Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.12 n.58, p.1256-1268. March/April, 2026.
DOI: https://doi.org/10.5935/jetia.v12i58.3305

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

A3D: JOINT SEARCH FOR ROBUST DNNS AND ADVERSARIAL ATTACKS VIA AUTOML

Sarala Patchala*[1], Rajani Bodapalli[2], Vullam Naga Gopi Raju[3], Desamala Prabhakara Rao[4], Vijay Babu Burra[5] and D. Kishore[6]

[1]Associate Professor, Department of ECE, KKR & KSR Institute of Technology and Sciences, Guntur, Andhra Pradesh, India, Andhra Pradesh, India
[2]Associate Professor, Department of EEE, Aditya University, Surampalem,A.P.
[3]Professor, Department of Computer Science and Engineering, Chalapathi Institute Of Engineering And Technology, Guntur
[4]Department f ECE, Chalapathi Institute of Technology, Mothadaka,Guntur, Andhra Pradesh, India
[5]Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, GUNTUR
[6]Professor, Department of ECE, Aditya University, Surampalem,A.P.

[1]https://orcid.org/0000-0002-5184-0814, [2]https://orcid.org/0000-0002-4736-3330, [3]https://orcid.org/0009-0008-4894-375X, [4]https://orcid.org/0009-0001-3930-3258, [5]https://orcid.org/0000-0003-0139-8025, [6]https://orcid.org/0000-0001-9258-0847

Email: *saralajntuk@gmail.com, drrajaniboddepalli2015@gmail.com, gopi.raju524@gmail.com, prabhakardesamala@gmail.com, vijay_gemini@kluniversity.in, kishore.dannina@gmail.com

ARTICLE INFO

Article History
Received: January 13, 2026
Reviewed: February 15, 2026
Accepted: March 27, 2026
Published: April 30, 2026

Keywords:
Deep neural networks
A3D
Attacks
Adversarial

ABSTRACT

Deep neural networks (DNNs) are widely used today**.** These methods work well in many tasks but show weakness against adversarial attacks. Attacks use small changes in input images to fool DNNs. Because of this, many platforms exist to test the strength or weakness of a DNN. However, most current platforms face two problems. First, improving the structure of neural networks is not possible. Second, creating stronger attacks is not supported. As a result, these platforms do not help make DNNs more secure or smarter. To address this, a new system was proposed named A3D. It stands for Auto-Adversarial Attack and Defense. This platform does two main jobs. It finds strong DNN structures. It finds smart attack methods. This is done by using automatic search techniques.A3D uses different search tools. These tools help to build better DNNs that are harder to fool. A3D finds better ways to fool the DNNs. The system works in both directions. Stronger the attacks, better the defense models it creates. The stronger the models, the smarter the attacks it designs.A3D is tested on popular datasets. These include CIFAR10, CIFAR100 and ImageNet. The results show that A3D works well. It creates DNNs that are more secure. It creates attacks that are more effective.A3D is a powerful tool. It helps researchers test and improve DNNs in a smart and automatic way. It solves the limits of older systems. And it brings better security and performance to deep learning.

I. INTRODUCTION

Deep learning plays an important role in many computers vision tasks [1]. These tasks include image classification, object detection, segmentation and tracking. DNNs are the main tools used in these tasks [2]. DNNs show very high performance when the input data is clean. However, itis not reliable when the input is slightly changed by an attacker. Even small and invisible changes in the input fool a DNN model [3]. These harmful inputs are named as adversarial examples (AEs).Adversarial examples are created by adding carefully designed noise to an image.

These changes do not affect the way humans see the image, but easily mislead a DNN. This weakness causes big problems in real-world applications for example self-driving cars and facial recognition systems [4]. There are two main areas in this field: adversarial attack and adversarial defense. Adversarial attack methods try to find the weakest points of a model [5]. It generates images that look normal but trick the model into making wrong predictions. These methods include FGSM, PGD and CW attacks.

These attacks change many factors like the direction of the gradient, the size of the noise, the number of iterations and the loss functions used [6]. On the other hand, adversarial defense objects to build models that resist these attacks. One common method is adversarial training. This involves training the model with both clean and attacked images [7]. Another way is to design network structures that are strong by default. Many researchers worked on different types of defenses to make DNNs more robust [8].To test and compare these methods, many open-source platforms are developed. Some popular ones include CleverHans, Foolbox and ART. However, these platforms mostly focus on testing. The main problem is that these tools cannot improve themselves [9]. As a result, this cannot push the limits of current deep learning models [10]. To solve this problem, the paper introduces the clue of using Auto Machine Learning (AutoML).AutoML is a method that searchs for the best models and settings automatically [11]. It reduces the need for manual tuning. AutoML works by defining a search space, using a search algorithm and applying an evaluation metric. It isalready shown great success in selecting models and tuning hyperparameters [12]. Some researchers used AutoML to build better attacks or find robust networks. But most of these efforts only focus on one side-either attack or defense.

Another challenge is that different AutoML methods use different search spaces, strategies and metrics [13]. This makes it hard to compare them fairly. Some attacks only work on weak models. Some models are strong against simple attacks. There is no unified way to test both sides together. This leads to a gap in the field. Without a combined approach, cannot build truly robust systems [14].To fill this gap, the paper proposes a new platform termed A3D. This platform joins both attack and defense into one loop. It automatically find robust network architectures. It discovers efficient and smart adversarial attacks. A3D works by letting the attack part test the defense part and vice versa [15]. This feedback loop makes both parts stronger over time.The A3D platform uses advanced techniques to improve DNNs. It searches for model architectures using different robustness evaluation methods. It uses optimization methods to find strong and efficient attacks. This dual approach allows A3D to handle many types of noise and test scenarios. The A3D platform is tested on standard datasets like CIFAR10, CIFAR100 and ImageNet [16]. The results show that it works well. It finds better models that resist attacks more effectively. It creates attacks that test the limits of these models. This proves that the platform is useful and practical for real-world problems. The Key Contributions of this paper are:

- To evaluate eleven NAS methods to identify robust model architectures. Test performance under four scenarios: adversarial noise, natural noise, system noise and quantitative metrics. Create tailored loss functions for each evaluation type.
- To develop evolutionary optimization algorithms to generate efficient adversarial attacks aimed at significantly reducing robust accuracy of machine learning models while minimizing attack cost.
- To develop an integrated AutoML framework (A3D) that unites attack and defense strategies. Use adversarial attacks to strengthen model robustness. Use robust models to guide the creation of stronger attacks. Enable mutual improvement through a closed-loop learning process.

This clearly explains why DNN models are vulnerable. It reviews current tools and methods and shows existing limitations. The A3D platform is proposed as a solution that brings AutoML to both attack and defense. It is the first unified system that improves both parts automatically and together. This marks a big step forward in building secure and smart deep learning systems.

II.BACKGROUND & RELATED WORK

This section reviews the background of adversarial attacks and defenses and the role of AutoML in improving them. It provides a complete understanding of the steps that led to the design of the A3D framework.Adversarial attacks seek to fool DNNs. These attacks add small and invisible noise to input images. This noise tricks the model into making wrong predictions. The image looks the same to humans, but the model fails. The task is to find a small change Δx that causes the model F to misclassify $x+\Delta x$. This optimization problem is shown in [17].Many popular attack methods are developed. FGSM (Fast Gradient Sign Method) was one of the earliest methods [17]. Later, PGD (Projected Gradient Descent) [18] and CW (Carlini & Wagner) attack [19] were introduced to create stronger attacks. Other methods include Momentum Iterative Attack (MI) [20] and MultiTargeted Attack (MT) [21]. To defend against these attacks, researchers proposed several methods. One strong defense is adversarial training. In this method, the model is trained with both clean and adversarial examples [17]. Other defense strategies include defensive distillation [22], input transformations [23] and building robust network architectures [24].

To support research in this field, various platforms were introduced. CleverHans is one of the first and most used platforms [25]. It supports FGSM and JSMA-based training and attacks. Foolbox [26] offers a wider variety of attacks. ART (Adversarial Robustness Toolbox) includes attacks for text, audio and image tasks [27]. Dong et al. introduced a benchmark for evaluating robustness on image classification tasks [28]. There are platforms for language [29] and graph data [30].However, all these tools focus only on testing. The settings are manual and fixed. As a result, it is limited in creating stronger defenses or smarter attacks.AutoML is a useful tool for this problem. AutoML is able search for the best model or best attack settings automatically. NAS is a core part of AutoML. It finds better model structures for a task.NAS splits into two parts. The inner part trains weights using gradients. The outer part updates model architecture. DARTS (Differentiable Architecture Search) is a popular NAS method [31]. It is fast and efficient. PC-DARTS [32] and FairDARTS [33] are improved versions.

For adversarial defense, researchers use metrics like the Jacobian matrix [34] or the Hessian matrix [35] to guide the search. There are non-differentiable NAS methods. These include Random Search, ENAS [36], DE-NAS, NAO [37] and BONAS [38]. Model structures are sampled and evaluated with complete training and testing.Though slower, these methods are useful when gradients are hard to compute.AutoML is used in generating adversarial attacks. Mao et al. proposed Composite Adversarial Attack (CAA) [39]. It uses NSGA-II to search for better attack combinations. Yao et al. expanded this by adding data transformations and multiple loss functions to the search space [40]. Fu et al. designed AutoAdv, a decision-based attack using AutoML [41]. Croce et al. showed that combining attacks gives better results than using a single one [42].But many of these works focus only on one side—either attack or defense. This is a problem. To address these problems, the A3D platform is introduced.

A3D combines NAS for model defense and evolutionary search for attack generation. It creates a loop. Stronger attacks test the models. Better models force stronger attacks. Over time, both sides improve together. This unified AutoML framework is what makes A3D unique and powerful.

III.THE FRAMEWORK OF AUTO-ADVERSARIAL ATTACK & DEFENSE

A3D has two main modules. The top part is the NAS module. The bottom part is the attack search module. It finds smart ways to fool the models. These two parts work together. The searched attack helps test the models. The searched model helps build better attacks. This loop makes the system stronger over time. A3D focuses on building strong models. These models should work well even when noise or attacks are added to the input images. The Figure 1 explains a framework that performs automatic neural architecture search to create robust models against adversarial attacks. It starts with data settings on the left. Datasets like CIFAR10, CIFAR100 and ImageNet are used. These datasets go through preprocessing steps. These preprocessed inputs are used in the main process of finding robust neural networks. This main process includes three steps.

These steps help in selecting strong architectures by testing them under different conditions. Once the robust models are found, it is passed through multiple layers and added to the adversarial attack pool to test the strength. These models, identified as robust architectures are stored in a pool and reused for training. On the right-side training settings like learning rate, batch size, epoch count and optimizer type are defined. These control the process for training and evaluating robust models. An evolutionary search is used for adversarial attacks. This includes attacker search space, attacker strategy and attacker evaluation. The purpose is to generate new and efficient attacks that challenge the robustness of the current models. These attacks are then added to the adversarial attacks pool, that helps improve the testing process. The robust architectures and attack pool are used together in retraining to build even stronger models. This loop of training and testing helps to confirm that the final models are more secure. It handles real-world adversarial challenges effectively.

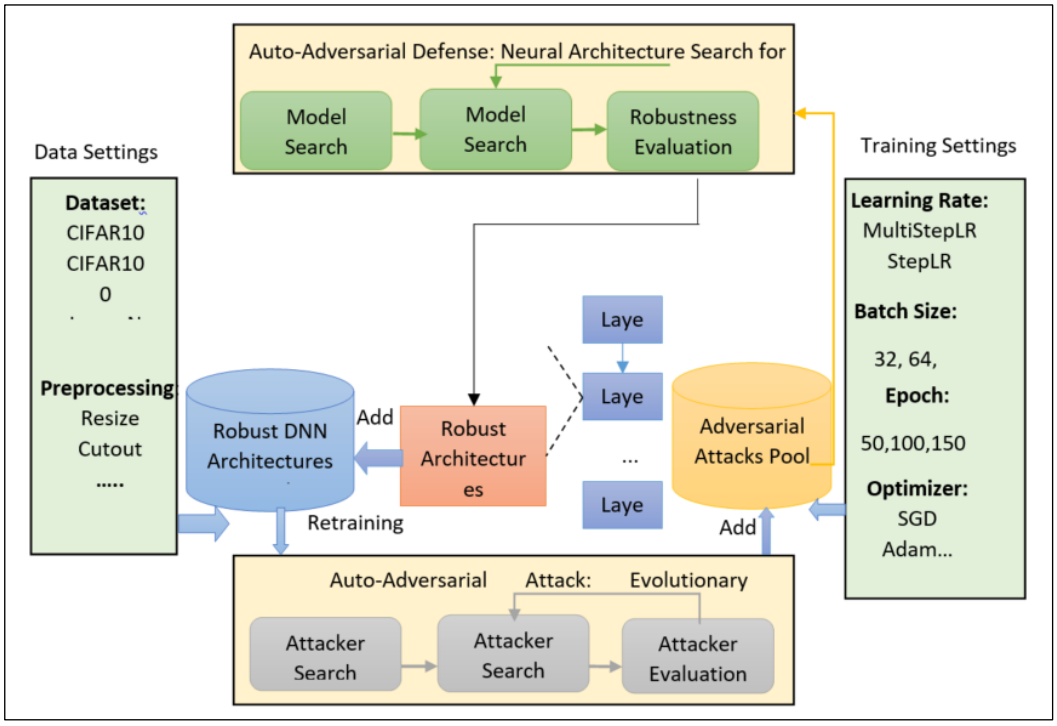

Figure 1: A3D framework.
Source: Authors, (2026).

To do this, use different NAS methods. First, find model designs (denoted as α) that give high accuracy and high robustness. The inner loop finds the best weights w for a given model design. The outer loop chooses the best model structure α. The optimization objective is:

$$min_{\alpha} L_{val}(W^{*}(\alpha), \alpha) + R(W^{*}(\alpha), \alpha) \quad (1)$$

$$s.t. W^{*}(\alpha) = argmin_{w} L_{train}^{adv}(w, \alpha) \quad (2)$$

Here, L_{val} is the loss on the validation set. L_{train}^{adv} is the adversarial training loss. R is the robustness metric. This way, search for strong models that perform well under both clean and noisy data. A3D checks model strength using four types of robustness tests. The first is **adversarial noise**, created by attacks like FGSM and PGD. These attacks add small, smart noise using gradients. If a model handle these, it is seen as strong. The second is **natural noise**, like lighting changes or image blur. This noise happens in real life and does not come from gradient methods. The third type is system noise**,** caused by hardware or software issues. Examples are image compression errors or memory faults. The last type is quantified metrics. These use math values like Jacobian or Hessian norms. Lower values mean the model is more stable. For adversarial noise, define the robust accuracy (RA) as:

$$RA(W^*(\alpha), \alpha, X_i^{adv}) \tag{3}$$

Here, x_i^{adv} is an adversarial example made by an attack algorithm. For quantified metrics use:

$$L\,(W^*(\alpha), \alpha;\, D_{val}) + \gamma\, R_J(W^*(\alpha), \alpha, x_i) \tag{4}$$

Here R_J is the Jacobian value and γ is a balancing factor.A3D uses two types of NAS methods. Differentiable **NAS**, like DARTS and FairDARTS, uses gradients. This makes the search fast and efficient. It scales well with larger models. Non-differentiable NAS does not use gradients. It includes random search and evolutionary algorithms. These are slower but work in more cases.In both types, robustness metric R are used to guide the search. The final score is the combination of clean accuracy and robustness value.In A3D, mix multiple robustness evaluations. For example, a model is tested with both PGD and natural noise. Combine the scores to find the best model.Let the final score S be:

$$S = \alpha_1 * cleanAcc + \alpha_2 * RA_{PGD} - \alpha_3 * R_J \tag{5}$$

Here, α_1, α_2 and α_3 are weights.This flexible setup allows us to test models under real and artificial conditions.The purpose is to create attacks that reduces model accuracy while keeping the attack cost low. The system searches for the best attack configuration by adjusting different parameters automatically.In this setup, the attacker wants to generate adversarial examples that fool the model. These examples are created by adding small changes to the input images. The attacker seeks to reduce the model's prediction accuracy while keeping the changes are small and not visible to humans. This problem is defined as an optimization task. The optimization objective includes two terms. One term focuses on reducing accuracy. The second term minimizes the cost of the attack. The loss function is written as:

$$min_\theta\, Acc\left(F\left(x^{adv}(\theta)\right)\right) + \lambda * CF(x^{adv}(\theta), x) \tag{6}$$

Here, $x^{adv}(\theta)$ is the adversarial input created with parameter θ. The function F represents the model under attack. The term $Acc(\times)$ is the accuracy of the model when tested on the adversarial input. The function $C(\times)$ represents the cost or effort needed to create the adversarial input. The parameter λ is used to balance the two tasks.To find the best attacks, A3D explores a wide range of settings. These settings form the attack search space. The search space includes the type of attack for example FGSM, PGD or CW. It includes the choice of loss function. This includes cross-entropy loss, margin-based loss or confidence-based loss. Other important settings include perturbation strength, number of attack steps and step size. Some attacks use image transformations like Gaussian blur or brightness shift. Each possible combination of these settings is treated as a candidate in the search.The attacker tries not only to fool the model but to do so efficiently.Therefore, the attacker uses a multi-objective function. One part of the function objects to reduce the robust accuracy of the model. The other part measures the cost of the attack. The function is given as:

$$min_\theta\left[1 - RA\left(F, x^{adv}(\theta)\right), C(x^{adv}(\theta)\right] \tag{7}$$

The attacker is trying to find a solution that reduces the accuracy while keeping the cost small. A solution is better if it lowers the robust accuracy without requiring too much time or computation.A3D uses several optimization strategies to search for the best attacks. Random search is the most basic method. It selects parameters randomly. Bayesian optimization creates a model to estimate the performance of each setting and chooses the next setting based on uncertainty. It is more intelligent than random search. Genetic Algorithms (GA) are inspired by biological evolution. Operations like crossover and mutation are used to evolve better attack settings over time. Differential Evolution (DE) is another evolutionary method that uses differences between candidate solutions to guide the search. One of the most powerful methods used in A3D is NSGA-II. It is designed for multi-objective problems. It finds a set of solutions titled as the Pareto front. These solutions offer different trade-offs between success rate and cost.

After each candidate attack is tested, it is scored using two metrics. The first metric is robust accuracy. A lower score shows a stronger attack. The second metric is the attack cost. This includes computation time or the number of steps needed to create the adversarial input. These scores are used to guide the search toward better attacks.At the end of the search, A3D returns the best attack settings. These include the type of attack, the parameters like ò(perturbation size), number of steps, step size and any transformation functions used. This optimized attack is then used to test the models found by the NAS module. It helps improve model training by providing stronger adversarial examples. In this way, the AutoML for attack module plays a key role in making A3D more powerful and adaptive. The A3D system works by joining the attack and defense modules in a loop.

This loop helps both parts get better over time. The concept is simple: use strong attacks to test models and use strong models to challenge attacks. This creates a cycle in which both parts improve.In traditional methods, models and attacks are trained separately. This causes a gap. A3D solves this problem using a co-evolution strategy.Each time a model is searched by the NAS module, it is tested with attacks from the attack module. These attacks are not fixed. So, the models are always tested with the current best attacks. This forces the model search to find better and more robust architectures.At the same time, the attack module keeps updating its strategies. It uses the latest searched models to find weak points. Since these models are harder to fool, the attacks should be stronger and smarter. This feedback helps the attack module grow. As a result, the attacks become better with each round.Let us understand this loop with a simple example. Suppose at round t, the NAS module finds a model α_t with weights w_t.

This model is passed to the attack module. The attack module uses it to search for an attack θ_t. Then, θ_t is used to test α_t. If the model performs poorly, the NAS module will try to find a stronger model α_{t+1} in the next round. Similarly, α_{t+1} is then used to update the attack θ_{t+1}. This cycle continues.Mathematically, represent this loop as:

$$\alpha_{t+1} = argmin_{\alpha} \left[L_{val}(W^*(\alpha), \alpha) + R\left(W^*(\alpha), \alpha; \theta_t\right]\right. \quad (8)$$

$$\theta_{t+1} = argmin_{\theta}\left[1 - RA\left(F_{\alpha+1}, x^{adv}(\theta)\right) + \lambda * C(x^{adv}(\theta), x)\right] \quad (9)$$

Here, the defense updates depend on the latest attack. The attack updates depend on the latest defense. This creates a tight loop.To make this loop efficient, A3D uses a shared controller. This controller manages both searches. It stores past results, compares new solutions and updates priorities. This shared memory allows better coordination.Another key concept in A3D is the use of multi-objective optimization.The defense module wants high accuracy and high robustness. The attack module wants high success and low cost. These are always in conflict.To solve this, A3D uses Pareto optimization. A solution is Pareto-optimal if you cannot improve one objective without hurting another. A3D keeps a Pareto front of the best models and best attacks. At each step, only the best non-dominated solutions are kept. For example, a model that has high clean accuracy and high robust accuracy is preferred.

Another model with better robust accuracy but slightly lower clean accuracy might be kept as well. These models lie on the Pareto frontier. Similarly, for attacks keep a set of attacks that each trade-off success rate and cost differently.This strategy allows A3D to offer a diverse set of solutions. A3D runs a two-way loop. Models help improve attacks and attacks help improve models. Each side is optimized using multi-objective search. The system tracks the Pareto front for both parts. This helps create a balanced and strong framework for testing and improving deep learning systems.The A3D framework works as a full pipeline that brings the defense module and the attack module together. Each part is improved step by step in an alternating loop. In every round, a new model is searched. A new attack is found. This process repeats until no more improvements is made.Let us define a cycle of this loop. In round t, a model α_t and an attack θ_t are present. First, the defense module updates the model:

$$\alpha_{t+1} = argmin_{\alpha} \left[\left[L_{val}(W^*(\alpha), \alpha) + R\left(W^*(\alpha), \alpha; \theta_t\right]\right]\right. \quad (10)$$

Here, L_{val} is the validation loss, R is the robustness score and θ_t is the current attack setting. Use the newly found model α_{t+1}, the attack module updates its attack setting:

$$\theta_{t+1} = argmin_{\theta}\left[1 - RA\left(F_{\alpha+1}, x^{adv}(\theta)\right) + \lambda * C(x^{adv}(\theta), x)\right] \quad (11)$$

In this case, RA is the robust accuracy, C is the cost of the attack and λ is a trade-off factor.This sequence continues in rounds. Each update helps the other improve. This is termed as co-evolution.A3D uses a controller to manage the search. This controller schedules when to search for a new model or a new attack. It stores the history of all trials and uses it to guide next steps.Let S_t^D and S_t^A represent the current search states for defense and attack, respectively. The controller uses a simple rule to schedule updates. For example:

$$if\ \Delta S_t^D < o'_D \text{ then switch to attack module} \quad (12)$$

$$if\ \Delta S_t^A < o'_A \text{ then switch to defense module} \quad (13)$$

Here, ΔS is the change in performance and ò is the early-stopping threshold. If there is no improvement, the system switches tasks.The controller ranks past results using a score function:

$$score = \alpha * \beta * RobustAcc - \gamma * cost \quad (14)$$

This score is used to select the best candidates from history. It helps adjust the search space by focusing on successful regions.A3D works with many datasets. Each dataset has different properties like image size, number of classes and complexity. The system adjusts its pipeline to fit each case.For small datasets like CIFAR-10, the system trains models for more epochs. The full training loop is allowed because the data is small. For large datasets like ImageNet, the system trains on proxy data. A proxy dataset D' is defined as:

$$D' = \{(x_i, y_i)\epsilon\ D/\ i\epsilon\ subset\} \quad (15)$$

This reduces training time. Later, the best model found using is fine-tuned or D'validated on the full dataset D.The NAS and attack search changes depending on dataset size. For CIFAR-10, A3D allows a larger search space:

$$SearchSpace_{CIFAR} = \{Depth \leq 20, Width \leq 128\} \quad (16)$$

For ImageNet, the search space is smaller to save resources:

$$SearchSpace_{ImageNet} = \{Depth \leq 10, Width \leq 64\} \quad (17)$$

Likewise, attack settings are adjusted. For small datasets, it is possible to run more complex attacks.For large ones, simple attacks are preferred during search and more complex attacks are used only for final testing.The attack cost C is calculated based on time and resources. For example:

$$C(x^{adv}, x) = \eta_1 * QueryCount + \eta_2 * Time + \eta_3 * (x^{adv} - x) \quad (18)$$

Here, η_1, η_2 and η_3 are weights for balancing cost dimensions.The controller uses all this information to adapt the search process. A3D pipeline is dynamic. It alternates between attacker and defender updates. A smart controller makes decisions based on performance. The system supports multiple datasets and adapts its search space and training based on the task. With well-designed equations and feedback loops, the system grows smarter over time.The A3D framework is a powerful system. It solves two problems at the same time. First, it finds neural networks that are strong against adversarial attacks.

Second, it finds smart attacks that breaks weak models. The system does both using AutoML. It learns and adapts with time.A3D works in a loop. This loop has two main modules. The first module creates models. The second module creates attacks. Each time a model is created, it is tested by the best attack. Each time a new attack is created, it is tested on the best model. This makes both parts stronger.The system uses many optimization methods. The model part uses NAS. The attack part uses algorithms like NSGA-II. It tests different ways to fool the models. Both modules use multiple objectives. For example, the model module wants both high clean accuracy and high robust accuracy. It solves the following problem:

$$min_{\alpha} L_{val}\big((W^*(\alpha), \alpha) + \gamma * R(W^*(\alpha), \alpha)\big) \quad (20)$$

The attack module wants to reduce robust accuracy while keeping cost low. It solves ad given in equation (6). Both parts use Pareto optimization. The system keeps a Pareto front. From this, users select the best trade-off needed.A3D works on different datasets. For small datasets like CIFAR-10, it runs deep searches. For large datasets like ImageNet, it runs fast proxy searches. The controller adjusts search settings based on the task. It sets training epochs, model depth, image size and more. This makes A3D flexible.The A3D framework is smart. It does not use fixed rules. Instead, it learns from each round. It knows what worked and what failed. It stores this knowledge. This concept is named as co-evolution. The attacker learns from the defender.

The defender learns from the attacker. Both get stronger with time. This makes the system different from past tools. Most other platforms test attacks and defenses separately. A3D joins them in one cycle.The controller plays a key role. It decides when to stop a search. It ranks solutions based on clean accuracy, robust accuracy and attack cost. It uses the score which is given in equation (12).A3D help test and improve them.It helps in building strong facial recognition systems. These systems are frequently targeted by attacks. A3D is used to find weak spots and fix them.Researchers use A3D as a benchmark tool. It gives fair tests for both models and attacks. It removes bias. It gives detailed results under many conditions.

Developers use A3D in industry. The system is automatic. It needs less manual work. It learns fast and adapts. This saves time and money.A3D is a smart and complete framework. It solves a hard problem in deep learning. It does not just test models or attacks. It improves them together. It works in a loop. It learns from its past. It adapts to each task.It uses many powerful tools. These include NAS, evolutionary algorithms, Pareto search and co-evolution. The result is a system that gets stronger over time.A3D works across datasets. It adjusts settings and resources as needed. It is smart, flexible and useful. It helps make AI systems safer and stronger. It makes research and testing faster and fairer.This framework marks an important step. It shows that AutoML is useful not just for building better models, but for creating stronger attackers. It shows that both improve each other within one smart system.

Algorithm 1: NSGA-II for Efficient Adversarial Attacks.

1. Initialize population P with N attack candidates
2. Evaluate fitness: robustness R(x) and time cost T(x) for each $x \in P$
3. for generation g = 1 to G do
4. Perform non-dominated sorting on P
5. Assign crowding distance to individuals
6. Select parents using tournament selection
7. Apply crossover and mutation to generate offspring Q
8. Evaluate R(x) and T(x) for each $x \in Q$
9. Combine P and Q into population R
10. Perform non-dominated sorting on R
11. Select the best N individuals to form next P
12. end for
13. Return Pareto front of final population

Source: Authors, (2026).

IV. EXPERIMENTS

This section describes the experimental results of the A3D framework. All experiments are performed on standard image classification datasets like CIFAR10, CIFAR100 and ImageNet. In the first part, the experiments focus on AutoML for adversarial attack. The objective here is to search for strong and efficient adversarial attack strategies. In specific, the framework is tested on models that is already been trained using different adversarial defense techniques.

Two types of norm-based attacks are considered: the ℓ_∞ norm and the ℓ_2 norm. The purpose is to compare the attacks discovered by the A3D framework with commonly used manual attack methods like FGSM, PGD and CW.The Table 1 shows the results of single-objective optimization using the ℓ_∞ norm. This test was conducted on 500 images from the CIFAR10 dataset. The table lists different attack methods along with the corresponding robust accuracy and time cost. The robust accuracy is the accuracy of a model under attack and a lower value indicates a stronger attack. The time cost shows the duration required to generate the attack.

Table 1: Comparison under ℓ_∞ Attack (500 CIFAR10 Images).

Attack Type	Robust Accuracy (%)	Time Cost (s)
FGSM	62.1	5
PGD	48.5	35
CW	44.3	60
MI	45.7	58
A3D- Local Search	42.8	42
A3D-DE	36.6	55
A3D-NSGA-II	38.2	65

Source: Authors, (2026).

From the Table 1, it is clear that the DE algorithm within A3D finds the most effective attack, achieving the lowest robust accuracy at 36.6%. The attack fool the model more effectively than others. Although CW and MI are strong attacks, the A3D-based search finds combinations that outperform them.In Table 2 multi-objective optimization is considered. Here, both robustness and time efficiency are important. The following table shows the comparison using the ℓ_2 norm with 500 CIFAR10 images.

Table 2: Multi-Objective Attack Comparison (ℓ_2 norm, 500 Images)

Algorithm	Robust Accuracy (%)	Time Cost (min)
Random Search	42.7	22
A3D-NSGA-II	39.5	26

Source: Authors, (2026).

This Table 2 shows that A3D's NSGA-II algorithm find stronger attacks with slightly more time cost than random search. It demonstrates that a smart search strategy improves both attack quality and efficiency.The second major part of the experiments focuses on AutoML for adversarial defense. The intention is to automatically search for model architectures that are robust to different types of perturbations. These include adversarial, natural and system noises. Eleven NAS algorithms are evaluated, both differentiable and non-differentiable. The robustness is measured by the accuracy of the model under attack.The following Table 3 presents the results for models trained on adversarial noise using the FGSM attack as evaluation. It includes the final robust accuracy and model size in terms of the number of parameters.

Table 3: Model Robustness on Adversarial Noises (CIFAR10, $\grave{o} = 1/255$)

Model	Robust Accuracy (%)	Params (M)
DARTS	50.3	3.4
Random Search	48.2	3.3
SmoothDARTS	52.6	3.5
DE_NAS	51.8	3.6
ResNet-18 (manual)	43.5	11.2

Source: Authors, (2026).

SmoothDARTS achieved the best robustness, showing that differentiable NAS methods outperform traditional hand-crafted models like ResNet-18. The architectures discovered by A3D were smaller in size, making them more efficient.Table 4 shows the interaction between attack and defense processes in a loop.First, it shows that better attacks help discover stronger models. When A3D's adversarial attack (AAA) is used to evaluate models during NAS, the resulting architectures are more robust. The Table 4 compares the accuracy under AAA when different search strategies are used.

Table 4: Effect of AAA on NAS (CIFAR10, $\grave{o} = 1/255$)

Search Strategy	Robust Accuracy under AAA (%)
Random_Search_FGSM	41.6
Random_Search_AAA	49.4

Source: Authors, (2026).

TheTable 4 shows that using stronger attack methods as evaluation metrics leads to finding more robust architectures. The robust accuracy improved by nearly 8%.Finally, the framework was tested in an iterative loop, with attack and defense modules running in sequence over multiple cycles. In each cycle, a new model is discovered and used to find even stronger attacks. This ongoing process helps both modules improve continuously. The improvements in accuracy and attack strength over multiple iterations demonstrate the power of the A3D framework. The experimental results clearly demonstrate the effectiveness of the A3D framework.

It automatically discovers both strong adversarial attacks and robust model architectures. Using these modules together in a loop improves both model performance and adversarial resistance.

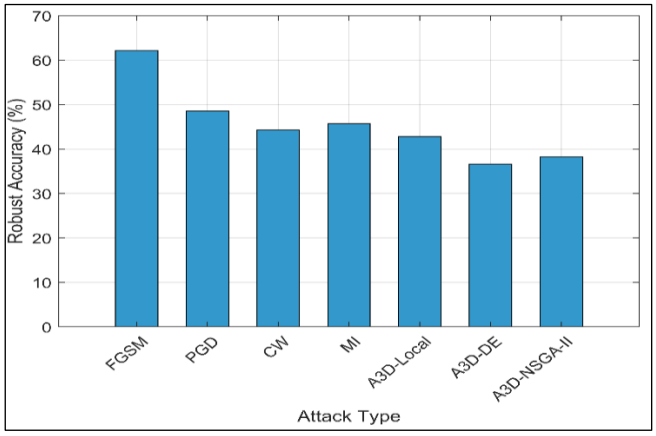

Figure 2: Robust Accuracy versus Attack Type.
Source: Authors, (2026).

The Figure 2 shows the robust accuracy of a model under different types of adversarial attacks. FGSM has the highest robust accuracy at around 64%. PGD follows with about 49%. CW results in a slightly lower accuracy near 46%. MI yields a robust accuracy of around 44%, indicating a stronger attack compared to FGSM and PGD. The attacks found by the A3D system are shown next starting with A3D-Local. It gives a robust accuracy near 47%. This shows it is more effective than PGD but slightly weaker than MI. Among the most powerful attacks are A3D-DE and A3D-NSGA-II. A3D-DE brings the robust accuracy down to about 37%. A3D-NSGA-II reduces it to around 38%. These are the most successful in fooling the model with A3D-DE being the strongest. The overall results shows that attacks designed using A3D expressly with DE and NSGA-II strategies are more efficient than standard gradient-based attacks. The drop in robust accuracy under these attacks proves that A3D-generated attacks are harder to defend. The difference of over 25% between FGSM and A3D-DE highlights the increased danger posed by A3D attacks.

The Figure 3compares the time cost of various attack methods. FGSM is the fastest, needing only about 5 seconds. This speed comes from its simple one-step gradient method. PGD takes longerabout 35 secondsbecause it runs multiple steps. CW takes even more time, around 60 seconds due to its optimization-based method. MI uses iterative updates and records a time cost of nearly 55 seconds. These three attacks show that stronger and more precise attacks tend to consume more time.The A3D-based attacks follow a similar trend. A3D-Local takes about 42 seconds. It is faster than CW and MI but slower than PGD. A3D-DE uses differential evolution and takes about 55 seconds. A3D-NSGA-II is the most time-consuming, at around 66 seconds because it handles multiple objectives. This shows that as A3D attacks become more advanced, the time cost increases. More powerful search strategies result in better attack performance while causing higher computational load.The chart shows this balance clearly.

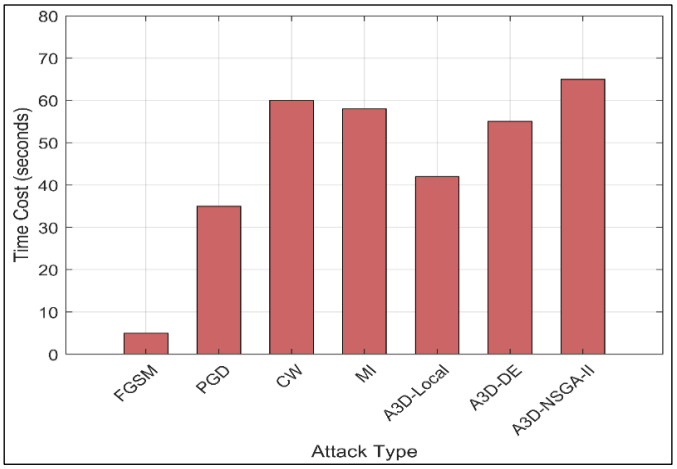

Figure 3: Time Cost versus Attack Type.
Source: Authors, (2026).

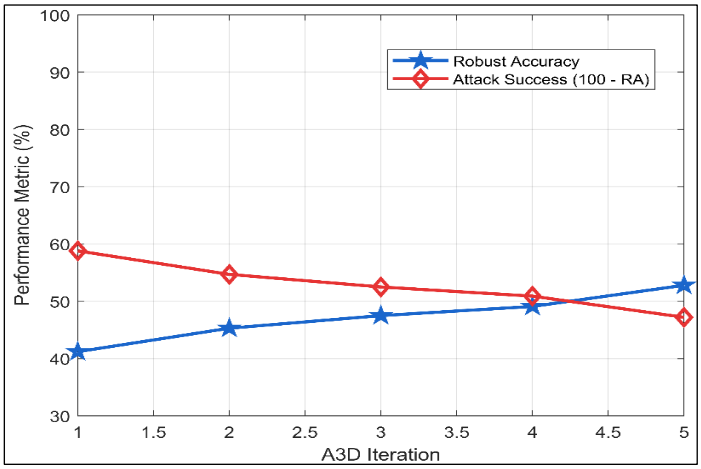

Figure 4: Iteration versus Performance in A3D Framework.
Source: Authors, (2026).

The Figure 4 shows two performance metrics Robust Accuracy and Attack Success change over five A3D iterations. At Iteration 1, the Robust Accuracy is around 42% and it gradually increases across the iterations, reaching approximately 53% by Iteration 5. This shows that the defense component of the A3D framework improves with each cycle. It makes the model more robust to adversarial attacks. Attack Success starts high at about 58% in Iteration 1 but steadily drops to around 47% by Iteration 5. This decrease in Attack Success mirrors the increase in Robust Accuracy. In Iteration 4, both curves intersect that suggests a turning point at which the defense performance begins to overtake the attack strength. The chart shows steady improvement in robust accuracy. Attack success drops over time. This shows the A3D loop is working well. Strong attacks help create better models.

Strong models push the system to find smarter attacks. Over five rounds, models get more robust. Attacks become less effective. This proves the A3D method works as designed. The Figure 5 illustrates a steady improvement in robust accuracy for three architecture search methods: DARTS, DE_NAS and SmoothDARTS. In the early phase, all three methods start from a similar point just above 40% with DARTS slightly behind the others. However, as each A3D cycle progresses, the paths begin to diverge. SmoothDARTS quickly establishes itself as the leading method gaining momentum with every iteration. By the third iteration, it has already moved ahead of DE_NAS and maintains this advantage through to the final cycle.

DARTS improves steadily, but its slope is shallower compared to the other two. It indicates slower growth in robustness. By the final iteration, the contrast between the models becomes clear. SmoothDARTS tops the chart at around 54.2% robust accuracy, followed closely by DE_NAS at 53.6%. DARTS, while improved ends at a lower value of approximately 49.4%. The gap between DARTS and other methods keeps growing. This shows DARTS improves slower. DE_NAS and SmoothDARTS adapt better to attacks. These designs help find stronger models. All methods get better with A3D tuning. But some improve faster. The search algorithm matters. A good design leads to quicker and stronger results. SmoothDARTS emerges as the most promising candidate for building resilient neural networks under adversarial stress.

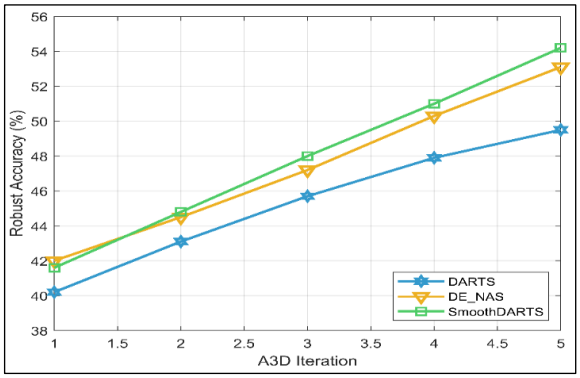

Figure 5: Robust Accuracy across A3D Iteration.
Source: Authors, (2026).

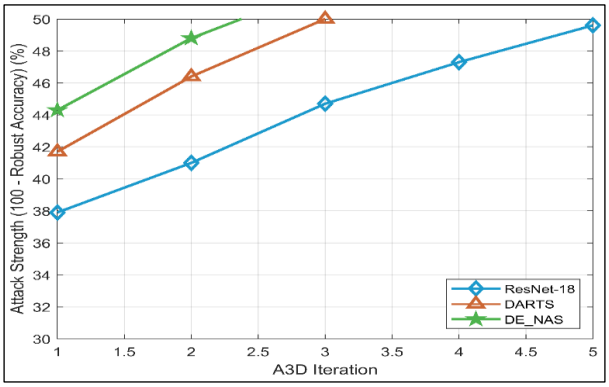

Figure 6: A3D Iteration versus Attack Strength.
Source: Authors, (2026).

The Figure 6 shows a line graph comparing the attack strength of three different models ResNet-18, DARTS and DE_NAS across five A3D iterations. The ResNet-18 model starts with low strength. In iteration 1, it is around 38%. It improves step by step. By iteration 5, it reaches just under 50%. The progress is steady. The model becomes stronger with each round. As stronger attacks are discovered, the ResNet-18 model becomes more vulnerable.DARTS begins at a higher attack strength of about 42% in iteration 1 and rises quickly, reaching approximately 50% by iteration 4.

After that, it shows no major change suggesting that it has reached a peak in vulnerability under the current attack strategies. DE_NAS shows the highest attack strength among the three from the start. It begins around 44% and rising sharply to 49% by iteration 3. It seems to plateau after that similar to DARTS. The graph shows a clear trend. DARTS and DE_NAS become weaker in early rounds. Robustness is lost quickly.ResNet-18 declines more slowly. But its drop is steady. Attacks are becoming stronger. The A3D system learns with each step. It finds weak spots in all models. Even strong models get exposed over time.

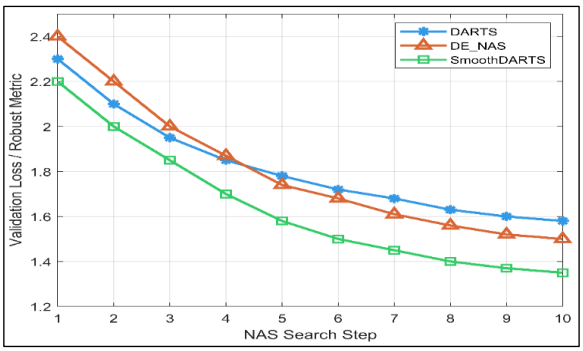

Figure 7: NAS Search Step versus Validation loss or Robustness Metric.
Source: Authors, (2026).

The Figure 7 visualizes the changes in validation loss or a robustness metric across ten NAS search steps for three architecture search methods.All three lines demonstrate a downward slope, indicating continuous improvement over time. DARTS begins at a relatively high metric value near 2.3 and declines slowly to finish just above 1.6. DE_NAS starts high at around 2.4. It catches up with DARTS midway. Later, it drops to about 1.5. SmoothDARTS begins near 2.2. It stays ahead in every step. By step ten, it reaches nearly 1.3. This shows SmoothDARTS starts stronger.

It improves faster over time. DE_NAS shows a similar but slightly slower improvement path, remaining reliably better than DARTS. The lines move increasingly apart as the search continues. This shows the search method matters more over time. All three methods get better with each step. But some improve more than others. SmoothDARTS performs the best. DE_NAS comes next. DARTS improves the least. The growing gap proves the impact of search efficiency.

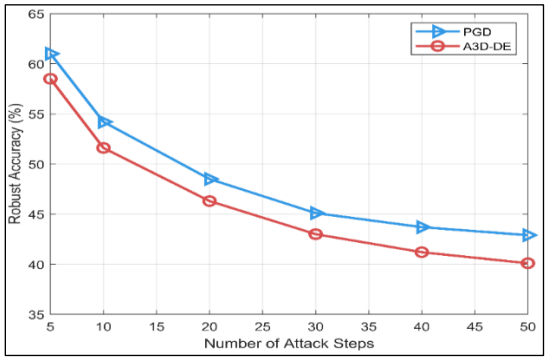

Figure 8: Attack Steps versus Robust Accuracy.
Source: Authors, (2026).

The Figure8 shows a line graph comparing the robust accuracy of two attack methods as the number of attack steps increases. At 10 steps, PGD starts at around 61% robust accuracy.While A3D-DE begins slightly lower at about 59%. As attack steps reach 20, accuracy drops for both methods. PGD falls to around 49%. A3D-DE drops to about 46%. The decline continues across the graph. More attack steps mean stronger attacks. Stronger attacks make the model weaker. The analysis shows higher vulnerability with more iterations. By 50 steps, the robust accuracy under PGD reaches roughly 43%.While A3D-DE drops to about 40%. Across all step sizes, A3D-DE constantly results in lower robust accuracy compared to PGD. This indicates that A3D-DE is more effective in breaking model defenses expressly at higher iteration counts. The gap between the two curves remains steady. This shows that both attacks get stronger with more steps. A3D-DE is always more effective than PGD. Both lines go down smoothly. This shows more attack steps reduce model accuracy. A3D-DE lowers the accuracy more than PGD. The figure proves that A3D-DE is better at breaking the model over many step sizes.

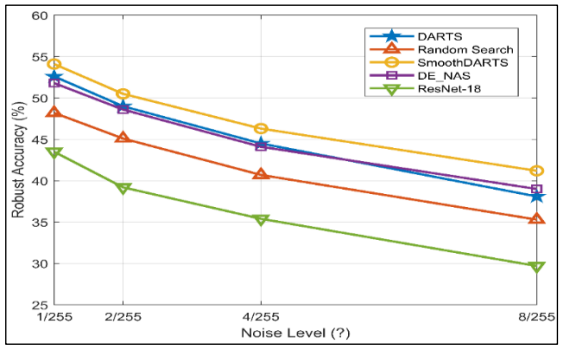

Figure 9: Robust Accuracy versus Noise Level.
Source: Authors, (2026).

The Figure 9 compares the robust accuracy of five models under different noise levels: 1/255, 2/255, 4/255 and 8/255. The models are DARTS, Random Search, SmoothDARTS, DE_NAS and ResNet-18. At the lowest noise level (1/255), SmoothDARTS performs the best with about 54% accuracy. DARTS and DE_NAS follow closely at around 52%.While Random Search scores 49% and ResNet-18 trails at 43%. As the noise level increases to 2/255, the accuracy of all models drops. SmoothDARTS remains on top at 50%, DARTS and DE_NAS are near 48%. Random Search falls to 45% and ResNet-18 declines to 39%.At 4/255 noise, SmoothDARTS holds the lead with 47% accuracy. DE_NAS and DARTS stay close at 44%, Random Search drops to 41% and ResNet-18 falls to 35%. When the noise reaches the highest level of 8/255, SmoothDARTS still performs best with 41%. DE_NAS scores just under 40% and DARTS ends at 38%. Random Search decreases to 35%, while ResNet-18 has the lowest accuracy of 30%. SmoothDARTS shows the best resistance to noise and ResNet-18 is the most affected. All models show steady performance decline as the noise level increases.

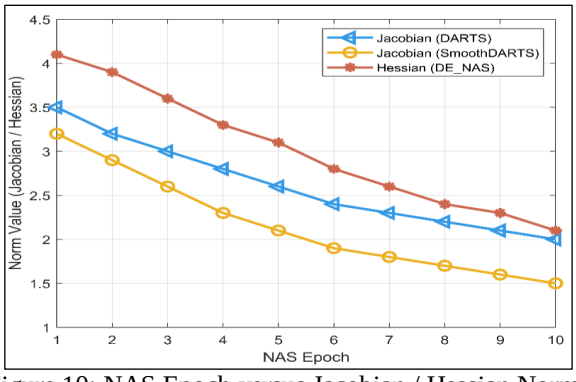

Figure 10: NAS Epoch versus Jacobian / Hessian Norm.
Source: Authors, (2026).

The Figure 10 shows the norm values of Jacobian and Hessian change over 10 NAS epochs. Three methods are shown: Jacobian (DARTS), Jacobian (SmoothDARTS) and Hessian (DE_NAS). At epoch 1, Hessian (DE_NAS) starts the highest with a norm value slightly over 4. Jacobian (DARTS) begins around 3.5 and Jacobian (SmoothDARTS) starts near 3.2. As the epochs increase, the values for all three methods steadily decrease.By epoch 5, Hessian (DE_NAS) has dropped to about 3.2. Jacobian (DARTS) reaches near 2.6 and Jacobian (SmoothDARTS) drops to about 2.1. At the final epoch 10, Hessian (DE_NAS) settles just below 2.5, Jacobian (DARTS) ends near 2. Jacobian (SmoothDARTS) goes down to about 1.5. Throughout all epochs, Jacobian (SmoothDARTS) has the lowest norm values, indicating smoother gradients. Hessian (DE_NAS) maintains the highest norm values, suggesting more variation in the learning dynamics. The overall pattern is a steady decline for all methods, showing that training reduces gradient magnitudes over time.

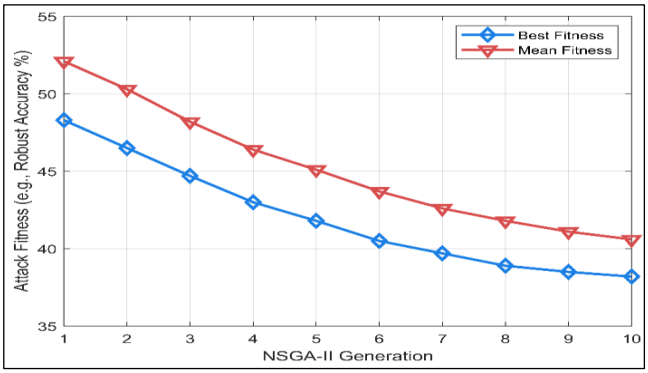

Figure 11: Search Generation versus Attack Fitness in NSGA-II.
Source: Authors, (2026).

TheFigure 11 illustrates attack fitness, represented by robust accuracy, changes over ten generations using the NSGA-II algorithm. It compares two metrics: Best Fitness and Mean Fitness. In generation 1, the Best Fitness begins at around 48% and the Mean Fitness starts higher at nearly 53%. As the generations move forward, both lines gradually decline. By generation 5, Best Fitness drops to about 43%.While Mean Fitness reaches roughly 46%. This pattern shows that as the optimization process continues, both the strongest and average solutions become less robust.In the final generations, the decline continues but slows slightly. At generation 10, Best Fitness ends near 40% and Mean Fitness closes around 41%. The gap between the two lines narrows, indicating that individual models in the population are becoming more similar in robustness. This suggests that the algorithm is converging to a certain performance level. The plot shows that robustness decreases over time. NSGA-II finds weaker architectures in later generations. However, the best models still perform better than the average. This holds true even in the last few generations.

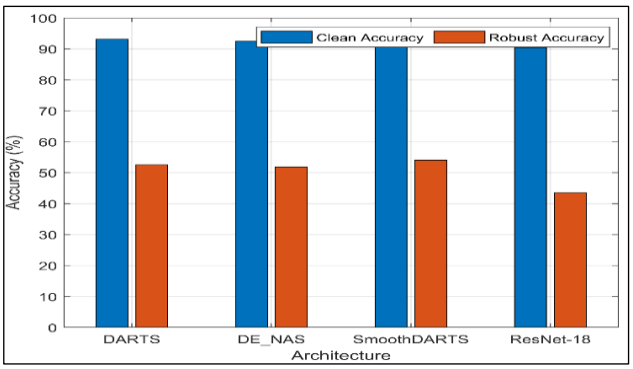

Figure 12: Clean versus Robust Accuracy by Architecture.
Source: Authors, (2026).

The Figure 12 compares clean accuracy and robust accuracy for four neural network architectures. All four models show clean accuracy values around 93%. Specifically, DARTS has the highest clean accuracy, slightly above 93%, while the others are close behind. ResNet-18 has the lowest clean accuracy, though the difference is small. This suggests that all models perform well on clean, unperturbed data. The robust accuracy tells a different story. SmoothDARTS achieves the highest robust accuracy, close to 54%, followed by DARTS at around 53%. DE_NAS is slightly behind at 52% and ResNet-18 performs the worst with only 44%. The gap between clean and robust accuracy is the smallest for SmoothDARTS. It indicates that it handles adversarial or noisy conditions better. ResNet-18 shows a larger drop in performance when moving from clean to robust scenarios. The chart highlights that although most models are similar in clean accuracy. SmoothDARTS provides the best trade-off between clean and robust performance.

The Figure 13 compares the GPU time in days required by different NAS methods. Among them DE_NAS has the highest search cost using about 3.2 GPU-days. SmoothDARTS is next requiring around 2.1 GPU-days. DARTS needs less time using 1.5 GPU-days. Random Search takes only 0.8 GPU-days and PC-DARTS uses around 1.0 GPU-days. These numbers show that some methods are much more demanding in terms of computational cost than others.From the figure, it is clear that DE_NAS is the most resource-heavy method.While Random Search is the least costly. SmoothDARTS and DARTS sit somewhere in the middle. PC-DARTS, although more efficient than DE_NAS still uses slightly more resources than Random Search.

This suggests a trade-off between cost and potential performance. Users with limited hardware or time should consider lower-cost options like Random Search. Meanwhile, those with more resources might choose DE_NAS for possibly better results despite the higher cost.

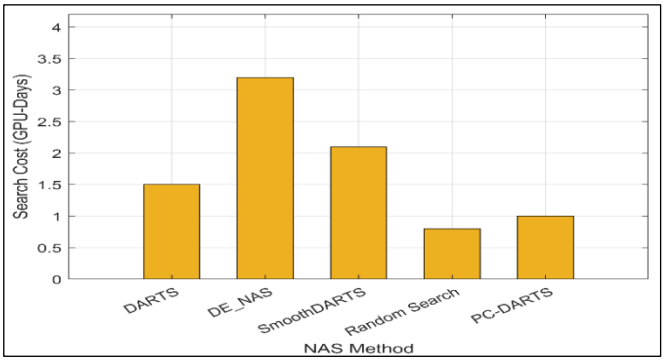

Figure 13: Search Cost versus NAS Method.
Source: Authors, (2026).

V. CONCLUSION

The main purpose of A3D is to automatically search for both strong adversarial attacks and robust neural network architectures. It brings together two important areas in adversarial machine learning attack generation and defense design.Most existing tools are limited to evaluating models or attacks. These tools do not improve or optimize the results.A3D solves this problem by using AutoML techniques. These techniques help in finding better model designs and better attack methods without human effort. The platform is powerful because it supports both single-objective and multi-objective optimization. This tells it finds results that are either very strong, very fast or both.For the defense part, A3D uses different types of robustness tests. These include adversarial noise, natural noise, system noise and mathematical metrics like the Jacobian and Hessian.

The system then searches for models that perform well under these tough conditions. It uses various NAS methods to do this.For the attack part, A3D uses search algorithms like Differential Evolution and NSGA-II. These help in finding strong attacks that fools even the most robust models. Attacks found by the system are used to test and improve models. The improved models are used to find even better attacks. This cycle continues and makes both parts stronger over time. This platform helps researchers build safer and smarter AI systems.A3D is a complete and automatic solution for improving AI robustness. It makes it easier to build models that are hard to attack and to test them with strong attacks.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Dr.Sarala Patchal, Rajani Bodapalli, Dr. Vullam Naga Gopi Raju, Desamala Prabhakara Rao, Dr.Vijay Babu Burra and Dr. D. Kishore.
Methodology: Dr. Sarala Patchala and Rajani Bodepalli
Investigation: Dr. Sarala Patchala and Rajani Bodepalli
Discussion of results:. Dr. Sarala Patchala and Dr. Vijay Babu Burra Vulla, Dr. Vullam Naga Gopi Raju.
Writing – Original Draft: Dr. Sarala Patchala
Writing – Review and Editing: Dr. Sarala Patchala and Dr. Vijay Babu Burra
Resources: Dr. Vijay Babu Burra
Supervision: Dr. Vijay Babu Burra Vullam Nagagopiraju
Approval of the final text: Dr. Sarala Patchala, Rajani Bodapalli Dr. Vijay babu Burra Vulla, Nagagopiraju, Desamala Prabhakara Rao and Dr. D. Kishore.

VII. REFERENCES

[1] A. A. Khan, A. A. Laghari, and S. A. Awan, "Machine learning in computervision: A review.," EAI Endorsed Transactions on Scalable InformationSystems, vol. 8, no. 32, 2021.

[2] Z. Zhang and A. Z. Kouzani, "Implementation of dnns on iot devices,"Neural Computing and Applications, vol. 32, no. 5, pp. 1327–1356, 2020.

[3] X. Ding, S. Zhang, M. Song, X. Ding, and F. Li, "Toward invisible adversarial examples against dnn-based privacy leakage for internet of things,"IEEE Internet of Things Journal, vol. 8, no. 2, pp. 802–812, 2020.

[4] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theoriesand applications for self-driving cars based on deep learning methods,"Applied Sciences, vol. 10, no. 8, p. 2749, 2020.

[5] S. Hu, T. Yu, C. Guo, W.-L. Chao, and K. Q. Weinberger, "A new defenseagainst adversarial images: Turning a weakness into a strength," Advancesin neural information processing systems, vol. 32, 2019.

[6] C. Yang, Q. Wu, H. Li, and Y. Chen, "Generative poisoning attack methodagainst neural networks," arXiv preprint arXiv:1703.01340, 2017.

[7] J. Wang and H. Zhang, "Bilateral adversarial training: Towards fast training of more robust models against adversarial attacks," in Proceedings of theIEEE/CVF international conference on computer vision, pp. 6629–6638,2019.

[8] D. J. Miller, Z. Xiang, and G. Kesidis, "Adversarial learning targeting deepneural network classification: A comprehensive review of defenses againstattacks," Proceedings of the IEEE, vol. 108, no. 3, pp. 402–433, 2020.

[9] J. Liu, J. C. Wyatt, and D. G. Altman, “Decision tools in health care: focuson the problem, not the solution,” BMC Medical Informatics and DecisionMaking, vol. 6, pp. 1–7, 2006.

[10] N. C. Thompson, K. Greenewald, K. Lee, G. F. Manso, et al., “The computational limits of deep learning,” arXiv preprint arXiv:2007.05558, vol. 10,2020.

[11] E. LeDell and S. Poirier, “H2o automl: Scalable automatic machine learning,” in Proceedings of the AutoML Workshop at ICML, vol. 2020, p. 24,2020.

[12] L. Liao, H. Li, W. Shang, and L. Ma, “An empirical study of the impact ofhyperparameter tuning and model optimization on the performance properties of deep neural networks,” ACM Transactions on Software Engineeringand Methodology (TOSEM), vol. 31, no. 3, pp. 1–40, 2022.

[13] C. Xue, M. Hu, X. Huang, and C.-G. Li, “Automated search spaceand search strategy selection for automl,” Pattern Recognition, vol. 124,p. 108474, 2022.

[14] K. M. Passino, “Bridging the gap between conventional and intelligentcontrol,” IEEE Control Systems Magazine, vol. 13, no. 3, pp. 12–18, 2002.

[15] Z. Zhao, G. Chen, J. Wang, Y. Yang, F. Song, and J. Sun, “Attack asdefense: Characterizing adversarial examples using robustness,” in Proceedings of the 30th ACM SIGSOFT International Symposium on SoftwareTesting and Analysis, pp. 42–55, 2021.

[16] H. Yang, C. Yuan, B. Li, Y. Du, J. Xing, W. Hu, and S. J. Maybank,“Asymmetric 3d convolutional neural networks for action recognition,” Pattern recognition, vol. 85, pp. 1–12, 2019.

[17] I. J. Goodfellow et al., “Explaining and harnessing adversarial examples,”arXiv:1412.6572, 2015.

[18] A. Madry et al., “Towards deep learning models resistant to adversarial attacks,”arXiv:1706.06083,2018.

[19] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in IEEE S&P, 2017.

[20] Y. Dong et al., “Boosting adversarial attacks with momentum,” in CVPR, 2018.

[21] S. Gowal et al., “An alternative surrogate loss for PGD-based adversarial testing,” arXiv:1910.09338,2019.

[22] N. Papernot et al., “Distillation as a defense to adversarial perturbations,” in IEEE S&P, 2016.

[23] C. Guo et al., “Countering adversarial images using input transformations,” arXiv:1711.00117, 2018.

[24] M. Guo et al., “When NAS meets robustness: In search of robust architectures,” in CVPR, 2020.

[25] N. Papernot et al., “CleverHans v2.1.0: An adversarial ML library,” arXiv:1610.00768, 2018.

[26] J. Rauber et al., “Foolbox: A python toolbox for adversarial robustness,” arXiv:1707.04131, 2017.

[27] M.-I. Nicolae et al., “Adversarial robustness toolbox,” arXiv:1807.01069, 2018.

[28] Y. Dong et al., “Benchmarking adversarial robustness on image classification,” in CVPR, 2020.

[29] J. Morris et al., “Textattack: A framework for adversarial attacks in NLP,” in EMNLP, 2020.

[30] D. Z ̈ugner et al., “Adversarial attacks on graph neural networks,” in KDD, 2018.

[31] H. Liu et al., “DARTS: Differentiable architecture search,” in ICLR, 2018.

[32] Y. Xu et al., “PC-DARTS: Partial channel connections,” in ICLR, 2020.

[33] X. Chu et al., “FairDARTS: Eliminating unfair advantages,” in ICLR, 2021.

[34] R. Hosseini et al., “DSRNA: Differentiable robust NAS,” in CVPR, 2021.

[35] J. Mok et al., “AdvRush: Robust architecture search with Hessian metrics,” arXiv:2108.01289, 2021.

[36] H. Pham et al., “Efficient NAS via parameter sharing,” in ICML, 2018.

[37] R. Luo et al., “Neural architecture optimization,” in NeurIPS, 2018.

[38] H. Shi et al., “Bridging sample-based and one-shot NAS,” in NeurIPS, 2019.

[39] X. Mao et al., “Composite adversarial attacks,” in AAAI, 2022.

[40] C. Yao et al., “Adaptive attacks via AutoML,” arXiv:2102.11860, 2022.

[41] Q. Fu et al., “Automated decision-based adversarial attacks,” in IJCAI, 2021.

[42] F. Croce and M. Hein, “RobustBench: Adversarial robustness benchmark,” arXiv:2006.04884, 2020.