that samples with the same label are brought closer together, while those with different labels are pushed farther apart. Specifically, for an anchor sample, a positive sample (same class), and a negative sample (different class), the loss function requires that the distance between the anchor and the negative sample exceeds the distance between the anchor and the positive sample by at least a margin α. This constraint not only ensures clear class separation but also promotes a broad distribution of features within the same class, which directly fulfills our requirement for a seed difference metric that prioritizes diversity. The objective function is formulated

$$\sum_{i=1}^{N}\left[\| f(x_i^a) - f(x_i^p)\|_2^2 - \| f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+. \quad (1)$$

where $\|*\|_2$ denotes the Euclidean distance, so $\| f(x_i^a) - f(x_i^n)\|_2^2$ represents the Euclidean distance measure between the Negative and Anchor points.

Leveraging this property, NFuzzer effectively selects dissimilar seeds. The complete procedure is formalized in Algorithm 2

3.2.2. Difference Metric Models: DMSNet and DMTNet

The NFuzzer component relies on accurate distance metrics to quantify perceptual differences between inputs, which is essential for its diversity-driven seed selection. This functionality is provided by two custom-designed difference metric models: the Deep Morphological Structure Network (DMSNet) and the Deep Morphological Texture Network (DMTNet).

Model Architecture. Both DMSNet and DMTNet leverage aResNet-18 backbone, pre-trained on ImageNet, as their foundational feature extractor. To tailor the network for difference measurement rather than classification, we remove the original 1000-class fully-connected classification layer. The final 2048-dimensional feature vector from theResNet backbone is then passed through an additional fully-connected layer, which projects the features into a lower-dimensional embedding space (typically 128-dimensional). This refined architecture enables the models to learn feature representations that are sensitive to inter-input differences specific to either structural or textural aspects.

Training Data and Independence. To ensure the generality of the learned metric and prevent bias towards any specific target model under test, both models are trained on a subset of the publicly available ImageNet dataset, which is entirely independent of all target DNNS used in the fuzzing campaign. ImageNet provides a rich and diverse visual corpus spanning 1000 categories, facilitating the learning of a robust and generalizable feature space. During training, we employ the triplet loss function (see Equation 1), which learns a discriminative embedding by pulling an anchor sample closer to a positive sample (same class) than to a negative sample (different class) by a defined margin. Crucially, once trained, the parameters of both DMSNet and DMTNet are frozen and remain static throughout the entire fuzzing process. This guarantees a stable and consistent distance metric D, a critical requirement for the reproducible and reliable operation of NFuzzer.

Model Characteristics. While sharing the same architectural backbone, DMSNet and DMTNet are specialized to capture complementary aspects of input differences: DMSNet is optimized to be sensitive to structural and morphological variations, whereas DMTNet focuses on textural and pattern-based distinctions. Within NFuzzer, these models can be employed individually or in tandem, offering a flexible, multi-perspective assessment of seed diversity for more intelligent queue construction.

3.2.3. Computational Complexity Implementation

The computational complexity of NFuzzer is primarily determined by the number of candidate samples (k), the target queue size (n), and the cost of computing the neural feature distance D.

For each iteration, NFuzzer samples k candidates from the remaining test cases and evaluates their distances to all seeds in the current queue L. Let the cost of computing one feature representation and distance be $C_{feat} + C_D$, then the total time complexity is $O(n \times k \times x(C_{feat} + C_D))$.

In practice, the computation of neural feature embeddings through DMSNet or DMTNet dominates the runtime.

To improve efficiency, feature vectors can be precomputed and cached, and approximate nearest-neighbor (ANN) search can be employed for large datasets.In implementation, candidates are sampled from (T\L) without replacement. If multiple candidates achieve the same maximal minimal distance, one is randomly chosen to maintain diversity. The initial seed can be selected randomly or through heuristics such as heuristics such as maximal predictive uncertainty". All random operations are conducted under fixed random seeds to ensure reproducibility. The neural distance metric D can be replaced or extended with other feature extractors depending on the target model architecture.

3.2.4. Module Workflow

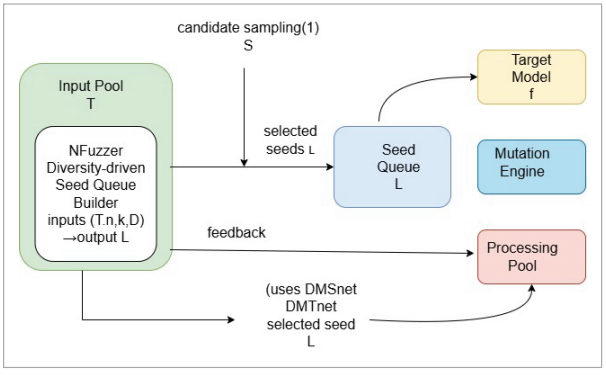
Figure 3 illustrates the internal workflow of the seed selection module and its interaction with other components of AdapFuzzer. NFuzzer receives the initial test case pool T and parameters (n, k, D), samples candidate inputs S, and iteratively selects the most diverse subset to form the seed queue L. This queue is subsequently passed to the mutation engine. Meanwhile, execution feedback (e.g., coverage or fault signals) is sent back to NFuzzer to refine its subsequent selection, enabling adaptive and feedback-driven fuzzing.

3.3. Mutation Engine FAGAN

The Fuzzing-oriented Adversarial Generative Network (FAGAN) serves as the core mutation engine in our design [8]. Distinct from generic adversarial example generators (e.g., AdvGAN [23]), which are primarily designed to craft imperceptible perturbations for a single attack objective, FAGAN is architected from the ground up as a fuzzing-oriented mutation engine. Its core design philosophy diverges in three key aspects: (1) its primary goal is to support long-term, diverse fault discovery within an adaptive testing loop, rather than to achieve a one-time high attack success rate; (2) it is tightly integrated as an actuator within a larger control system, allowing its generation strategy to be dynamically tuned; and (3) it balances the triple objectives of attack effectiveness, output diversity, and semantic realism specifically to serve coverage-guided fuzzing. While its architecture is grounded in the Generative Adversarial Network (GAN) framework, it has been specifically optimized for fuzzing tasks. The system is primarily composed of three core components that form a collaborative adversarial attack framework:

Figure 3

NFuzzer: Diversity-driven seed selection workflow.



Generator (G): Responsible for generating imperceptible perturbations. It takes an original sample x as input and outputs a perturbation vector G(x). The objective of the generator is to create perturbations that can deceive the target model f, while ensuring that remains highly similar to x in both visual and semantic characteristics.

Discriminator (D): Tasked with distinguishing between original samples and mutated samples produced by the generator. Its input is,and its goal is to accurately determine whether the input represents real data. Through this adversarial training process between the generator and discriminator, FAGAN can produce high-quality, highly deceptive mutation seeds.

Target Neural Network (f): Represents the deep learning model under test. This component does not participate in the training process of FAGAN but instead acts as an "adjudicator". The performance of the generator G is evaluated based on the effectiveness of its output mutated samples in deceiving the target model f, as measured by the adversarial loss.

The crucial distinction between FAGAN and traditional GANs lies in the incorporation of the target model f for computing adversarial loss. This design ensures that the generator's optimization objectives extend beyond merely deceiving the discrim-

inator D to directly targeting vulnerabilities in the tested model ff, thereby generating test cases that can effectively trigger erroneous behaviors (such as misclassification). Furthermore, we introduce triplet loss during training to ensure that the generated mutation seeds possess sufficient diversity in the feature space. This approach synergizes with NFuzzer's diverse seed selection strategy, collectively enhancing test coverage. the architecture of FAGAN is shown in Figure 4.

The adversarial loss function for the GAN component is defined as the standard min-max objective:

$$L_{GAN} = E_x[\log D(x)] + E_z[\log(1 - D(G(x)))]. \quad (2)$$

The adversarial loss L_{adv} is designed to generate effective test cases that reveal model vulnerabilities. Depending on the attack scenario:

For targeted attacks (when a specific misclassification target is desired):

$$L_{adv} = E_x\,[\,l(f(x + G(x)), t)\,], \quad (3)$$

where t is the target class, and $l(\cdot,\cdot)$ is the loss function used to train f (e.g., cross-entropy loss). This loss encourages the perturbed image $x+G(x)$ to be misclassified into the target class t.

For non-targeted attacks (when any misclassification or significant confidence drop is sufficient, as is typical in fuzzing):

$$L_{adv} = E_x[\max(0, C(x) - C(x + G(x)) + \kappa)], \quad (4)$$

where $C(\cdot)$ represents the target model's confidence for its predicted class on input x, and κ is a margin parameter. Importantly, this hinge loss formulation directly enforces the adversarial success condition by requiring the model's confidence to drop by at least κ, rather than constraining perturbation magnitude.

To ensure the generated test cases remain realistic and close to the original data distribution, we incorporate a perturbation constraint loss:

$$L_{reg} = E_x\left[\| G(x)\|_2\right]... \quad (5)$$

This l_2 regularization term encourages small perturbations, maintaining the semantic similarity between the original and generated samples.

In the context of this work, we aim for FAGAN to generate high-quality adversarial examples. The quality of an adversarial example is explicitly defined by three quantifiable objectives:

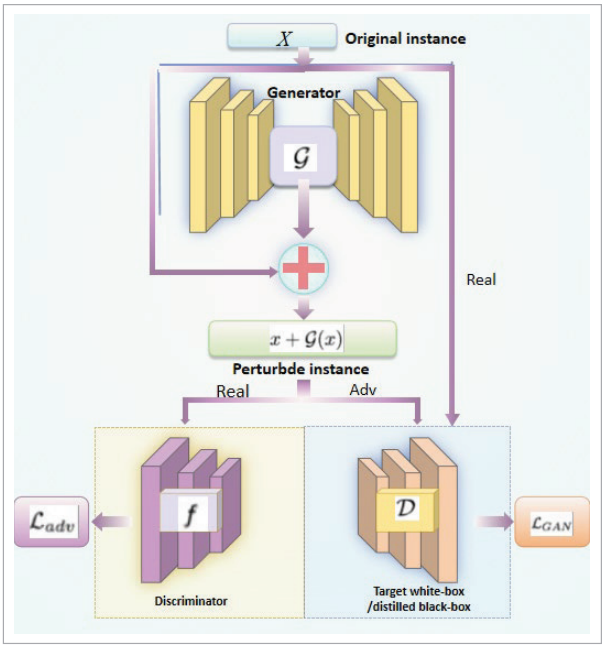
High Attack Success Rate: The example should effectively deceive the target model, causing misclassification or a significant drop in prediction confidence.

High Perceptual Realism: The generated sample should adhere closely to the original data distribution, ensuring it is a visually natural and semantically plausible input.

Low Perturbation Magnitude: The introduced perturbation should be minimal, guaranteeing high similarity between the adversarial example and the original seed – a requirement aligned with the validity of test inputs in fuzzing.

The overall objective function of FAGAN (see Equation (6)) is designed to optimize these three objectives jointly: the adversarial loss term L_{adv} drives high attack success; the GAN loss term L_{GAN} enforces high perceptual realism; and the regularization term L_{reg}

Figure 4
The structure of FAGAN for generating adversarial examples.



constrains low perturbation magnitude. The weighting coefficients α and β are used to dynamically balance the trade-offs among these goals.

The total loss for training the generator G is then a weighted combination of these objectives:

$$L_{total} = \lambda_{adv} * L_{adv} + \lambda_{gan} * L_{GAN} + \cdot\lambda_{reg} *\cdot L_{reg} \quad (6)$$

where L_{GAN} encourages the perturbed data $x+G(x)$ to resemble the original data x, thereby improving realism, while L_{adv} focuses on maximizing the attack success rate against the target model f. The coefficients α and β control the balance between these goals.

We solve this optimization problem by finding the equilibrium point of the min-max game: $argmin_G max_D L_{total}$.

The fuzzing process begins with dataset preprocessing and training the FAGAN model. All preprocessed samples are stored in a processing pool. During the iterative fuzzing loop, original examples are selected from this pool based on their priority. For each selected example, FAGAN is employed to generate multiple adversarial variants. The deep features of both the original and the adversarial examples are then extracted and compared to compute a similarity score. All generated adversarial examples are sorted in descending order of their similarity to the original, and the top-k are selected as candidate test cases. Therefore, FAGAN's uniqueness lies not in its basic adversarial generative architecture, but in its role-specific optimization and its capacity for online adaptation. Unlike static models like AdvGAN whose parameters are fixed after training, FAGAN's generation behavior (guided by weights λ_{gan} and λ_{reg} in Equation (6)) is dynamically adjusted by the Mutation Strategy Controller (see Section 3.4.1) based on real-time feedback. This transforms it from a standalone attack tool into an intelligent, adaptable mutation core for continuous fuzzing campaigns, making it fundamentally different from classical adversarial generators in both purpose and operation. The neuron coverage of these candidates is analyzed; those that improve the coverage of the target DNN are stored back into the processing pool, and their selection priority is updated for the next iteration.

3.4. Monitoring and Control Module: The Adaptive Brain

The Monitoring and Control Module serves as the adaptive brain of the AdapFuzzer framework, its design directly inspired by two cornerstone concepts from control theory: closed-loop feedback and adaptive control [5].

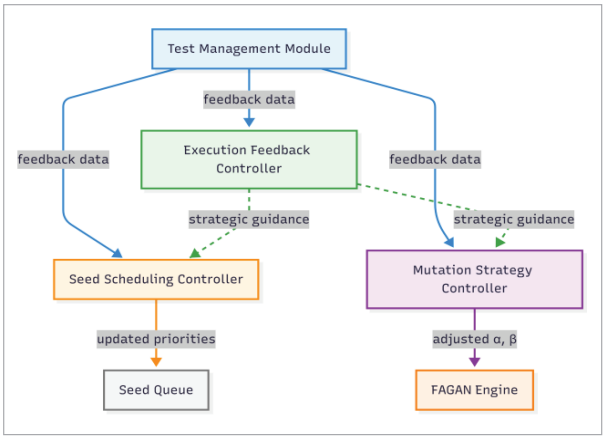
Closed-loop feedback refers to a system that continuously monitors its output (execution results) and uses this information to adjust its future actions (seed selection and mutation). In AdapFuzzer, this is realized through the continuous cycle of test execution, feedback collection by the Test Management Module, and strategic adjustments issued by the controllers.

Adaptive control enables a system to autonomously modify its internal parameters or strategies in response to changing conditions or performance objectives. This is embodied in AdapFuzzer through the three specialized controllers (Seed Scheduling, Mutation Strategy, and Execution Feedback), which dynamically tune the fuzzing campaign based on real-time metrics like coverage trends, diversity, and fault discovery rates.

By embedding these principles, AdapFuzzer transcends a static, open-loop testing sequence and achieves a self-optimizing, intelligent testing process. It continuously analyzes the execution feedback aggregated by the Test Management Module and dynamically steers the fuzzing campaign through the three coordinated controllers. This

Figure 5

Control Flow of the Adaptive Fuzzing Controller.



closed-loop control system, depicted in Figure 2, enables AdapFuzzer to intelligently allocate resources, refine its mutation strategy, and optimize the overall testing process for maximum efficiency and fault-discovery effectiveness.

The Monitoring and Control Module serves as the adaptive brain of the AdapFuzzer framework. It continuously analyzes the execution feedback aggregated by the Test Management Module and dynamically steers the fuzzing campaign through three specialized, coordinated controllers. This closed-loop control system, depicted in Figure 2, enables AdapFuzzer to intelligently allocate resources, refine its mutation strategy, and optimize the overall testing process for maximum efficiency and fault-discovery effectiveness. The detailed internal decision-making logic of the three core controllers is further illustrated in Figure 5.

3.4.1. Controller Logic

The core adaptive logic of each controller is detailed below. Collectively, they implement a rule-based adaptive heuristic control scheme, as opposed to classical continuous controllers (e.g., PID).This design is tailored for the discrete, iterative nature of fuzzing, where objectives like coverage and diversity are best served by multi-objective scoring and threshold-triggered rules. Algorithm 3 provides a high-level pseudocode overview of their integrated operation within the main fuzzing loop, Seed Scheduling Controller: This controller optimizes seed selection by implementing a multi-objective, performance-history-based priority scoring mechanism. The priority $P(s)$ for a seed s in the queue is dynamically updated after each evaluation cycle. It is calculated as a weighted function of its recent contribution to new coverage gain $C_{new}(s)$ and its frequency of triggering unique fault behaviors $F_{unique}(s)$. Seeds that fail to produce novel coverage or faults over a sustained period undergo exponential priority decay. This ensures that seeds with higher potential to explore uncharted model states or reveal critical vulnerabilities are preferentially selected for subsequent mutations and executions.

Mutation Strategy Controller: This controller fine-tunes the FAGAN mutation engine by dynamically adjusting the loss weights in its objective function (e.g., α and β in Equation 5). It monitors the quality of recent batches generated by FAGAN, focusing on two key metrics: the diversity of generated examples (measured via variance in their latent feature representations) and their adversarial success rate. For instance, if the feature-space diversity falls below a threshold, the controller increases the weight (λ_{gan}) of the loss term responsible for maximizing feature distance, thereby steering FAGAN to produce more varied outputs. Conversely, if success rate is low, emphasis is shifted towards the primary adversarial objective.

The specific adjustment rules are formalized in Algorithm 3. This dynamic, feedback-driven parameter update is a cornerstone of FAGAN's design and a key differentiator from static adversarial generators (e.g., AdvGAN). While such classical models employ a fixed loss formulation after training, FAGAN operates as atunable actuator within the adaptive loop. The controller's output—the updated weights (λ_{gan}, λ_{reg})—directly and quantitatively reconfigures FAGAN's generation strategy for the next cycle. This embodies the online adaptability of our framework, ensuring the mutation engine continuously self-optimizes for the evolving needs of the fuzzing process, a capability absent in traditional, one-off attack generators.

Execution Feedback Controller: This controller performs macro-level analysis of the entire testing process. It synthesizes longitudinal data—including trends in aggregate code coverage, resource utilization (e.g., memory, time), and fault detection rates [4] – to identify performance bottlenecks and long-term optimization opportunities. Based on this analysis, it provides strategic, high-level guidance (e.g., "explore more deeply," "widen input space exploration") to the Seed Scheduling and Mutation Strategy controllers, ensuring the campaign adapts not just tactically but also strategically.

Coordination: The three controllers operate in concert, forming a hierarchical feedback loop. The Seed and Mutation controllers handle tactical, per-iteration adaptations, while the Execution Feedback controller provides strategic oversight. This layered approach allows AdapFuzzer to automatically and intelligently refine its testing focus, balancing exploration, exploitation, and resource efficiency throughout the campaign.

Algorithm 3 Controller Adaptation Logic.

1. procedure UPDATE_SEED_SCHEDULER(C_new, F_unique, age)
   for each seed s in queue do
      base_score(s) = ω_c × C_new(s) + ω_f × F_unique(s)
      decay_factor = exp(-λ × age(s))
      $P(s)$ = base_score(s) × decay_factor
   end for
   sort_queue_by_priority(P)
end procedure
2. procedure UPDATE_MUTATION_CONTROLLER (D_batch, S_batch)
   if D_batch < τ_D then
      λ_gan = λ_gan × (1 + δ)
   end if
   if S_batch < τ_S then
      λ_reg = λ_reg × (1 + δ)
   end if
   return (λ_gan, β)
end procedure
3. procedure UPDATE_EXECUTION_FEEDBACK (∇Coverage, ∇Resource, FaultRate)
   if ∇Coverage ≈ 0 then
   broadcast_strategy("INCREASE_EXPLORATION_PRESSURE")
   else if ∇Resource > θ then
   broadcast_strategy("FOCUS_MEMORY_INTENSIVE_PATHS")
   else if FaultRate < ε then
   broadcast_strategy("ADJUST_MUTATION_INTENSITY")
   end if
end procedure
apply_priority_decay(Q)end while
return F

4. Experiment

4.1. Experiment Setup

To comprehensively evaluate the effectiveness and efficiency of AdapFuzzer, this study carefully selected three open – source datasets, eight deep learning models with varying degrees of complexity, and four representative deep learning model testing techniques for experimental evaluation. Among them, the testing techniques cover three fuzz testing techniques and one gradient – descent-based testing technique. Specifically, this paper selected the three most commonly used datasets, MNIST [10], ImageNet [20], and CIFAR-10 [24]. Considering the differences in data complexity among MNIST, imagenet, and CIFAR-10, we matched models of different complexities according to the characteristics of the datasets, including LeNet1, LeNet-4, LeNet-5, as well as VGG-16 and VGG-19 [22], ResNet-50, etc., to ensure the scientific and comprehensive nature of the experimental setup。

To ensure the reproducibility of our experiments and provide a complete technical specification, we detail the key operational and training parameters used for the core components of AdapFuzzer [9]. For the NFuzzer module, the seed queue size n and candidate set size k were set to 100 and 20, respectively, across all experiments. These values were chosen to balance diversity exploration and computational overhead based on preliminary studies.

For the FAGAN mutation engine, the model was trained using the Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 32. The loss weights in the total objective function (Equation (6)) were set as λ_{adv}=1.0 and λ_{reg}=0.1 to emphasize adversarial success while maintaining perturbation constraint. All models and datasets were processed with a fixed random seed to ensure deterministic behavior. A summary of these parameters is provided in Table 2 below.

4.2. Main Result

Our experimental evaluation is structured in two tiers to thoroughly validate the contributions of AdapFuzzer. First, to demonstrate the general applicability and effectiveness of our proposed diversity-driven seed selection mechanism, we evaluate NFuzzer as a plug-in component within several established fuzzing frameworks. The results, summarized in Table 3 and Table 4, show that replacing the native seed selection of these tools with NFuzzer consistently improves their performance. Second, and more critically, we evaluate AdapFuzzer as a complete, standalone framework against state-of-the-art fuzzing techniques. The results of this end-to-end comparison, presented in Table 5 and Table

Table 2
Key parameters of the AdapFuzzer framework.

Component	Parameter	Symbol	Value	Description
NFuzzer	Seed Queue Size	n	100	Maximum number of seeds in the diversity-driven queue.
	Candidate Set Size	k	20	Number of candidates sampled for each selection iteration.
FAGAN	Learning Rate	-	1×10^{-4}	Adam optimizer learning rate for training G and D.
	Batch Size	-	32	Number of samples per training batch.
	Adversarial Loss Weight	λ_{adv}	1.0	Weight for the adversarial loss term L_adv. Serves as the baseline. (Equation (6)).
	Regularization Weight	λ_{reg}	0.1	Weight for the perturbation regularization term L_reg in (Equation (6))
	GAN Loss Weight	λ_{gan}	1.0	Weight for the GAN loss term L_GAN in Equation (6).
General	Random Seed	-	42	Seed for all random operations to ensure reproducibility.

Note: The total loss function of the FAGAN generator is $L_{total} = \lambda_{adv} \cdot L_{adv} + \lambda_{gan} \cdot L_{GAN} + \lambda_{reg} \cdot L_{reg}$, where the weights λ_{gan} and λ_{reg} are dynamically adjusted by the Monitoring and Control Module during fuzzing to achieve adaptive mutation.

Table 3
The average number of adversarial inputs generated by the seed queues constructed based on different techniques of various fuzz testing technologies. The seed queue generated by DMTNet produced, on average, more adversarial inputs than the seed queues generated by using random and DMSNet.

Fuzz Testing TechMethod	Method	Number of adversarial Example Generation					
		LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	ResNet-50
DeepXplore [18]	Random	13.9	10.1	11.6	14.1	13.2	15.3
	DMSNet	14.1	11.2	12.3	14.3	14.8	15.7
	DMTNet	15.3	13.2	14.3	15.4	16.5	17.9
DLFuzz [6]	Random	3.3	11.1	11.3	13.1	14.2	15.1
	DMSNet	14.1	12.2	12.8	14.3	14.8	16.7
	DMTNet	15.3	14.2	16.3	17.4	17.5	18.9
Tensorfuzz [1718]	Random	331.9	311.9	355.1	351.2	341.9	374.1
	DMSNet	358.4	348.4	378.4	368.1	341.2	388.2
	DMTNet	441.6	541.2	461.1	451.5	421.1	471.3
Deephunter [24]	Random	77.1	73.6	66.4	87.1	83.6	87.4
	DMSNet	87.6	82.6	81.4	89.4	86.7	82.4
	DMTNet	91.2	91.5	89.1	93.1	92.4	87.1

6, directly attest to the overall superiority of our integrated approach.

The effectiveness of the proposed seed selection strategy is first demonstrated by the quantity of adversarial inputs it helps to generate. Table 3 summarizes the average number of adversarial inputs produced by various fuzzing techniques when using seed queues constructed with random selection, DMSNet, and DMTNet. The results clearly show that seed queues generated by DMTNet consistent-

ly yield more adversarial inputs than those generated by random selection or DMSNet. In most cases, when equipped with DMTNet-constructed seed queues, baseline tools like DeepXplore, DLFuzz, Tensorfuzz, and Deephunter see a marked increase in their adversarial output.

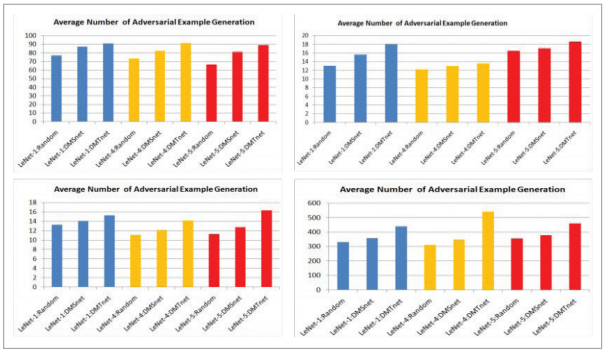
The distribution of these results is further illustrated in Figure 6, compared to random, both DMSNet and DMTNet significantly help fuzzing techniques in generating a larger number of adversarial inputs. As for DMTNet versus DMSNet, DMTNet generally outperforms DMSNet in most scenarios. The distribution of the number of adversarial inputs generated by fuzzing techniques using seed queues created with different methods is illustrated in the figure. Compared to random, NFuzzer also aids in producing more adversarial inputs. For all models, DMTNet assists in generating a greater quantity of adversarial inputs. In most cases (except forResNet and LeNet-4), DMSNet helps in generating more adversarial inputs. On average, the number of adversarial inputs generated by fuzzing techniques using seed queues built by DMTNet/DMSNet is 13 times that of random. In the best case, the number of adversarial inputs generated by fuzzing techniques using DMTNet/DMSNet can be 150 times that of random. In the optimal scenario, the number of adversarial inputs generated by the fuzzing technique using the seed queue built by DMTNet/DMSNet is 200 times that of the one built by random.

We next evaluate the testing adequacy in terms of neuron coverage. Table 4 provides the average nc (neuron coverage) or tfc (top-k feature coverage) results achieved by different fuzzing techniques using various seed queue construction methods (deepxplore, DLFuzz, and Deephunter use nc as their coverage criterion). Similar to the adversarial input results, DMSNet and DMTNet, in comparison to random, help fuzzing techniques achieve better coverage results and explore more internal states of the model. More specifically, in most instances, DMSNet and DMTNet both achieve higher coverage than random. In a few cases (DLFuzz testing LeNet-1 and Deephunter testing LeNet-1), DMSNet, DMTNet, and random have the same coverage. On average, the neuron coverage achieved by fuzzing techniques using DMTNet-constructed seed queues is higher compared to random. In the best case, the neuron coverage achieved by fuzzing techniques using DMSNet/DMTNet is 443% higher than that achieved with random. Similarly, the coverage achieved by fuzzing techniques using DMSNet/DMTNet is increased. In the best case, the coverage achieved by fuzzing techniques using DMSNet/DMTNet is 556% higher than that achieved with random.

Furthermore, we note an insightful detail in Table 4 worthy of deeper analysis: the neuron coverage (27.1) achieved when Deephunter is combined with DMTNet for testing LeNet-5 is lower than the random baseline (36.4). We posit that this stems primarily from a characteristic mismatch between the texture-focused nature of DMTNet and the LeNet-5 model/dataset. MNIST images processed by LeNet-5 contain minimal texture information. In such an environment, DMTNet struggles to leverage its strengths, potentially causing the seed selection process to fail to optimally select seeds that maximize the activation of neurons based on structural discrepancies. This phenomenon precisely illustrates the importance of selecting or adapting the difference metric strategy according to the characteristics of the target model and data. It also points to a direction for future work-developing more environmentally adaptive metric models. Nonetheless, this localized anomaly does not undermine the overall conclusions of this paper, as DMTNet consistently demonstrates performance superior to the random baseline in all other combinations of models and testing frameworks, robustly validating the general effectiveness of seed selection based on deep difference metrics.

Figure 6

The distribution results of the number of adversarial inputs generated by the seed queues produced by different fuzz testing techniques based on various technologies.



The superior performance of AdapFuzzer in generating a higher number of adversarial inputs (Table 4) and achieving greater neuron coverage (Table 5) can be largely attributed to the design of the FAGAN mutation engine. Unlike traditional mutation operators that apply random, hand-crafted, or single-step gradient-based perturbations, FAGAN leverages a generative adversarial network trained with a multi-objective loss function (Equation (6)). This enables it to produce targeted, realistic, and diverse adversarial examples in a systematic manner.

Key advantages of FAGAN include:

- **Directed Exploration:** While methods like DeepXplore and DLFuzz rely on gradient signals that may converge to local optima or similar regions, FAGAN's generator learns to produce perturbations that effectively deceive the target model, guiding the search towards regions of the input space that are more likely to reveal faults.
- **High Realism and Coherence:** The adversarial training with the discriminator (via LGAN) ensures that generated samples remain close to the natural data manifold, producing semantically valid inputs – a property often lacking in random or aggressive pixel-level perturbations.
- **Diversity through Feature-Space Optimization:** The incorporation of feature-space constraints (implicitly via the GAN architecture and explicitly via the triplet loss during training) encourages the generation of varied test cases, complementing NFuzzer's seed-level diversity.

In contrast, baseline methods often struggle with one or more of these aspects: random mutations (as in Tensorfuzz) are inefficient; gradient-based methods may lack diversity; and hand-crafted operators may not adapt to different models. The consistent outperformance of AdapFuzzer across diverse models and datasets underscores that the adaptive, learning-based mutation strategy embodied by FAGAN is a cornerstone of our framework's effectiveness [16].

Table 5 and Table 6 show that, benefiting from AdapFuzzer's adoption of the efficient mutant seed generator FAGAN, AdapFuzzer can generate more adversarial inputs compared with commonly used testing techniques. When implementing AdapFuzzer and observing the achieved neuron coverage, we set an upper limit on the size of the test suite and applied AdapFuzzer to the testing of DNNS. As a result, AdapFuzzer achieved a neuron coverage of 48.1% when testing LeNet-1 and 35% when testing VGG-16. AdapFuzzer

Table 4

The average nc or tfc results achieved by different fuzz testing techniques using different seed queue construction techniques.

Fuzz Testing Tech	Method	Average NC/TFC					
		LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	ResNet-50
DeepXplore [18]	Random	37.1	36.6	38.2	4.1	3.2	1.3
	DMSNet	44.1	41.2	42.3	4.3	4.8	1.7
	DMTNet	45.3	43.2	44.3	5.4	6.5	1.9
DLFuzz [6]	Random	33.3	31.1	31.3	3.1	4.2	0.1
	DMSNet	34.1	42.2	42.8	4.3	4.8	0.4
	DMTNet	45.3	44.2	46.3	5.4	5.5	0.6
Tensorfuzz [17]	Random	51.9	171.9	175.1	5.1	11.9	4.1
	DMSNet	58.4	188.4	188.4	5.4	18.4	6.4
	DMTNet	61.6	191.2	191.1	6.6	19.2	4.4
Deephunter [24]	Random	37.1	41.6	36.4	7.1	4.1	0.2
	DMSNet	37.6	42.6	37.4	7.4	4.6	0.4
	DMTNet	41.2	44.5	27.1	7.6	4.6	0.4

Table 5
Analysis of the capability of AdapFuzzer in generating adversarial samples.

Test Frame	Number of Adversarial Example Generation					
	MNIST-LeNet-1	MNIST-LeNet-4	MNIST-LeNet-5	ImageNet-VGG-16	ImageNet-VGG-19	CIFAR10-ResNet-50
DeepXplore [18]	26.4	28.6	22.9	141.1	141.1	241.1
Tensorfuzz [6]	20.4	20.6	20.9	111.1	211.4	211.1
Deephunter [15]	20.5	21.4	30.4	165.4	115.4	365.4
FGSM [22]	22.5	20.9	16.4	211.3	165.2	295.2
AdapFuzzer	38.3	44.5	50.1	20.5	465.2	742.2

Table 6
Analysis of the coverage capability of AdapFuzzer.

Test Frame	Av.NC					
	MNIST-LeNet-1	MNIST-LeNet-4	MNIST-LeNet-5	ImageNet-VGG-16	ImageNet-VGG-19	CIFAR10-ResNet-50
DeepXplore [18]	36.4	48.6	52.9	22.1	21.1	5.1
Tensorfuzz [6]	40.4	50.6	50.9	21.1	21.4	4.1
Deephunter [24]	40.5	61.4	50.4	25.4	25.4	5.4
FGSM [22]	32.5	60.9	66.4	21.3	25.2	5.2
AdapFuzzer	48.3	74.5	70.1	30.5	35.2	6.2

requires fewer test cases to reach the same level of neuron coverage. We also compared the performance of AdapFuzzer on VGG of CIFAR10 [11] and imagenet with that of the benchmark fuzzer. According to our reproduction results, although the advantages of AdapFuzzer are not significant, it can still compete with the existing mutation-based fuzzers. Compared with other testing methods, AdapFuzzer achieves higher neuron coverage and has a stronger ability to explore the internal state of the model.

4.3. Analysis

A detailed breakdown of the coverage improvement is presented in Table 7. The data shows that in the vast majority of scenarios, DMSNet achieves higher coverage than random methods (i.e., the improvement value is positive). It is worth noting that there is no case where DMSNet's coverage was lower than random; moreover, DMTNet typically outperforms DMSNet.

The time efficiency of our approach is evaluated in Table 8. It can be seen that, compared to random, both DMSNet and DMTNet help fuzzing techniques generate adversarial inputs in a shorter amount of time. In other words, NFuzzer can assist fuzzing techniques in generating adversarial inputs faster than random. While DMTNet performs slightly better than DMSNet, the advantage is not significant. Compared to random, DMSNet and DMTNet take longer to construct seed queues, but on average, they require less time to generate each adversarial input. Therefore, the time overhead of the proposed NFuzzer is acceptable.

Finally, we demonstrate the practical utility of the tests generated by AdapFuzzer through adversarial retraining. As shown in Figure 7, by employing adversarial retraining, we retrained the tested DNNS. We added 2000 adversarial examples to the original training set of each dnn. The DNNS were then retrained for 5 epochs on the augmented training

Table 7
When using the seed queue constructed by DMSNet for fuzz testing,the improvement in coverage compared to the random method.

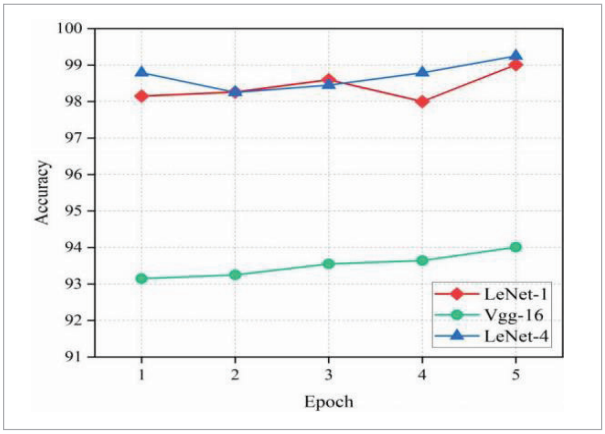
Fuzz Testing Tech	Method	NC/TFC improve percentage base on random					
		LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	ResNet-50
DeepXplore [18]	DMSNet	6.1	7.2	5.3	1.3	1.8	1.7
	DMTNet	7.3	7.2	6.3	5.4	6.5	1.9
DLFuzz [6]	DMSNet	5.1	7.2	6.8	8.3	3.8	5.4
	DMTNet	7.3	6.2	8.3	4.4	6.5	7.6
Tensorfuzz [6]	DMSNet	3.4	4.4	2.4	1.4	1.5	1.3
	DMTNet	5.6	5.2	4.1	3.6	4.2	3.1
Deephunter [24]	DMSNet	0.6	0.6	0.4	0.3	0.5	0.4
	DMTNet	1.2	1.5	1.1	0.6	0.7	1.1

Table 8
The time overheads of random,DMSNet,and DMTNet in constructing seed queues on different datasets.

Fuzz Testing Tech	Method	Time(s)					
		LeNet-1	LeNet-4	LeNet-5	VGG-16	VGG-19	ResNet-50
DeepXplore [18]	Random	0.9	1.6	1.2	11.1	13.2	11.3
	DMSNet	0.5	1.2	0.3	3.3	3.8	3.7
	DMTNet	0.5	1.4	0.4	5.4	6.5	4.9
DLFuzz [6]	Random	1.3	1.1	1.3	83.1	74.2	9.1
	DMSNet	1.1	2.2	1.8	74.3	64.8	8.4
	DMTNet	1.3	1.2	1.3	65.4	75.5	8.6
Tensorfuzz [6]	Random	3.9	3.9	3.1	39.9	32.2	34.1
	DMSNet	1.4	2.4	3.4	15.1	21.6	35.4
	DMTNet	1.6	1.2	1.1	11.5	13.2	11.7
Deephunter [24]	Random	4.1	3.6	2.4	46.1	30.6	20.3
	DMSNet	3.6	1.6	1.4	30.6	1.3	14..1
	DMTNet	1.2	1.5	1.1	11.5	12.7	11.3

set. The figure displays the trend of accuracy improvement on the test set for the retrained models. We compared the performance of the newly trained networks with that of the previous ones using a test set consisting of 1000 test cases and 2000 real-world samples. As a result, the retrained LeNet-1, LeNet-4, and VGG-16 networks showed an increase in accuracy on the mixed test set by 1.22% and 1.17%, respectively. This indicates that the test cases generated by AdapFuzzer can, to some extent, fix erroneous behavior, thereby effectively improving the tested DNNS.

Figure 7
The figure shows the growth trend of the accuracy of the retrained model on the test set.



5. Conclusion

In this paper, we proposed AdapFuzzer, an adaptive fuzzing framework that embeds principles of control theory into the testing process for deep learning systems. By leveraging a storage module to record key testing information and a feedback loop to guide seed selection and mutation strategies, AdapFuzzer achieves autonomous and intelligent fuzzing. The core of its adaptability lies in the novel NFuzzer algorithm, which constructs a diversity-aware seed queue by iteratively selecting the most dissimilar test cases. The perceptual difference between inputs is quantified by our specifically developed neural network-based models, DMSNet and DMTNet.

Extensive experimental results demonstrate that AdapFuzzer, powered by NFuzzer and the FAGAN mutation engine, significantly improved the detection efficiency and accuracy compared to existing fuzzing techniques. It not only generates a substantially higher number of adversarial inputs but also achieves greater neuron coverage, providing a more robust and comprehensive security assessment for deep learning models.

In summary, this work offers a novel and effective methodology for probing the robustness of deep learning systems. The introduced adaptive mechanism and quantitative difference metrics address the critical limitations of diversity-agnostic and non-adaptive testing strategies. In summary, this work offers a novel and effective methodology for probing the robustness of deep learning systems. The introduced adaptive mechanism and quantitative difference metrics address the critical limitations of diversity-agnostic and non-adaptive testing strategies [12]. Future research will focus on extending and refining the AdapFuzzer framework along several promising directions. First, we plan to adapt the framework to test emerging Transformer-based models and other novel architectures, as well as extend its application to different data modalities beyond images. Second, exploring new, more semantically-grounded testing adequacy criteria – such as behavior-based coverage or concept interpretability – could complement neuron coverage and provide deeper insights into model robustness [27]. Third, to enhance practical utility, we aim to optimize the computational overhead of FAGAN through techniques like model distillation, efficient generative architectures, or adaptive batch scheduling [13].

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, author-ship, and/or publication of this article.

Data Sharing Agreement

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Funding

The authors gratefully acknowledge the financial support provided by the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), which contributed to the successful completion of this research.

References

1. Aditya Baddukonda, T., Kanakam, R. K., Yasotha, B. Adversarial Attacks Using Carlini Wagner. In, Proceedings of the IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS), 2024, 1-7. https://doi.org/10.1109/ICITEICS61368.2024.10624973

2. Carlini, N., Wagner, D. Towards Evaluating the Robustness of Neural Networks. In, Proceedings of the IEEE Symposium on Security and Privacy (SP), 2017, 39-57. https://doi.org/10.1109/SP.2017.49

3. Chen, X., Cui, N., Pan, Z., Chen, L., Shi, G., Meng, D. Critical Variable State-Aware Directed Greybox Fuzzing. In, Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2025, 755-755. https://doi.org/10.1109/ICSE55347.2025.00219

4. Dai, H., Murphy, C., Kaiser, G. Configuration Fuzzing for Software Vulnerability Detection. In, Proceedings of the International Conference on Availability, Reliability and Security (ARES), 2010, 525-530. https://doi.org/10.1109/ARES.2010.22

5. Grano, C., Laaber, C., Panichella, A., Panichella, S. Testing with Fewer Resources: An Adaptive Approach to Performance-Aware Test Case Generation. IEEE Transactions on Software Engineering, 2021, 47(11), 2332-2347. https://doi.org/10.1109/TSE.2019.2946773

6. Guo, J., Jiang, Y., Zhao, Y., Chen, Q., Sun, J. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In, Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2018, 739-743. https://doi.org/10.1145/3236024.3264835

7. Hoffer, E., Ailon, N. Deep Metric Learning Using Triplet Network. In, Proceedings of the International Workshop on Similarity-Based Pattern Recognition, 2015, 84-92. https://doi.org/10.1007/978-3-319-24261-3_7

8. Hu, L., Ma, X., Xie, X., Yu, B., Liu, Y., Zhao, J. DeepMutation++: A Mutation Testing Framework for Deep Learning Systems. In, Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, 1158-1161. https://doi.org/10.1109/ASE.2019.00126

9. Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P. Taxonomy of Real Faults in Deep Learning Systems. In, Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE), 2020, 1110-1121. https://doi.org/10.1145/3377811.3380395

10. LeCun, Y. The MNIST Database of Handwritten Digits, 1998. Available at, MNIST Database.

11. Li, H., Liu, H., Ji, X., Li, G., Shi, L. CIFAR10-DVS: An Event-Stream Dataset for Object Classification. Frontiers in Neuroscience, 2017, 11, 309. https://doi.org/10.3389/fnins.2017.00309

12. Li, Z., Du, X., Lei, N., Chen, L., Wang, W. NoPain: No-Box Point Cloud Attack via Optimal Transport Singular Boundary. In, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025, 3492-3502. https://doi.org/10.1109/CVPR52734.2025.00331

13. Lin, Z., Peng, C., Tan, W., He, X. Image Adversarial Example Generation Method Based on Adaptive Parameter Adjustable Differential Evolution. Entropy, 2023, 25(3), 487. https://doi.org/10.3390/e25030487

14. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv Preprint arXiv:1706.06083, 2017. DOI: 10.48550/arXiv.1706.06083.

15. Min, Y., Wang, Z., Liu, Y., Wang, Z. FS-RSDD: Few-Shot Rail Surface Defect Detection with Prototype Learning. Sensors, 2023, 23(18), 7894. https://doi.org/10.3390/s23187894

16. Moreno-Armendáriz, M. A., Calvo, H., Duchanoy, C. A., Lara-Cázares, A., Ramos-Diaz, E., Morales-Flores, V. L. Deep-Learning-Based Adaptive Advertising with Augmented Reality. Sensors, 2022, 22(1), 63. https://doi.org/10.3390/s22010063

17. Odena, A., Olsson, C., Andersen, D., Goodfellow, I. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In, Proceedings of the International Conference on Machine Learning (ICML), 2019, 4901-4911. Available at, PMLR Proceedings.

18. Pei, K., Cao, Y., Yang, J., Jana, S. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In, Proceedings of the Symposium on Operating Systems Principles (SOSP), 2017, 1-18. https://doi.org/10.1145/3132747.3132785

19. Ranzato, M. A., Boureau, Y. L., LeCun, Y. Sparse Feature Learning for Deep Belief Networks. In, Advances in Neural Information Processing Systems (NeurIPS), 2007, 20. Available at, NeurIPS Paper.

20. Russakovsky, O., Deng, J., Su, H., et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 2015, 115, 211-252. https://doi.org/10.1007/s11263-015-0816-y

21. Wang, H., You, J., Chen, Y., Zhang, X., Dong, X., Zhang, W. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. In, Proceedings of the IEEE/

ACM International Conference on Software Engineering (ICSE), 2021, 397-409. https://doi.org/10.1109/ICSE43902.2021.00046

22. Wang, Y., Liu, J., Chang, X., Wang, J., Rodríguez, R. J. AB-FGSM: AdaBelief Optimizer and FGSM-Based Approach to Generate Adversarial Examples. Journal of Information Security and Applications, 2022, 68, 103227. https://doi.org/10.1016/j.jisa.2022.103227
23. Xiao, C., Li, B., Zhu, J., He, W., Liu, M., Song, D. Generating Adversarial Examples with Adversarial Networks. In, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2018, 3905-3911. https://doi.org/10.24963/ijcai.2018/543
24. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In, Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2019, 146-157. https://doi.org/10.1145/3293882.3330579
25. Zhao, L., Lv, X., Zhu, L., Luo, B., Cao, H., Cui, J., Li, H., Peng, J. A Local Adversarial Attack with a Maximum Aggregated Region Sparseness Strategy for 3D Objects. Journal of Imaging, 2025, 11(1), 25. https://doi.org/10.3390/jimaging11010025
26. Zhao, Z., Woloszynek, S., Agbavor, F., Mell, J. C., Sokhansanj, B. A., Rosen, G. L. Learning, Visualizing and Exploring 16S rRNA Structure Using an Attention-Based Deep Neural Network. PLoS Computational Biology, 2021, 17(9), e1009345. https://doi.org/10.1371/journal.pcbi.1009345
27. Zhou, Z., Abawajy, J. Reinforcement Learning-Based Edge Server Placement in the Intelligent Internet of Vehicles Environment. IEEE Transactions on Intelligent Transportation Systems, 2025, 1-11. https://doi.org/10.1109/TITS.2025.3557259