# LEAP: Efficient and Automated Test Method for NLP Software

Mingxuan Xiao
*College of Computer and Information*
*Hohai University*
Nanjing, China
xiaomx@hhu.edu.cn

Yan Xiao
*School of Cyber Science and Technology*
*Shenzhen Campus of Sun Yat-sen University*
Shenzhen, China
xiaoy367@mail.sysu.edu.cn

Hai Dong
*School of Computing Technologies*
*RMIT University*
Melbourne, Australia
hai.dong@rmit.edu.au

Shunhui Ji
*College of Computer and Information*
*Hohai University*
Nanjing, China
shunhuiji@hhu.edu.cn

Pengcheng Zhang*
*College of Computer and Information*
*Hohai University*
Nanjing, China
pchzhang@hhu.edu.cn

2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 979-8-3503-2996-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ASE56229.2023.00052

***Abstract*—The widespread adoption of DNNs in NLP software has highlighted the need for robustness. Researchers proposed various automatic testing techniques for adversarial test cases. However, existing methods suffer from two limitations: weak error-discovering capabilities, with success rates ranging from 0% to 24.6% for BERT-based NLP software, and time inefficiency, taking 177.8s to 205.28s per test case, making them challenging for time-constrained scenarios.**

**To address these issues, this paper proposes LEAP, an automated test method that uses LEvy flight-based Adaptive Particle swarm optimization integrated with textual features to generate adversarial test cases. Specifically, we adopt Levy flight for population initialization to increase the diversity of generated test cases. We also design an inertial weight adaptive update operator to improve the efficiency of LEAP's global optimization of high-dimensional text examples and a mutation operator based on the greedy strategy to reduce the search time.**

**We conducted a series of experiments to validate LEAP's ability to test NLP software and found that the average success rate of LEAP in generating adversarial test cases is 79.1%, which is 6.1% higher than the next best approach ($PSO_{attack}$). While ensuring high success rates, LEAP significantly reduces time overhead by up to 147.6s compared to other heuristic-based methods. Additionally, the experimental results demonstrate that LEAP can generate more transferable test cases and significantly enhance the robustness of DNN-based systems.**

***Index Terms*—NLP Software Testing, Particle Swarm Optimization**

## I. Introduction

In the field of NLP, Deep Neural Networks (DNNs) (*e.g.*, ELMo [1], BERT [2], GPT [3], T5 [4]) have been developing rapidly in recent years. These networks are capable of extracting semantic, structural, and other information from text and have been widely integrated as new software components in safety-critical systems like market monitoring [5], code review [6], and intelligence analysis [7]. Such systems are referred to as *DNN-based systems*. To address issues caused by malicious inputs, the software engineering (SE) community has proposed various techniques, including test coverage [8]–[10], fuzz testing [11]–[13], and automated user interface testing [14]–[16]. However, unlike software development methods that follow lifecycle frameworks [17], [18], DNN-based systems do not require developers to design the system's rules. Instead, they rely on DNNs learning from large amounts of data to make decisions, which makes it challenging to ensure the robustness of DNN-based systems using traditional software testing methods. Moreover, recent studies [19], [20] have shown that DNN-based systems have significant robustness pitfalls due to the uninterpretability of such systems and the complexity of training data, as demonstrated by the following scenario.

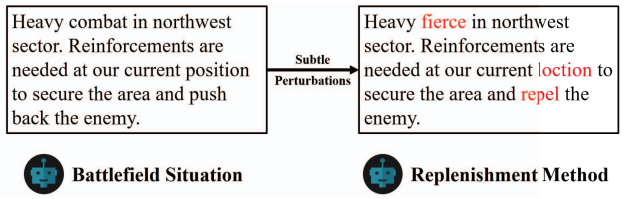


Fig. 1. Subtle perturbed text (red) misleads military intelligence analysis systems to judge text labels from "Battlefield Situation" to "Replenishment Method".

As shown in Fig. 1, the military intelligence analysis system is crucial to military information construction. It must classify a vast amount of text quickly to enhance intelligence analysis effectiveness and reduce command information loop cycles. However, when minor perturbations are added to the original intelligence, the system incorrectly classifies the text label as "Replenishment Method" instead of "Battlefield Situation." This error can result in valuable information being overlooked in the intelligence database, leading to missed fighting opportunities. Therefore, generating as many adversarial texts as possible as test cases is crucial to improving military intelligence analysis capabilities and advancing subsequent

*Corresponding author.

2643-1572/23/$31.00 ©2023 IEEE
DOI 10.1109/ASE56229.2023.00052

strategic deployments. Since it is difficult to manually write numerous test cases for the DNN under test, which we refer to as the *victim model*, in this paper, inspired by fuzz testing, we explore the potential of generating adversarial test cases [19] in a heuristic manner to deceive DNNs' decision-making. This approach facilitates efficient detection of defects and vulnerabilities in NLP software.

We summarize the challenges faced by existing work as follows:

(1) *Enhancing the ability to detect errors for DNN-based systems is the most urgent issue*. The testing process builds confidence in the system's quality by identifying and resolving defects. However, existing white-box and greedy strategy-based testing methods [21]–[23] generate adversarial test cases based on a fixed perturbation paradigm, resulting in a low success rate of 0.4% to 15.2% on the commonly used AG's News dataset [24] for toxic text detection tasks. Although heuristic testing methods [25], [26] generate more successful test cases with a success rate of up to 70.5% by iterating multiple times in an ample perturbation space, there is still room for improvement. Fig. 1 illustrates the perturbation strategies of two existing works, including synonym replacement [25] and character deletion [27], which may generate syntactic errors when the replaced word has different part-of-speech tags or meanings. Such perturbations can be easily detected by syntactic-checking tools in software systems, leading to the generated test cases incapable of revealing errors in the system. A low success rate generates numerous invalid test cases, making testing methods difficult to work on small datasets.

(2) *The existing methods take too much time to generate test cases.* Take the military software testing scenario in Fig. 1 as an example – the rapid change of the battlefield situation requires the test methods to generate test cases quickly [28]. Once the time limit of testing the victim model is exceeded, the generated test cases by the test method are useless for the improvement of the robustness of the victim model even if they can mislead the system's decision. Although current heuristic testing methods [25], [26] can generate more successful test cases, the time of generating test cases for text sequences of length up to 250 is 58.53s (IMDB [29]) and 177.81s (AG's News [24]) on average, making them impractical for time and query-constrained scenarios.

To this end, we propose LEAP, an automated black-box testing method that employs PSO [30] to search for adversarial test cases in NLP discriminative models. To increase the diversity of the population and improve the attack success rate of the test case, LEAP first generates the initial population using Levy flight and Brownian motion based on synonyms for each word, prepared using WordNet [31]. Next, as stated in the existing work [32] that the exponentially increasing perturbation space and complex search process require the search algorithm to have nonlinear search capability. Inspired by Shi et al.'s work [33], we design a new adaptive inertia weight update strategy for LEAP to optimize the search path in an exponentially growing text space, which makes the search process more efficient. If LEAP fails to find any successful adversarial test case after each round of updating particles, a greedy mutation is performed to accelerate convergence.

In this paper, we investigate the ability of LEAP to generate adversarial test cases for three victim models on three datasets, including the classical LSTM model [34] and the two popular pre-trained models, BERT [2] and DistilBERT [35], with metrics including attack success rate [36], change rate [36], and perplexity score [37]. We compared LEAP against different types of baselines, including gradient-based (i.e., A2T), greedy-based (i.e., Checklist and PRUTHI), and heuristic-based (i.e., $\text{PSO}_{attack}$ and IGA). Our results show that LEAP-generated test cases have the highest attack success rates with an average value of 79.1% against 73.0% for the next best approach ($\text{PSO}_{attack}$). Furthermore, LEAP consumes lower time overhead than other heuristic-based methods by 2.14s˜147.57s. It thus can efficiently detect defects in the system. In addition, we conducted a transferability test, adversarial training, and an ablation study to further evaluate the performance of LEAP. We also assessed the naturalness of LEAP's test cases and found that it generates less modified and more natural test cases in most cases, as evidenced by the lower perplexity scores [37].

The contributions of this paper include the following:

- We propose a new automated testing method, LEAP, which uses Levy flight [38] along with Brownian motion to reasonably extend the perturbation range and improve the quality of adversarial test cases. During the iterative search in the perturbation space, LEAP utilizes the proposed adaptive algorithm and greedy mutation for planning the search path to reduce the time overhead and query count. Our implementation and all raw data are open-source[1].
- We conducted extensive experiments comparing LEAP with state-of-the-art automated testing methods for DNN-based NLP models. LEAP generated test cases with higher attack success rates while consuming less time.
- We evaluated the effectiveness of adversarial test cases in improving the robustness of DNN-based systems. The experimental results show that adversarial training using LEAP's test cases can substantially (9.5%˜13.2%) enhance the robustness of most victim models.

## II. Background

### A. Problem Definition

As a fundamental aspect of testing techniques for DNN-based NLP systems, the test data of a test case comprises a perturbed text sequence, and the expected result is the predicted label of the original text. LEAP performs automated testing on DNNs embedded in NLP software to generate adversarial examples as perturbed sequences of the test cases. The notion of adversarial testing was introduced by Szegedy et al. [19]. In this test method, a tester adds subtle perturbations $\varepsilon$ to the original data $x$, which can be digested by a machine learning model (i.e., the victim model) $f$, but is difficult for

[1]https://github.com/lumos-xiao/LEAP

humans to perceive. This results in an adversarial example that can cause the victim model to produce erroneous results that differ from the original output $f(x)$. This paper focuses on generating test cases using black-box adversarial test methods, which only manipulate the inputs to the model.

LEAP uses the requirements of non-target adversarial testing as the objective function to find more test cases and test the DNNs more adequately. Specifically, given an original text segment $T_{ori}$ in the dataset and the corresponding adversarial test case $T_{adv}$, the optimization problem of LEAP can be defined as

$$\begin{aligned} &\arg\min_{T_{adv} \in C(T_{ori})} \|T_{ori}, T_{adv}\| \\ &\text{s.t. } F(T_{ori}) \neq F(T_{adv}) \end{aligned} \tag{1}$$

where $\|a, b\|$ denotes the difference between two pieces of text segments $a$ and $b$, such as change rate, embedding distance, etc; $F$ denotes the victim model; $C$ denotes LEAP's constraint on the quality of the adversarial test cases, here including the stop word filter [39] and the maximum change rate limit, because an excessive change rate affects the semantics and naturalness of the generated cases.

### B. Particle Swarm Optimization

PSO is a population collaborative-based search algorithm developed by Kennedy and Eberhart [30] in 1995. It simulates the foraging behavior of a flock of birds, where each individual is called a particle. It has been successfully applied in many fields, such as economic management [40], information science [41], engineering technology [42] and emotional binary classification in NLP [43]. In the original PSO, the particles simulate the solution of the optimization problem in the search space. The fitness value of a particle is evaluated according to its position, usually in terms of the objective function or optimization problem, and the particle velocity is a vector indicating the direction and distance it will move. The PSO process is described as follows:

(1) Initialization. A random population of particles is generated, and the initialization involves randomly generating each particle's position and velocity vector.

(2) Evolutionary iteration. Each particle searches the entire solution space by updating its velocity and position according to its optimal position $lBest$ so far and the optimal position $gBest$ of the population. When the particle population position is updated, the particle's optimal position and the population's optimal position are also updated.

(3) Iteration termination. When the iteration termination condition is met, the algorithm stops searching, and the last optimal position searched is the optimal solution.

In the evolutionary iteration, the updated equation for the velocity $v_d^n$ of the $n$-th particle in $d$ directions is

$$v_d^n = w v_d^n + c_1 * r_1 * (lBest_d^n - x_d^n) + c_2 * r_2 * (gBest_d^n - x_d^n) \tag{2}$$

The position update equation of the particle is

$$x_d^n = x_d^n + v_d^n \tag{3}$$

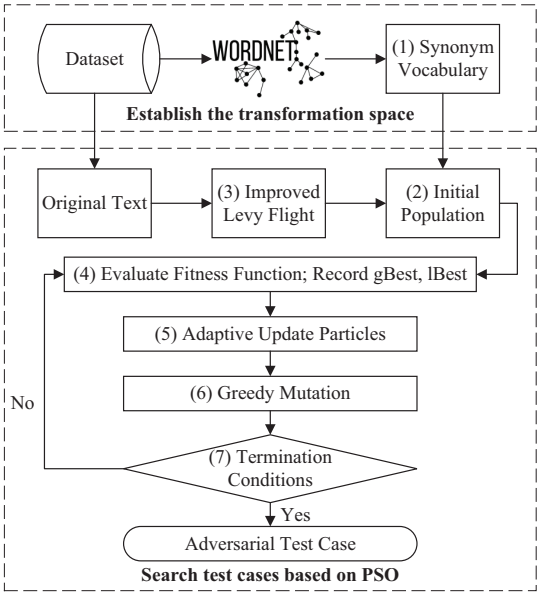


Fig. 2. Overview of LEAP.

where $x_d^n$ denotes the $d$-th dimension of the $n$-th particle in the current population; $w$ is the inertia weight; $c_1$ and $c_2$ are learning factors; $r_1$ and $r_2$ are random numbers uniformly distributed in the range of [0,1].

The setting of control parameters tremendously influences the performance of PSO [33]. The parameters $c_1$ and $r_1$ indicate the degree of influence of particles by $lBest$, i.e., how the particles assess their own information sharing and cooperation with other particles in the current population; $c_2$ and $r_2$ indicate the degree of influence of other particles by $gBest$, i.e., how the particles assess the information sharing and cooperation of other particles. The inertia weight determines the succession to the current velocity of the particle [44].

For the iteration termination, there are two general termination conditions: (1) the current iteration number $t$ reaches the preset maximum iteration number; or (2) there are individuals in the population that satisfy the accuracy requirements of the optimization problem.

## III. Design of LEAP

Fig. 2 overviews the proposed LEAP, which aims to generate adversarial test cases using actual examples from the test dataset. (1) It begins by counting all the words in the dataset and using a synonym lexicon called WordNet [31] to find synonyms for each word. (2) It then selects an original text sequence from the dataset and replaces a word with its synonym to obtain the initial position. The initial velocity (3) is obtained through a modified Levy flight. The initial position and velocity together determine the initial population of particles. Next, LEAP (4) performs an iterative search, using the confidence score of the victim model as the fitness function. It then (5) adaptively updates the velocity and position of the particles. LEAP also (6) performs greedy mutation based on the change rate and fitness score. Suppose the best individual

in the population (7) satisfies the termination conditions, which include successfully changing the prediction of the original text and reaching the maximum number of iterations. In such a case, the output is an adversarial test case. Otherwise, the iteration continues.

The depiction of LEAP is divided into two parts: 1) establishing the transformation space and 2) searching test cases based on PSO.

### A. Establishing the transformation space

To heuristically search for adversarial test cases, LEAP first defines the search space. Given that the original text $T_{ori}$=$\{w_1$, $w_2$, ..., $w_n\}$ contains $n$ words, LEAP generates potential test cases $T'_{ori}$ by replacing a word $w_i$ in $T_{ori}$ with its synonym $w'_i$, and multiple $T'_{ori}$s for each original text $T_{ori}$ constitute the search space of the test dataset together. LEAP focuses on generating semantically correct test cases and therefore uses WordNet to construct a synonym vocabulary for each word in the dataset. WordNet is a broad-coverage English lexical-semantic network where nouns, verbs, adjectives, and adverbs are respectively organized into a network of related words, with each set of synonyms representing a basic semantic concept and various relations connecting these sets.

The process of generating a synonym vocabulary in LEAP using WordNet is superior to other methods, such as using word embedding [25], language model [39], and HowNet [26], this is because:

- The word embedding method can find many candidate words by changing the embedding distance threshold to ensure diversity in the search space. However, it also introduces low-quality substitutions, such as lexical errors.
- The method using language models to build the search space produces fluent sentences because these models (especially pre-trained models [2], [3]) are obtained from large text datasets and contain contextual semantic knowledge. However, they are prone to syntactic errors because linguistic features such as syntax and semantics are ignored.
- HowNet [45] is an extensive dictionary that uses "sememe" to describe words and semantics. Different from WordNet, it only considers synonymy and positive and negative colors in semantic relations, ignoring the summary of related words, such as antonyms of words. The search space established using HowNet is too small, reducing the population diversity of PSO and thereby affecting the algorithm's ability to find higher-quality test cases.

We thus use WordNet to generate a synonym vocabulary for LEAP. The output of WordNet is a list of candidate words that are the synonym of each word $w_i$ in the original text $T_{ori}$.

### B. Searching test cases based on PSO

The respective synonym vocabulary for each word in the original text forms the search space of LEAP, which approaches automated testing as a combinatorial optimization problem and uses our improved PSO to find adversarial test cases that satisfy the objective function and constraints within the search space. We improve PSO since the original one is only suitable for continuous search spaces, but the perturbation space for the NLP test case generation task is discrete, LEAP thus updates PSO by probability according to the scalar shift discussed in Section III-B2 inspired by [26]. In addition, it improves PSO using Levy flight and adaptive methods to generate higher-quality adversarial test cases with less time overhead. Algorithm 1 outlines the search process. Next, we detail this algorithm.

**Algorithm 1** Search Process in LEAP

**Input:** $T_{ori}$: Original text, $max\_iters$: Max iteration, $pop\_size$: Number of the population in each iteration.
**Output:** $T_{adv}$: Adversarial test case.

```
T_pop←Levy-Initialization(T_ori) via Eq.7;
if T_adv in T_pop then
    return T_adv
end if
gBest=max{T_pop};
lBest=copy{T_pop};
while not exceed max_iters do
    Adaptively set inertia weight ω via Eq.8;
    for n in pop_size do
        Update the velocity and position of particle n;
    end for
    Evaluate current population;
    Greedy-Mutation based on change rate via Eq.11;
    for n in pop_size do
        if fit(n)>fit(lBest) then
            lBest=pop_n;
        end if
    end for
    if fit(lBest)>fit(gBest) then
        gBest=lBest;
    end if
    Evaluate current population;
end while
return T_adv←gBest
```

*1) Population initialization based on Levy flight:* The main task of population initialization is to determine the initial velocity and position of the particles to perform the search. To achieve this, LEAP uses the confidence of the victim model as the fitness of the particles, since the aim of an adversarial test method is to create inputs that can fool the model into making incorrect predictions with high confidence. By using confidence as the fitness function, the optimization algorithm can search for inputs that are most likely to be misclassified. LEAP generates the initial position based on the fitness score. Specifically, for each word $w_i$ in the original input text sequence $T_{ori}$, LEAP replaces only one word with its synonym at a time to minimize the modification of $T_{ori}$. This generates a series of new sequences to construct the search space corresponding to the current input. LEAP then traverses the search space to find the example with the highest fitness score, which flips the original prediction. The replaced synonym word in LEAP is designated as the best neighbor of the original word. For an original sequence $T_{ori}$, the new sequences generated from $T_{ori}$ by replacing different words with their synonym have different fitness values. LEAP thus

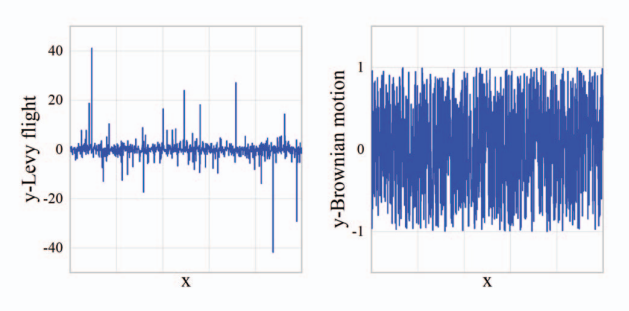


Fig. 3. Comparison of Levy flight and Brownian motion. x represents the number of steps performed and y represents the step length. Obviously, the search area covered by Levy flight (in range [-40, 40]) is much broader than Brownian motion (in range [-1, 1]).

uses these fitness values as probabilities associated with each new sequence. Based on the probabilities, it randomly selects a word in $T_{ori}$ and uses the best neighbor of this word to replace it. The replaced text is the initial position of the particle.

In [26] that uses PSO to search for adversarial test cases, the velocity of the particles is initialized using Brownian motion [46], which focuses on local search. However, the search space for the NLP test case generation task increases exponentially as the number of words in the input case increases. This search process is prone to get stuck in local optima. To address this issue, LEAP uses Levy flight to initialize the velocity of the particles (Lines 1-6). Levy flight is a random wandering mode proposed by French mathematician Paul Pierre Levy in 1930s [38], in which the steps follow the Levy distribution and can move in multidimensional space with isotropic random directions. Fig. 3 illustrates the difference between Levy flight and Brownian motion. Within 500 steps, the step length of Brownian motion mainly bounces around the current point in a small area, while Levy flight has a wandering characteristic that combines short walks and long jumps. This means that Levy flight has a higher probability of taking long steps than normal random walks. In the context of NLP, this can be useful for exploring a larger potential search space, which can improve the chances of finding effective adversarial test cases. Specifically, Levy flight allows the population to explore a wider range of input space, leading to more diverse populations. A diverse population increases the chances of finding effective adversarial test cases and helps to avoid local optima.

The step size of the Levy flight is determined by the Levy distribution, which is complex and has not been implemented yet. It is thus usually simulated using the Mantegna algorithm [47] with a step size $s$ calculated by:

$$s = \frac{\mu}{|v|^{1/\beta}} \tag{4}$$

where $\mu \sim N\left(0, \sigma_\mu^2\right)$, $v \sim N\left(0, \sigma_v^2\right)$, $\beta$ usually takes the value 1.5, and

$$\sigma_\mu = \left\{ \frac{\Gamma(1+\beta)\sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left[\frac{(1+\beta)}{2}\right]\beta 2^{\frac{(\beta-1)}{2}}} \right\}^{1/\beta} \tag{5}$$

$$\sigma_v = 1 \tag{6}$$

LEAP randomly generates the Brownian motion's step size, and each particle's initial velocity is obtained by combining Levy flight and Brownian motion. The assignment formula is:

$$v_{\text{init}} = \begin{cases} \text{levy}\left(\beta, \sigma_v\right) & , \text{levy}\left(\beta, \sigma_v\right) > \text{rand}\left(v_{\min}, v_{\max}\right) \\ \text{rand}\left(v_{\min}, v_{\max}\right), & \text{others} \end{cases} \tag{7}$$

It is observed that the step size of Brownian motion is broader than that of Levy flight, which almost occurs when both values are small. The minor oscillation feature of Brownian motion makes it have better local search capability, so the value generated by Brownian motion is used in this case. The rest of the cases use the step size generated by the Levy flight to enhance the global search capability of LEAP and thus generate better-quality adversarial test cases.

*2) Adaptive update particles:*

If there are no test cases in the initial population of LEAP that can test successfully, the population will be iterated, with the velocity of the particles being adaptively updated first, and then the particles being shifted according to the velocity (Lines 8-11). Balancing global and local search by adjusting the step size is vital for the success and efficiency of the iterative search in heuristic algorithms. PSO uses inertia weights to balance global and local search capabilities, with larger weights contributing to global search and smaller weights contributing to local search. Changing the inertia weights allows for dynamic adjustment of the search capability. The existing method [26] uses a linearly decreasing inertia weight to dynamically adjust the search process so that PSO has more global search capability at the beginning and more local search capability near the end of the run. However, the search space increases exponentially with the number of replaced words, which means that the search process of LEAP is non-linear and requires tremendous time overhead. Besides, the method of linearly decreasing inertia weights has a linear transition of search capability from global to local search, resulting in it easily falling into the saddle of high-dimensional text space later in the search. Therefore, the inertia weights should be nonlinear and change dynamically to provide a better dynamic balance between global and local search capabilities and achieve better performance.

$$\omega_n^i = \begin{cases} \omega_{\min} + \frac{\left(fit_n^i - fit_{\min}^i\right)\left(fit_{\max}^i - fit_{\min}^i\right)}{fit_{\text{mean}}^i - fit_{\min}^i}, & fit_n^i < fit_{\text{mean}}^i \\ \text{levy}\left(\beta, \sigma_v\right) \in \left(\omega_{\text{mean}}, \omega_{\max}\right), & \text{others} \end{cases} \tag{8}$$

LEAP uses a new adaptive inertia weight update method, as shown in Equation 8, where $\omega_{\min}$ and $\omega_{\max}$ are hyperparameters. Suppose the fitness score of the $n$-th particle in the $i$-th generation is less than the average value of all fitness scores. In that case, this particle can be considered far from the actual value or stuck in a local search. Then, its inertia weight is adaptively adjusted based on the fitness score. Otherwise, the inertia weight is the value generated by Levy flight, ensuring that the search process has certain randomness and explores

a larger perturbation space. After obtaining the new inertia weights, the velocity is updated according to Equation 9.

$$v_d^n = \omega^n v_d^n + v_{\max}\left(1-\omega^n\right)\left[I\left(lBest^i, x_n^i\right) + I\left(gBest, x_n^i\right)\right] \tag{9}$$

In order to search the discrete perturbation space, where

$$I(a,b) = \begin{cases} 1, a = b \\ -1, a \neq b \end{cases} \tag{10}$$

The update of the position is similarly divided into two steps. In the first step, a new move probability $P_1$ is introduced by which a particle determines whether to move to its individual best position; in the second step, each particle determines whether to move to the global best position with another move probability $P_2$. The change of each position dimension depends on $softmax(v_d^n)$. $P_1$ and $P_2$ are hyperparameters that change with iteration to improve the search efficiency by adjusting the balance between local and global search.

*3) Greedy mutation:*

In biology, genetic mutations result in differences among individuals within a population, in terms of their structure and function. To simulate this process and ensure population diversity, LEAP introduces a mutation operator to the original PSO algorithm (Line 13). To prevent excessive modification of the text, LEAP generates variation probabilities based on the change rate (*C-rate*) of the current particle from the original text, as shown in Equation 11.

$$p_{\text{mutation}} = 1 - \gamma \cdot \textit{C-rate} \tag{11}$$

Randomness is ensured by comparing the mutation probability with a random number in the range [0,1). If the generated random value is less than $p_{\text{mutation}}$, greedy mutation is performed on the particle: the words in the text sequence are replaced one by one to find the perturbed position that makes the greatest improvement in the fitness score, and then the original particle is replaced using the perturbed text. Next, LEAP updates $gBest$ and $lBest$ by fitness score, and $gBest$ is output as an adversarial test case when the iteration terminates.

## IV. Experiment Setup

We have conducted a series of experiments on three text classification datasets and three deep learning models to validate the performance of LEAP in generating test cases. We have made LEAP and all raw data publicly available. All experiments were conducted on an Ubuntu 18.04.5 LTS server with NVIDIA RTX A4000, a 12-core 2.20GHz processor Intel(R) Xeon(R) Gold 5320, and 32GB physical memory. We conducted three repetitions of experiments and averaged the experiment results for each metric. Similar to many well-acknowledged studies [25], [26], [48], [49], the victim models were tested on a set of 1,000 randomly selected examples in each experiment. Therefore, it is believed that this experimental scale is sufficient to cover different input data types and ensure the representativeness and credibility of the experiment results.

### A. Hyperparameters

LEAP is a heuristic testing method based on PSO, and the selection of hyperparameters significantly influences its performance. Among them, the population size ($pop\_size$) determines the coverage of the discrete text space, and the maximum number of iterations ($max\_iters$) affects the computational cost required for the search process. The inertia weight ($\omega$) and acceleration coefficients ($P_1$, $P_2$) jointly determine the breadth and depth of the search; overly large or small values of these hyperparameters may cause LEAP to get trapped in local optima. By parameter tuning, we set the number of individuals in the particle swarm to 60, the maximum number of iterations to 20, and the hyperparameters $\omega_{\min}$, $\omega_{\max}$, $P_1$, $P_2$ and $\gamma$ 0.2, 0.8, 0.8, 0.2 and 1, respectively.

### B. Datasets

*IMDB*[2] [29]. A dataset for emotional binary classification containing 50,000 positive and negative movie reviews was grabbed from online sources. The average length of each sequence is 215.63 words. It is divided into two parts, namely 25,000 training reviews and 25,000 test reviews. Their polarization characterizes these movie reviews.

*AG's News*[3] [24]. This dataset quotes 496,835 news articles from more than 2,000 news sources in the 4 categories of AG's News Corpus (World, Sports, Business, and Science/Technology) in the title and description fields. We concatenate the title and description fields of each news article and use the dataset organized by kaggle[4], in which each category contains 30,000 training examples and 1,900 test examples. Each example contains an average of 43 words.

*Poem Sentiment(POEM)*[5] [50]. This dataset contains 3,085,117 lines of poetry from hundreds of Project Gutenberg books, which can be used for tasks such as sentiment classification. Each line has a corresponding Gutenberg ID (1,191 unique values) from Project Gutenberg. These text segments are divided into four categories, with an average length of 8 words per segment.

### C. Victim models

To evaluate the test performance of LEAP on different DNN-based systems, we choose BERT [2] and its concise scheme Distil-BERT [35], thus verifying the performance of researchers' most common NLP models. We also report experimental results on a LSTM for text classification [34], which is widely used as a classical deep learning model with excellent performance before the advent of pre-trained models. By parameter tuning, the number of hidden layer neurons of TextBiRNN was set to 150; the dropout ratio was set to 0.1, and the maximum sequence length was set to 250. All these models have been pre-trained on BookCorpus [51], a dataset consisting of 11,038 unpublished books and English Wikipedia

[2]https://s3.amazonaws.com/fast-ai-nlp/imdb.tgz
[3]https://s3.amazonaws.com/fast-ai-nlp/ag_news_csv.tgz
[4]https://www.kaggle.com/amananandrai/ag-news-classification-dataset
[5]https://github.com/google-research-datasets/poem-sentiment

(excluding lists, tables, and titles). We also finetuned the bert-base-uncased[6], distilbert-base-uncased[7] models published by Hugging Face for each dataset.

### D. Baselines

We investigated the recent works in terms of the testing framework [36], [52], degree of automation [53], [54], and application scenario [27], [39], [48], [55]. Among these, we selected the testing framework Textattack [36], which does not require manual intervention, and conducted experiments in the context of soft-label black-box testing. To compare LEAP with different fully automated testing methods, we implemented four popular black-box testing methods and one state-of-the-art white-box testing method. Specifically, these methods are:

1) *IGA* proposed by Wang et al. [25]: the fitness function consists of confidence and alienation rate. Using single-point crossover, the text of the two parents is randomly cut to merge into a new text segment. Allowing to replace the words that have been replaced before, to a certain extent, avoids falling into the trap of local optima.

2) *PSO$_{attack}$* proposed by Zang et al. [26]: a word-level automated testing method which reforms in two steps – reducing search space and searching for adversarial test cases through designing a word substitution method based on sememes, and presenting a search algorithm based on particle swarm optimization.

3) *CheckList* proposed by Ribeiro et al. [21]: inspired by principles of behavioral testing in software engineering, CheckList guides users in what to test by providing a list of linguistic capabilities. To break down potential capability failures into specific behaviors, CheckList introduces different test types and then implements multiple abstractions to generate adversarial test cases.

4) *PRUTHI* proposed by Pruthi et al. [22]: explores adversaries which perturb sentences with four types of character-level edits: (1) Swap: swapping two adjacent internal characters of a word. (2) Drop: removing an internal character of a word. (3) Keyboard: substituting an internal character with adjacent characters of QWERTY keyboard (4) Add: inserting a new character internally in a word.

5) *A2T* proposed by Yoo et al. [23]: the component of this method is designed to generate adversarial test cases with lower computational cost, which is accelerated by making two key choices when constructing the test: (1) DistilBERT semantic textual similarity constraint, and (2) a cheaper gradient-based word importance ranking white-box method.

### E. Evaluation measures

We choose five evaluation indicators for the experiment:

1) *Success rate* (*S-rate*) [36] of generated adversarial test cases among all targeted text segments. In this experiment, its formula can be expressed as follows:

$$\text{S-rate} = \frac{N_{adv}}{N} \tag{12}$$

[6]https://huggingface.co/bert-base-uncased

[7]https://https://huggingface.co/distilbert-base-uncased

where, $N_{adv}$ is the number of adversarial test cases that test victim models successfully, and $N$ is the total number of input examples ($N$ = 1,000 in our experiment) for the current test method.

2) *Change rate* (*C-rate*) [36], which represents the average proportion of the changed words in the original text. C-rate can be expressed as:

$$\text{C-rate} = \frac{1}{N_{adv}} \sum_{k=1}^{N_{adv}} \frac{\text{diff}\, T_k}{\text{len}\,(T_k)} \tag{13}$$

where $\text{diff}\, T_k$ represents the number of words replaced in the input text $T_k$ and $\text{len}(*)$ represents the sequence length. C-rate is an indicator designed to measure the difference in content between the generated test cases and the original examples.

3) *Perplexity* (*PPL*) [37], an indicator used to assess the fluency of textual test cases. Perplexity is defined as the exponentiated average negative log-likelihood of a sequence. If we have a tokenized sequence $X$=($x_0$,$x_1$,...,$x_t$), then the perplexity of $X$ is,

$$\text{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_{i}^{t} \log p_\theta\left(x_i \mid x_{<i}\right)\right\} \tag{14}$$

where $\log p_\theta\left(x_i \mid x_{<i}\right)$ is the log-likelihood of the $i$-th token conditioned on the preceding tokens $x_{<i}$ according to the language model [56]. Intuitively, given the language model for computing PPL, the more fluent the test case, the less confusing it is.

4) *Time overhead* (*T-O*) [36], which refers to the average time it takes for a test method to generate a successful test case.

5) *Query number* (*Q-N*) [36], which indicates the average number of times a population-based method needed to query the victim model when generating a test case. The query number and the time overhead together reflect the efficiency of the testing method.

We use C-rate and PPL to quantitatively measure the naturalness and similarity between adversarial test cases and original ones, as both are easier to reproduce than human evaluation. Regarding time overhead and query number, we compare LEAP with IGA and PSO$_{attack}$, which are also heuristic testing methods, considering that non-heuristic test methods [21]–[23] generate test cases much faster due to the different search strategies. However, the experimental results in Section V show that the quality of test cases generated by such methods is much inferior to that of heuristic methods.

### F. Definition of robustness

IEEE [57] defines the robustness in software engineering as "degree to which a system, product or component performs specified functions under specified conditions for a specified period of time". Similar to [58], we define robustness as follows: denoting the input as $x$ and the relevant gold label for the main task as $y$, assuming that a model $f$ is trained on $(x, y) \sim \mathcal{D}$. Now given the adversarial test case $(x', y') \sim \mathcal{D}' \neq \mathcal{D}$, we can measure the robustness of the

model by the prediction results of $f$ on $(x', y')$. Compared to the raw prediction accuracy on $\mathcal{D}$, the less the model's prediction accuracy on $\mathcal{D}'$ drops, the fewer test cases the model misclassifies, the more robust it is.

## V. Experiment Results and Analysis

In this section, we present five research questions and discuss the experimental results.

**RQ1: How is the quality of the generated test cases by LEAP for different victim models and datasets?**

To evaluate LEAP, we compare its success rate, change rate, and perplexity with other baselines. Table I shows the comparison results on different datasets and victim models.

Compared to all the baselines, LEAP achieves higher success rates for each dataset and victim model, especially on the multi-categorical and long-series dataset, i.e., AG's News. When generating test cases for BERT, LEAP achieves a success rate of 81.2% compared to the baseline success rates of 69.6%, 70.0%, 0.4%, 9.6%, and 9.2%, which implies that LEAP can test more thoroughly against DNN-based systems with robust performance. In terms of change rate, LEAP achieves optimal results in only a few cases, with PRUTHI and CheckList often having better performance because these two methods have strict restrictions on the modification of the original text and therefore sacrifice too much performance in success rate. In the experiments tested on Distil-BERT finetuned by IMDB, the change rates of Checklist, PRUTHI, and LEAP are 61.16%, 3.4%, and 11.5%. However, the success rate of the three is 1.6%, 18.8%, and 91%, respectively, the disparity of which is significant. In addition, LEAP's PPL scores are the lowest for most cases, indicating that LEAP can generate more fluent and natural test cases. Even though LEAP's PPL is not the lowest in a few cases, it guarantees a sufficiently high success rate. For example, when testing a bidirectional LSTM trained by Poem Sentiment, LEAP outperforms PRUTHI by 52.2% in terms of success rate, while PRUTHI achieves a slightly better PPL score than LEAP.

In addition, Table I shows that the three heuristics of PSO$_{attack}$, IGA, and LEAP always have the highest success rate. Besides, Table II presents test cases generated from the same testing sequence by the three methods on a BERT finetuned by AG's News. It can be seen that the test case generated by LEAP not only deceives the victim model with high confidence but also makes minor and more natural changes to the original text. On the other hand, although LEAP and PSO$_{attack}$ also chose PSO for the iterative search, the adversarial test case generated by LEAP shows better text quality regarding the change rate and PPL score.

**Answer to RQ1:** LEAP generates higher-quality test cases for structurally different victim models and datasets with different characteristics, and it performs exceptionally well in terms of success rates.

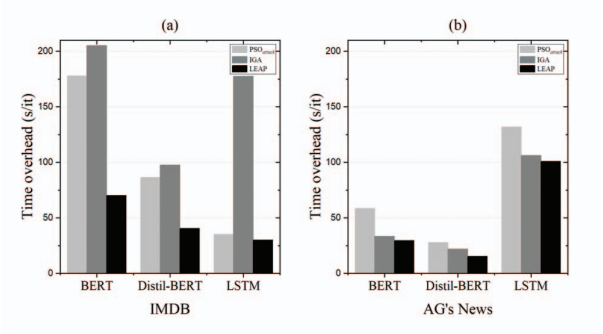


Fig. 4. Results of the time overhead for testing different victim models. The lower the values are, the more efficient the method is.

**RQ2: Can LEAP generate test cases more efficiently?**

Apart from the quality of test cases, the efficiency of the testing method, including time overhead and query number, is also our main concern. Fig. 4 shows the time overhead of generating test cases for the long text datasets IMDB and AG's News. As we can see, for all victim models, LEAP has less time overhead per successfully generated test case. On average, LEAP is 2.14s˜147.57s faster than the best baseline per generated test case. When testing BERT finetuned by AG's News, the time overheads of IGA, PSO$_{attack}$, and LEAP are 205.28s/it, 177.81s/it, and 70.17s/it, which indicates that LEAP is more efficient. The vast majority of the testing process is spent on querying the victim models [59], so the reduction in time overhead also indicates that LEAP has fewer query numbers. We show such results in the repository[8] due to limited space.

**Answer to RQ2:** In terms of testing efficiency, LEAP can generate successful test cases with less time overhead and fewer query numbers, thus saving more testing time.

**RQ3: How transferable are the test cases generated by LEAP?**

Figure 5 shows the transferability comparison of LEAP with the baselines, where we selected one baseline (i.e. IGA and PRUTHI) with excellent performance from heuristic and non-heuristic test methods, respectively. Fig. 5(a) shows the success rate results of transferring the test cases made for testing Distil-BERT to BERT and vice versa in Fig. 5(b). We find that, for the victim model finetuned on the three different types of datasets, the test cases generated by LEAP all exhibit the highest transferability, and the migrated test cases have a higher success rate [60]. Taking the IMDB dataset as an example, the success rates of test cases generated by PRUTHI, IGA, and LEAP on BERT are 13.4%, 90.8%, and 92.2%, respectively. The success rates of migration to Distil-BERT are 8.4%, 35.6%, and 65.6%, and LEAP still maintains the highest success rate.

[8]https://github.com/lumos-xiao/LEAP

TABLE I
PERFORMANCE OF SIX METHODS TO GENERATE TEST CASES. *nan* IN CHECKLIST REFERS TO THE PREDICTIONS OF ALL GENERATED TEST CASES BEING THE SAME AS THE ORIGINAL LABELS.

| Dataset | Baseline | BERT | | | Distil-BERT | | | LSTM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *S-rate* | *C-rate* | *PPL* | *S-rate* | *C-rate* | *PPL* | *S-rate* | *C-rate* | *PPL* |
| **IMDB** | **PSO**$_{attack}$ | 0.913 | 0.174 | 82.836 | 0.902 | 0.166 | 239.405 | 0.832 | 0.025 | 42.003 |
| | **IGA** | 0.908 | 0.123 | 49.584 | 0.892 | 0.121 | 58.197 | 0.799 | 0.121 | 44.165 |
| | **Checklist** | 0.020 | 0.224 | 46.291 | 0.016 | 0.611 | 46.012 | 0.188 | 0.407 | 64.049 |
| | **A2T** | 0.246 | 0.083 | 270.717 | 0.304 | 0.068 | 45.541 | 0.668 | 0.043 | 39.628 |
| | **PRUTHI** | 0.134 | **0.046** | 207.779 | 0.188 | **0.034** | 43.227 | 0.224 | **0.005** | 45.263 |
| | **LEAP** | **0.922** | 0.113 | **43.603** | **0.910** | 0.115 | **42.179** | **0.860** | 0.040 | **36.364** |
| **AG's News** | **PSO**$_{attack}$ | 0.696 | 0.244 | 690.094 | 0.644 | 0.248 | 800.665 | 0.692 | 0.197 | 343.657 |
| | **IGA** | 0.705 | 0.179 | 1013.61 | 0.621 | 0.168 | 750.969 | 0.656 | 0.165 | 305.628 |
| | **Checklist** | 0.004 | 0.032 | 1555.637 | 0.008 | **0.029** | 1677.115 | 0.036 | 0.066 | 445.628 |
| | **A2T** | 0.096 | 0.091 | 1142.844 | 0.076 | 0.090 | 925.113 | 0.152 | 0.077 | 512.617 |
| | **PRUTHI** | 0.092 | **0.029** | 1202.431 | 0.064 | 0.031 | 1182.057 | 0.104 | **0.031** | 423.197 |
| | **LEAP** | **0.812** | 0.157 | **673.893** | **0.672** | 0.162 | **744.605** | **0.896** | 0.211 | **214.954** |
| **POEM** | **PSO**$_{attack}$ | 0.658 | 0.196 | 2176.741 | 0.640 | 0.201 | **620.848** | 0.595 | 0.165 | 616.108 |
| | **IGA** | 0.576 | 0.179 | 2198.094 | 0.588 | 0.200 | 3921.968 | 0.499 | 0.143 | 523.979 |
| | **Checklist** | *nan* | *nan* | *nan* | *nan* | *nan* | *nan* | 0.048 | 0.051 | 503.804 |
| | **A2T** | 0.100 | 0.169 | **718.661** | 0.165 | 0.167 | 729.203 | 0.141 | 0.063 | 552.704 |
| | **PRUTHI** | 0.469 | 0.168 | 2472.985 | 0.416 | 0.162 | 5795.784 | 0.129 | **0.025** | **448.347** |
| | **LEAP** | **0.714** | **0.161** | 2076.027 | **0.681** | **0.157** | 991.764 | **0.651** | 0.133 | 489.931 |

TABLE II
EXAMPLES OF ADVERSARIAL TEST CASES GENERATED BY THREE METHODS USING BERT AS THE VICTIM MODEL.

| |
|---|
| (Original Text) Prediction = **Sci/Tech**. (Confidence = **0.983**) |
| TheStreet.com May Be Up for Sale – Report (Reuters) Reuters - The Street.com Inc. , the financial news and commentary Web site, may be up for sale, according to a report in Business Week, sparking a 7 percent rise in its shares. |
| (IGA) Prediction = **Business**. (Confidence = **0.920** ) |
| TheStreet.kom May Be Up for Sale – Report (Reuters) Reuters - The Street.com Inc. , the financial novice and commentary Network site, may be up for sale, according to a report in Business Week, sparking a 7 percent rise in its shares. |
| (PSO$_{attack}$) Prediction = **Business**. (Confidence = **0.983** ) |
| TheStreet.com May Be Up for Sale – Exposition (Reuters) Reuters - TheStreet.com Inc. , the fiscal news and critique Web locale, may be up for monopoly, according to a report in Business Week, sparking a 7 percent rise in its stocks. |
| (LEAP) Prediction = **Business**. (Confidence = **0.998** ) |
| TheStreet.com May Be Up for Sale – Report (Reuters) Reuters - The Street.com Inc. ,the financial news and commentary vane site, may be up for sale, according to a report in job Week, sparking a 7 percent rise in its shares. |

Note: As the text in AG's News is of moderate length, we use it to showcase the adversarial test cases. The modified words in the adversarial test cases are highlighted in red.

**Answer to RQ3:** Test cases generated by LEAP have higher transferability, which means that LEAP is able to uncover more defects in DNN-based systems even without access to their internal DNN models.

**RQ4: Whether the test cases generated by LEAP contribute to enhancing the robustness of the victim model?**

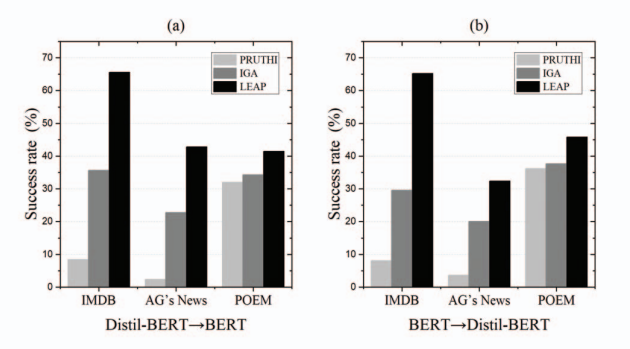


Fig. 5. The success rates of transferred adversarial test cases on the three datasets (want ↑)

For this research question, to simulate the low-resource scenario, we mixed the adversarial test cases generated from 10% of the original training set with the original training set according to the experimental setting of [61]. We used the IMDB with the most extended text length (i.e., 215 words/it) in our experimental datasets as the original dataset, resulting in three adversarial training sets as shown in Table III. The success rates of all the methods on different adversarial training datasets decreased, and the success rates on the victim models finetuned with IMDB$_{LEAP}$ are 3.9%, 77.59%, and 80.4%, respectively, with the most significant decreases. This implies that the test cases generated by LEAP improve the model's robustness more than the other baselines, since a lower success rate demonstrates that the victim model correctly classifies more adversarial test cases. Notably, LEAP still manages to obtain the highest success rate regardless of which adversarial training set finetuned victim model is tested, which further illustrates the excellent performance of LEAP in mining the defects of DNN-based systems.

We use the change rate to measure the quality of test cases. The adversarially trained victim models, especially

TABLE III
PERFORMANCE COMPARISON OF TEST METHODS ON BERT AFTER ADVERSARIAL TRAINING.

| Baseline | Indicator | IMDB | IMDB$_{PRU}$ | IMDB$_{IGA}$ | IMDB$_{LEAP}$ |
|---|---|---|---|---|---|
| **PRU THI** | *S-rate* | 0.134 | 0.081 | 0.076 | 0.039 |
| | *C-rate* | **0.046** | **0.036** | **0.037** | **0.053** |
| | *T-O(s/it)* | **5.164** | **4.052** | **5.858** | **4.599** |
| **IGA** | *S-rate* | 0.908 | 0.904 | 0.848 | 0.776 |
| | *C-rate* | 0.123 | 0.137 | 0.162 | 0.166 |
| | *T-O(s/it)* | 33.195 | 26.387 | 58.124 | 69.357 |
| **LEAP** | *S-rate* | **0.922** | **0.918** | **0.909** | **0.804** |
| | *C-rate* | 0.112 | 0.132 | 0.146 | 0.161 |
| | *T-O(s/it)* | 29.383 | 25.632 | 41.766 | 54.431 |

those finetuned using IMDB$_{LEAP}$, force the test method to increase the original text's perturbation to generate successful test cases. As shown in Table III, when testing the victim model finetuned by IMDB$_{PRU}$ and IMDB$_{IGA}$, the change rate of PRUTHI becomes lower instead. We believe this is because PRUTHI increases the perturbation on the original text to generate mostly failed test cases, which leads to an excessive decrease in the success rate compared to the one on the original training set. In addition, we observed that models finetuned by the adversarial training sets significantly increased the time overhead of the test methods, with the models finetuned using IMDB$_{LEAP}$ increasing the most. This also indicates that testing tools have the most difficulty to find successful adversarial test cases for the model finetuned with LEAP-generated test cases, which on the other hand indicates LEAP can improve the robustness of victim models.

**Answer to RQ4:** The training set with test cases generated by LEAP significantly reduces the success rate, case quality, and efficiency of the test methods. Therefore, the adversarial test cases generated by LEAP are efficacious for improving the robustness of the victim model.

**RQ5: Does each of the method components proposed in this paper improve the quality of the generated test cases and the testing efficiency of LEAP?**

We finetuned BERT as the victim model on three datasets, ablating each component of LEAP that is different from the most similar existing work PSO$_{attack}$ to investigate its effectiveness. Table IV shows the experimental results using 1000 test examples. Since the test set of POEM only contains 104 examples, we sampled the test set 10 times using different random seeds with 100 examples each time. On the IMDB dataset, LEAP only improves the success rate by 0.9% compared to PSO$_{attack}$. However, LEAP is nearly twice as fast as PSO$_{attack}$ in terms of time overhead. On AG's News, LEAP shows significant improvement in all the metrics. In particular, the use of Levy flight for population initialization and adaptive update operator increases the success rate by 11.6%, reduces the change rate by 8.66%, and decreases the time overhead by 107.64s. On POEM, the use of the greedy variation operator reduces the success rate of LEAP by 0.3%, which is because the introduction of the greedy strategy increases the risk of the search algorithm falling into local optima in high-dimensional text data. However, it effectively reduces the change rate by 0.8% and the time overhead by 0.41s. Overall, despite the datasets being from different domains with different textual features, LEAP's improved strategy achieves better test results than PSO$_{attack}$.

TABLE IV
RESULTS OF ABLATION STUDY ON LEVY FLIGHT-BASED POPULATION INITIALIZATION, ADAPTIVE UPDATE PARTICLES (*adaptive*), AND GREEDY MUTATION (*greedy*).

| Dataset | Testing method | S-rate | C-rate | T-O(s/it) |
|---|---|---|---|---|
| **IMDB** | PSO$_{attack}$ | 0.913 | 0.173 | 58.533 |
| | *w/o adaptive,greedy* | 0.916 | 0.135 | 44.026 |
| | *w/o greedy* | 0.916 | 0.118 | 34.785 |
| | LEAP | **0.922** | **0.113** | **29.380** |
| **AG's News** | PSO$_{attack}$ | 0.696 | 0.244 | 177.811 |
| | *w/o adaptive,greedy* | 0.712 | 0.178 | 123.042 |
| | *w/o greedy* | 0.778 | 0.158 | 99.688 |
| | LEAP | **0.812** | **0.157** | **70.174** |
| **POEM** | PSO$_{attack}$ | 0.658 | 0.196 | 1.457 |
| | *w/o adaptive,greedy* | 0.690 | 0.189 | 1.930 |
| | *w/o greedy* | 0.711 | 0.169 | 1.835 |
| | LEAP | **0.714** | **0.161** | **1.426** |

**Answer to RQ5:** Compared to the most similar existing work, LEAP's components are effective in generating high-quality test cases more efficiently.

## VI. THREATS TO VALIDITY

Our experimental results demonstrate LEAP's effectiveness. However, we also acknowledge some threats to validity.

**Internal validity**. The main threat comes from the setting of hyperparameters in the experiments, such as population size and the maximum number of iterations. To mitigate the threat, we use the same hyperparameters for all experiments on each dataset, and try to choose the same hyperparameters as the existing method PSO$_{attack}$ to show the validity of our method.

**External validity**. Our experiments focused on testing DNNs in an English environment, which may threaten the generality of LEAP for other languages. But applying LEAP on DNNs in other languages requires only minor input adjustments. We mitigate this threat by evaluating our approach on three kinds of datasets and three types of NLP models. This makes us confident that LEAP will work across a variety of NLP applications.

## VII. RELATED WORK

**Testing AI Software.** The development of Artificial Intelligence (AI) software has been gaining momentum in recent years, with a growing need for effective testing strategies to ensure their reliability and performance. Automated testing techniques have been widely used by software professionals due to their efficiency, cost-effectiveness and reusability. In the field of Computer Vision (CV), a large number of automated testing techniques have been proposed [62]–[64]. The primary difference between NLP and CV software is that the feature space of text data is discrete, and any modifications to the original example are more likely to result in errors in semantics

and sentence fluency, which can be easily detected [20]. Morris et al. [36] decomposed the testing process into four components: goal function, constraint list, transformation, and search method, and unified them within the Python framework TextAttack. Tan et al. [52] demonstrated the incorporation of adversarial attacks as reliability tests into the reliability testing framework DOCTOR, presenting a method to enhance accountability in existing efforts. Overall, there are three main types of DNN-based automated testing methods for AI software: 1) white-box testing methods [23], [65] based on internal information such as DNN gradients, 2) greedy methods [21], [22] that modify the text at each specific index to minimize the original DNN prediction, and 3) heuristic methods [25], [26] that heuristically search for the optimal option among potential test cases.

**Testing NLP Software.** In the field of NLP, Ribeiro et al. [53] utilized large-scale language models and human feedback to generate adaptive unit tests for victim models. Wu et al. [54] developed Errudite, an interactive tool that utilizes domain-specific language to facilitate precise error grouping and analysis. In contrast, our paper focuses on automating the testing of NLP software, taking into account time and cost constraints. Based on the minimum perturbation units used in applications, related works are divided into three aspects:

**1) Sentence-level method.** Sentence-level testing methods are more flexible in terms of perturbation, and the modified sentence can be inserted in any part of the text when the semantics and syntax are correct. It is executed by adding ordered words of a certain length. Sentence-level methods are widely used in Q&A [66], [67] and machine understanding [68], [69] systems, but have yet received more research in text classification [70]. Since the sentence-level method modifies the entire sentence with a substantial impact on the semantics of the paragraph [71], the naturalness of the generated test cases is particularly affected. Even if the test is successful, it is often incomprehensible to humans. In contrast, our method only modifies individual words of the original text with controlled modification restrictions, thus ensuring better naturalness.

**2) Char-level method.** Char-level methods aim to modify a few characters within a word to generate test cases that cause DNNs to make decisions incorrectly [72], [73]. Given that the modifications typically involve spelling errors, Li et al. [27] generated adversarial test cases by inserting, swapping, and deleting specific characters, combined with the Jacobian matrix of the victim model. Since character-level methods are prone to produce misspelled words [74], today's splendid spell-checking tools can easily detect such perturbations. In contrast, LEAP plans the perturbation space utilizing a lexical network to produce a synonym dictionary, and the potential perturbations are all actual words, so there is no problem with misspellings.

**3) Word-level method.** Word-level methods perturb text by inserting, deleting, and replacing whole words, which is significantly better than other methods in naturalness and transferability, and therefore has gained the most attention [75], [76]. Li et al. [39] utilized pre-trained language models as masked models to generate substitute words, considering contextual information. Jin et al. [48] employed word importance ranking and cosine similarity between word vectors for synonym replacement. Ye et al. [55] formulated a hard-label scenery as an optimization problem based on gradient perturbation metrics in word embedding space, generating test cases with smaller query budgets and higher semantic similarity. LEAP is a word-level testing method that uses PSO to determine the words to be replaced and redesigns the internal arithmetic of PSO by combining the features of NLP test cases. This allows our method to guarantee the same high success rate as other heuristic testing methods while requiring less time overhead and fewer queries.

## VIII. Conclusion

In this paper, we propose LEAP, a black-box testing method for DNN-based NLP systems that efficiently generates adversarial test cases. To address the problems of low data utilization and high time overhead in current testing methods, we design new components for discrete text data, including initializing populations using Levy flight, adaptively updating particles, and employing a greedy mutation approach. We evaluate the performance of LEAP using three datasets, three advanced deep learning models, and five baselines. The experimental results demonstrate that the average success rate of adversarial test cases generated by LEAP is 79.1%, surpassing other baselines, and that the time overhead is reduced by 2.14s˜147.57s compared to other heuristic-based methods. We also investigate the value of adversarial test cases generated by LEAP in enhancing the robustness of victim models.

For future work, we plan to enhance the scalability of LEAP to encompass a broader range of NLP downstream tasks and accommodate more complex perturbation scenarios, including character-level or sentence-level modifications. To achieve this, we will explore modular granularity settings and adaptive search algorithms as potential solutions.

## Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grants 62272145 and U21B2016.

## References

[1] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, (New Orleans, Louisiana), pp. 2227–2237, Association for Computational Linguistics, June 2018.

[2] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer.," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[5] H. Liu and Z. Long, "An improved deep learning model for predicting stock market price time series," *Digital Signal Processing*, vol. 102, p. 102741, 2020.

[6] T. H. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–38, 2020.

[7] S. Cho, W. Shin, N. Kim, J. Jeong, and H. P. In, "Priority determination to apply artificial intelligence technology in military intelligence areas," *Electronics*, vol. 9, no. 12, p. 2187, 2020.

[8] S. Lee, S. Cha, D. Lee, and H. Oh, "Effective white-box testing of deep neural networks with adaptive neuron-selection strategy," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 165–176, 2020.

[9] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 477–486, IEEE, 2007.

[10] A. Kolchin and S. Potiyenko, "Extending data flow coverage to test constraint refinements," in *Integrated Formal Methods: 17th International Conference, IFM 2022, Lugano, Switzerland, June 7–10, 2022, Proceedings*, pp. 313–321, Springer, 2022.

[11] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 739–743, 2018.

[12] C. Lemieux and K. Sen, "Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 475–485, 2018.

[13] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 146–157, 2019.

[14] Z. Yu, F. Fahid, T. Menzies, G. Rothermel, K. Patrick, and S. Cherian, "Terminator: Better automated ui test case prioritization," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 883–894, 2019.

[15] N. Yousaf, F. Azam, W. H. Butt, M. W. Anwar, and M. Rashid, "Automated model-based test case generation for web user interfaces (wui) from interaction flow modeling language (ifml) models," *IEEE Access*, vol. 7, pp. 67331–67354, 2019.

[16] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Humanoid: A deep learning-based approach to automated black-box android app testing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1070–1073, IEEE, 2019.

[17] M. A. Cusumano and S. A. Smith, "Beyond the waterfall: Software development at microsoft," 1995.

[18] A. M. Dima and M. A. Maassen, "From waterfall to agile software: Development models in the it sector, 2006 to 2018. impacts on company management," *Journal of International Studies*, vol. 11, no. 2, pp. 315–326, 2018.

[19] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[20] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," in *IJCAI*, 2018.

[21] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of nlp models with checklist (extended abstract)," in *Thirtieth International Joint Conference on Artificial Intelligence IJCAI-21*, 2021.

[22] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5582–5591, 2019.

[23] J. Y. Yoo and Y. Qi, "Towards improving adversarial training of nlp models," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 945–956, 2021.

[24] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, pp. 649–657, 2015.

[25] X. Wang, H. Jin, and K. He, "Natural language adversarial attacks and defenses in word level," *arXiv preprint arXiv:1909.06723*, 2019.

[26] Y. Zang, F. Qi, C. Yang, Z. Liu, M. Zhang, Q. Liu, and M. Sun, "Word-level textual adversarial attacking as combinatorial optimization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (Online), pp. 6066–6080, Association for Computational Linguistics, July 2020.

[27] J. Li, S. Ji, T. Du, B. Li, and T. Wang, "Textbugger: Generating adversarial text against real-world applications," *arXiv preprint arXiv:1812.05271*, 2018.

[28] C. Hagen and J. Sorenson, "Delivering military software affordably," *Defense AT&L*, vol. 42, no. 2, pp. 30–34, 2013.

[29] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

[30] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[31] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, p. 39–41, nov 1995.

[32] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1362–1381, 2009.

[33] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, vol. 1, pp. 101–106, IEEE, 2001.

[34] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 2873–2879, 2016.

[35] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2019.

[36] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 119–126, 2020.

[37] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, "Perplexity—a measure of the difficulty of speech recognition tasks," *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, 1977.

[38] P. Lévy, "L'addition des variables aléatoires définies sur une circonférence," *Bulletin de la Société mathématique de France*, vol. 67, pp. 1–41, 1939.

[39] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "Bert-attack: Adversarial attack against bert using bert," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6193–6202, 2020.

[40] S. Phommixay, M. L. Doumbia, and D. Lupien St-Pierre, "Review on the cost optimization of microgrids via particle swarm optimization," *International Journal of Energy and Environmental Engineering*, vol. 11, no. 1, pp. 73–89, 2020.

[41] T. Latchoumi, T. Ezhilarasi, and K. Balamurugan, "Bio-inspired weighed quantum particle swarm optimization and smooth support vector machine ensembles for identification of abnormalities in medical data," *SN Applied Sciences*, vol. 1, no. 10, pp. 1–10, 2019.

[42] B. Su, Y. Lin, J. Wang, X. Quan, Z. Chang, and C. Rui, "Sewage treatment system for improving energy efficiency based on particle swarm optimization algorithm," *Energy Reports*, vol. 8, pp. 8701–8708, 2022.

[43] G. Tambouratzis, "Pso optimal parameters and fitness functions in an nlp task," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 611–618, IEEE, 2019.

[44] D. Li, W. Guo, A. Lerch, Y. Li, L. Wang, and Q. Wu, "An adaptive particle swarm optimizer with decoupled exploration and exploitation for large scale optimization," *Swarm and Evolutionary Computation*, vol. 60, p. 100789, 2021.

[45] Z. Dong and Q. Dong, "Hownet - a hybrid language and knowledge resource," in *International Conference on Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003*, pp. 820–824, 2003.

[46] A. Einstein, *Investigations on the Theory of the Brownian Movement*. Courier Corporation, 1956.

[47] R. N. Mantegna, "Fast, accurate algorithm for numerical simulation of levy stable stochastic processes," *Physical Review E*, vol. 49, no. 5, p. 4677, 1994.

[48] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, "Is bert really robust? a strong baseline for natural language attack on text classification and entailment," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 8018–8025, 2020.

[49] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, "Generating natural language adversarial examples," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2890–2896, 2018.

[50] E. Sheng and D. C. Uthus, "Investigating societal biases in a poetry composition system," in *Proceedings of the Second Workshop on Gender Bias in Natural Language Processing*, pp. 93–106, 2020.

[51] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

[52] S. Tan, S. Joty, K. Baxter, A. Taeihagh, G. A. Bennett, and M.-Y. Kan, "Reliability testing for natural language processing systems," *arXiv preprint arXiv:2105.02590*, 2021.

[53] M. T. Ribeiro and S. Lundberg, "Adaptive testing and debugging of nlp models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3253–3267, 2022.

[54] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld, "Errudite: Scalable, reproducible, and testable error analysis," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 747–763, 2019.

[55] M. Ye, C. Miao, T. Wang, and F. Ma, "Texthoaxer: budgeted hard-label adversarial attacks on text," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 3877–3884, 2022.

[56] C. I. Meister and R. Cotterell, "Language model evaluation beyond perplexity," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, vol. 1, pp. 5328–5339, Association for Computational Linguistics, 2021.

[57] I. ISO and N. IEC, "Iso/iec," *IEEE International Standard-Systems and software engineering–Vocabulary*, pp. 1–541, 2017.

[58] X. Wang, H. Wang, and D. Yang, "Measure and improve robustness in nlp models: A survey," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4569–4586, 2022.

[59] Z. Yang, J. Shi, J. He, and D. Lo, "Natural attack for pre-trained models of code," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, (Los Alamitos, CA, USA), pp. 1482–1493, IEEE Computer Society, may 2022.

[60] T. Wang, X. Wang, Y. Qin, B. Packer, K. Li, J. Chen, A. Beutel, and E. Chi, "Cat-gen: Improving robustness in nlp models via controlled adversarial text generation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5141–5146, 2020.

[61] D. Li, Y. Zhang, H. Peng, L. Chen, C. Brockett, M.-T. Sun, and W. B. Dolan, "Contextualized perturbation for textual adversarial attack," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5053–5069, 2021.

[62] Z. Yuan, J. Zhang, Y. Jia, C. Tan, T. Xue, and S. Shan, "Meta gradient adversarial attack," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7728–7737, IEEE Computer Society, 2021.

[63] J. Rony, E. Granger, M. Pedersoli, and I. B. Ayed, "Augmented lagrangian adversarial attacks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7718–7727, IEEE, 2021.

[64] P. Zhang, B. Ren, H. Dong, and Q. Dai, "Cagfuzz: coverage-guided adversarial generative fuzzing testing for image-based deep learning systems," *IEEE Transactions on Software Engineering*, 2021.

[65] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 31–36, 2018.

[66] W. C. Gan and H. T. Ng, "Improving the robustness of question answering systems to question paraphrasing," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 6065–6075, 2019.

[67] E. Wallace, P. Rodriguez, S. Feng, I. Yamada, and J. Boyd-Graber, "Trick me if you can: Human-in-the-loop generation of adversarial examples for question answering," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 387–401, 2019.

[68] J. Lin, J. Zou, and N. Ding, "Using adversarial attacks to reveal the statistical bias in machine reading comprehension models," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 333–342, 2021.

[69] Y. Wang and M. Bansal, "Robust machine comprehension models via adversarial training," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 575–581, 2018.

[70] Y. Xu, X. Zhong, A. J. Yepes, and J. H. Lau, "Grey-box adversarial attack and defence for sentiment classification," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4078–4087, 2021.

[71] J.-t. Huang, J. Zhang, W. Wang, P. He, Y. Su, and M. R. Lyu, "Aeon: A method for automatic evaluation of nlp test cases," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2022, (New York, NY, USA), p. 202–214, Association for Computing Machinery, 2022.

[72] S. Eger, G. G. Şahin, A. Rücklé, J.-U. Lee, C. Schulz, M. Mesgar, K. Swarnkar, E. Simpson, and I. Gurevych, "Text processing like humans do: Visually attacking and shielding nlp systems," in *Proceedings of NAACL-HLT*, pp. 1634–1647, 2019.

[73] J. Ebrahimi, D. Lowd, and D. Dou, "On adversarial examples for character-level neural machine translation," in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 653–663, 2018.

[74] X. Yang, W. Liu, D. Tao, and W. Liu, "Besa: Bert-based simulated annealing for adversarial text attacks.," in *IJCAI*, pp. 3293–3299, 2021.

[75] D. Lee, S. Moon, J. Lee, and H. O. Song, "Query-efficient and scalable black-box adversarial attacks on discrete sequential data via bayesian optimization," in *International Conference on Machine Learning*, pp. 12478–12497, PMLR, 2022.

[76] L. Yuan, X. Zheng, Y. Zhou, C.-J. Hsieh, and K.-W. Chang, "On the transferability of adversarial attacks against neural text classifier," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 1612–1625, 2021.