CoT-DPG: A Co-Training based Dynamic Password Guessing Method

Chenyang Wang, Fan Shi, Min Zhang✉, Chengxi Xu, Miao Hu, Pengfei Xue, Shasha Guo, Jinghua Zheng
National University of Defense Technology
Emails: {wcy, shifan17, zhangmindy, xuchengxi, humiao17, xuepengfei, guoshasha13, zhengjinghua}@nudt.edu.cn

***Abstract*—Password is still the primary authentication method, and the security community researches password guessing to improve password security. Dynamic password guessing continuously collects target's information and dynamically fits the distribution during the guessing process, thus expanding the threat. Existing methods are mainly of two types: dynamic adjustment of password policies and dynamic generation based on generative models. However, these methods fit the target distribution from a single perspective, ignoring the complementary effects of information between different dimensions. Dynamic password guessing performance will be greatly improved if information from multiple dimensions is well utilized, but how to effectively fuse multidimensional information is a challenge.**

Motivated by this, we propose CoT-DPG, a new dynamic password guessing framework that allows multiple guessing models to learn collaboratively and complement each other's knowledge. This is the first application of the co-training approach in multi-view learning to password guessing. Firstly, at the feature level, we dynamically update the neural network parameters and fit the target distribution based on incremental training. Secondly, at the character level, we design a policy distribution optimization approach to alleviate the blindness of policy selection. Thirdly, we use the co-training approach for complementary learning, iterative training, and password generation in multiple dimensions. Finally, the experiments demonstrate the effectiveness of the proposed framework, with the absolute improvement in cracking rate of 6.4% to 26.7% over the state-of-the-art method on eight real-world password datasets.

I. Introduction

Textual password is the primary authentication approach in almost all web services. Although various authentication solutions such as two-factor authentication [1], [2] and biometric authentication [3], [4] are applied, textual passwords will continue to be the main method used for the foreseeable future. Because textual passwords have the advantage of being low-cost and easy to deploy [5], [6], [7], [8], [9]. A large number of password guessing methods have been proposed in order to improve password security by evaluating the security strength of password dataset under guessing attacks. These methods construct models to learn the distribution of historically leaked password data and generate a large number of guesses to crack the target dataset. The three predominant ways of password guessing are rule-based (such as Hashcat [10] and John the Ripper [11]), probability-based (such as PCFG [12] and Markov [13]), deep learning-based (such as FLA [14], PassGAN [15]). Most of the existing work is geared towards the scenario where some of the passwords have been leaked from the target site, and these methods construct models to break other passwords of the site. Yet few studies have focused on how to guess passwords for completely unknown sites. At this point, because the distribution of passwords on the target site is completely unknown, the difficulty of cracking is greatly increased.

Dynamic password guessing (DPG) is the password guessing approach that dynamically adapts the guessing strategy based on the feedback received from the interaction with the attacked password set [16]. Existing dynamic password guessing methods can be classified into two types: Adaptive Rule based Password Guessing (ARPG) based [17], [18] and generative model based [16]. These two types of methods dynamically fit the target's password distribution at the character level or feature level. Adaptive rule based password guessing aims at dynamically adjusting the transformation policies (mangling rules) based on feedback information. This type of method essentially exploits the **character-level password locality**, as string-similar passwords are clustered in character space. This type of method suffers from the problem that rule definition requires domain knowledge and policy configuration is difficult. The generative model based method exploits **feature-level password locality** to continuously generate guesses around hit passwords. This is because semantically similar passwords cluster together in the feature space of deep neural networks.

However, existing methods only implement DPG at a single level, which is an insufficient use of information. This will lead to the difficulty of generating more surrounding passwords during the dynamic guessing process. We compare dynamic password generation using only a single method with multiple methods, as shown in Fig. 1. We count all passwords around the password 'pakistan' (with an edit distance of 1 from its string) in the 000webhost dataset. That is, these passwords are all real-world passwords and are similar. We then observe the percentage of these passwords generated during each iteration. When generating using only the FLA method, it fails to gen-

✉Min Zhang is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230755
www.ndss-symposium.org

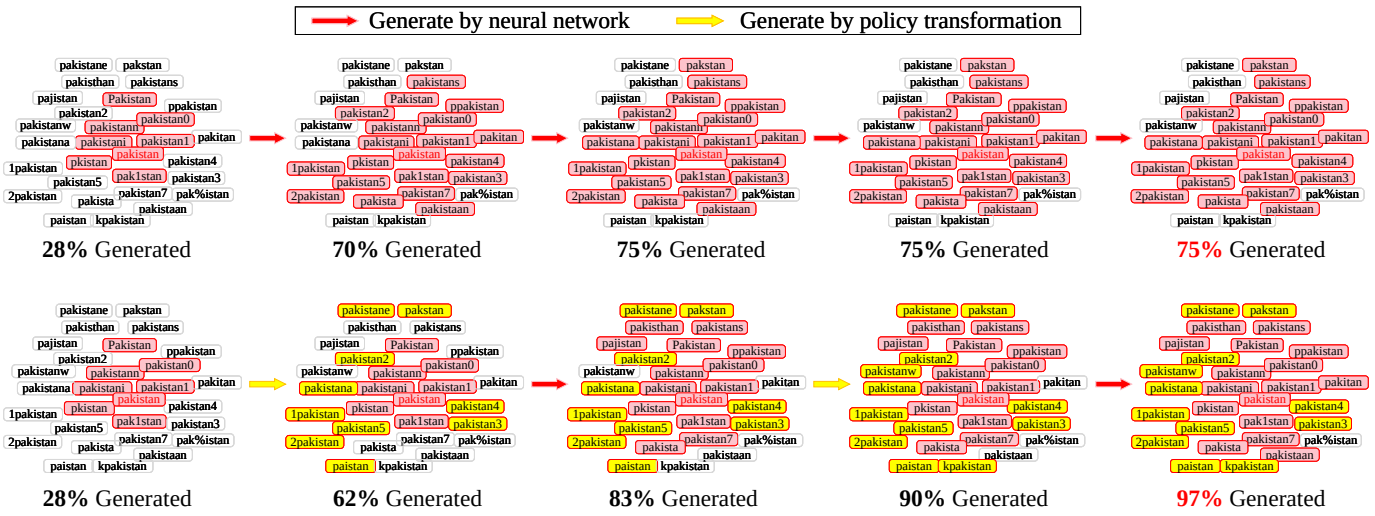


Fig. 1. An example of dynamically generated passwords around the password 'pakistan' using a single model and multiple models, respectively. Passwords in the figure are real-world passwords in the 000webhost dataset. Red arrows indicate generation using a neural network model (FLA) and yellow arrows indicate generation using policy transformation. During the iteration, we use the last hit passwords as the training set. It can be seen that using multiple models iteratively is able to hit more passwords than a single model.

erate more passwords around 'pakistan' after the third round by 75%. While more hit passwords are generated overall, it is difficult to generate more on a local scale. However, more passwords can be generated when using the FLA model and policy transformation method iteratively, which is 97%. This illustrates that feature-level password locality and character-level password locality can complement each other. Thus we can generate more password guesses and improve the dynamic password guessing performance by using the dual password locality for iterative learning.

Motivated by this, we propose CoT-DPG, a co-training based dynamic password guessing framework, which allows multiple guessing models to learn from each other for efficient password guessing. Unlike existing methods that dynamically generate from only a single information dimension, CoT-DPG learns complementarily at both feature and character levels to guide the DPG process. CoT-DPG improves the generalization while ensuring accuracy, thus generating more valid passwords and increasing the cracking rate. Firstly, at the feature level, we design a feedback-based password learning approach based on incremental training, which implicitly represents the feature-level password locality and generates password guesses through neural networks. The incremental training approach is easy to implement and applicable to multiple methods. Secondly, at the character level, we design the local password transformation policy generation algorithm that explicitly represent character-level password locality. We also design a policy distribution optimization method based on the Particle Swarm Optimization (PSO) algorithm to optimize the password policy distribution and improve the quality of password generation. Thirdly, based on the co-training approach, model incremental training and policy distribution optimization are performed alternately. CoT-DPG makes full use of the multidimensional information to guide DPG, which enables faster and better fitting of the target distribution. Finally, we demonstrate the effectiveness of the proposed method through extensive experiments, and the cracking rate of CoT-DPG exceeds the state-of-the-art methods by 6.4% to 26.7% absolutely. The key contributions are as follows.

- We propose a new dynamic password guessing framework CoT-DPG, which enables multiple guessing methods to complement each other's knowledge and improve DPG performance. This is the first application of co-training in multi-view learning to password guessing.
- At the feature level, we design a feedback-based password incremental training approach to implicitly represent feature-level password locality. At the character level, we design a policy generation algorithm that explicitly represents character-level password locality. We design a PSO-based policy distribution optimization algorithm to alleviate the blindness issue of policy selection.
- Based on the co-training approach, model incremental training and policy distribution optimization are carried out in alternating iterations, thus complementing each other's knowledge and generating password guesses more comprehensively.
- Through password guessing experiments on eight real-world password datasets, we demonstrate the effectiveness of CoT-DPG, with a higher cracking rate than the state-of-the-art dynamic password guessing methods.

The remainder of this paper is organized as follows. In Section II, we introduce related work on general and dynamic password guessing as well as background on co-training. Section III describes the dynamic password guessing framework CoT-DPG. In Section IV we evaluate the proposed method through password guessing experiments. In Section V, we discuss some insights and future work. Finally, Section VI concludes the paper.

II. Related Work and Background

A. General Password Guessing

Research on password guessing dates back to 1979 [19], [20], [21]. Initially, brute force and dictionary attack methods were employed. Later researchers proposed rule-based guessing methods and data-driven guessing models to effectively crack generic passwords within larger guesses in offline trawling attack scenario [22], [23], [24].

Rule-based Methods. Rule-based methods are also known as dictionary attacks and policy-based attacks. The most widely known are two open-source password cracking software, Hashcat [10] and John the Ripper (JtR) [11]. Liu et al. [25] introduce techniques to reason analytically and efficiently about transformation-based password cracking in software tools. Di et al. [26] propose automated training techniques to improve the cracking rate of these rule-based cracking tools. Eckroth et al. [27] develop an algorithm that automatically finds successful rules via the combinatorial generation of rules and empirical observation of how often each generated rule transforms a dictionary word to a target password.

Data-Driven Guessing Models. Data-driven guessing models are trained on historically leaked password datasets to learn the data distribution and generate password guesses. Traditional machine learning models applied to password guessing are Markov model [13], Probabilistic Context Free Grammar (PCFG) [12], and random forest [28]. Improved password guessing models based on the basic Markov model [29], [30], [31] and improved password guessing models based on the PCFG model [32], [33], [34], [35] have been proposed continuously. With the rapid development of deep learning, a large number of deep learning-based password guessing models have emerged. Melicher et al. [14] first propose a Recurrent Neural Network (RNN)-based [36] password guessing model, named Fast, Lean, Accurate (FLA). Hitaj et al. [15] propose PassGAN, a GAN-based [37] password guessing model to autonomously learn the distribution of password data. Password guessing models based on recurrent neural networks [38], [39], [40], [41], [42], [43], [44], [45], [46] and generative adversarial networks [47], [48], [49], [50], [51], [52] have proliferated in recent years. Xiu et al. [53] propose a new targeted guessing mode, PointerGuess, to accurately and comprehensively characterize users' password reuse behaviors. With the application of pre-trained language models, Xu et al. [18] propose a bi-directional-transformer (BERT)-based [54] guessing framework, named PassBERT, which first applies the pre-training/finetuning paradigm to password guessing attacks. Rando et al. [55] present PassGPT, a Generative Pre-trained Tansformer (GPT) [56] based password generation model. Su et al. [57] present PagPassGPT, a GPT based model which performs pattern guided guessing by incorporating pattern structure information as background knowledge. In addition, Xie et al. [58] propose Guessfuse, a hybrid password guessing framework, which combines multiple password guessing methods.

B. Dynamic Password Guessing

Dynamic password guessing is a method that dynamically adjusts the model based on feedback for password guessing. DPG is described as follows: (1) We don't have any information about the target password set. (2) Once the first password has been cracked, we can begin to observe and model the distribution of the target set. (3) Each new hit guess provides valuable information that we can use to improve the quality of guessing. (4) We continue to crack more passwords of the target set through an iterative process.

Adaptive Rule based Password Guessing. In 2021, Pasquini et al [17] devise the Adaptive Mangling Rules attack relying on deep learning techniques. They build neural networks to model the functional relationship between mangling rules (transformation policies) and dictionary words. They then introduce a dynamic guessing strategy in the dictionary attack, which dynamically adjusts the strategy during the guessing process. In 2023, Xu et al. [18] propose a bi-directional-transformer-based guessing framework and design three attack-specific fine-tuning models for CPG, TPG, and ARPG. Similarly, they construct neural networks to model the adaptive relationship between passwords and rules. They constructed a BERT-based classification model that outputs a continuous value between 0 and 1 that measures the fitness of the rule to the word.

In summary, both methods aim to establish mappings between passwords and rules and dynamically adjust the rules during the guessing process. This essentially leverages the **character-level password locality**. Because string-similar passwords are clustered in the character space, we can generate nearby passwords using mangling rules. However, there are several issues that are worth discussing: (1) Definition of mangling rules (transformation policies) often requires domain expert knowledge. (2) When neural networks are used to derive the adaptation rule set, a larger number of rules leads to a larger output dimension and the problem of feature sparsity. (3) When only rule transformations are used, as in tools such as Hashcat, it is inefficient to sample and transform all rules uniformly. Because in dynamic password guessing we don't know which rules apply to the target set and which are more effective. To cope with the above problems, in this paper, we design the password transformation policy automatic generation algorithm to automatically generate policies (mangling rules) that are applicable to the character-level password locality. Meanwhile, we design a policy distribution optimization scheme based on PSO [59] algorithm to optimize the policy distribution and dynamically adapt the dictionary.

Generative Model based Method. In 2021, Pasquini et al. [16] introduce an expectation maximization-inspired framework that can dynamically adapt the estimated password distribution to match the distribution of the attacked password set. The authors introduce the concept of weak locality of passwords and dynamically generate password guesses based on weak locality. They train a Generative Adversarial Network (GAN) to learn the representation of passwords in the potential

space as well as to generate guesses. By controlling the latent distribution, the method can increase the probability of the region that may be covered by passwords from the target distribution.

In summary, the weak locality of passwords is actually **feature-level password locality**, where semantically similar passwords are close in the representation space of the neural network. The higher the probability of generating the target password, the higher the probability of generating the surrounding passwords. However there are several issues worth discussing: (1) The approach by sampling and generating is applies to GANs but is difficult to apply to other models such as Markov, PCFG, and RNN. (2) As more passwords are cracked, the authors' proposed method did not learn the new passwords, and the model parameters are not updated. (3) The GAN model is unable to assign probabilities and rank the generated password guesses. To cope with the above problems, we design a feedback-based password learning approach based on incremental training, which is applicable to a variety of models including Markov, PCFG, and RNN. By adding the newly cracked passwords to the training set, the model can reduce the bias from the target distribution. Models such as Markov, PCFG, and RNN can assign probabilities to and rank the generated password guesses.

Finally, we make full use of feature-level and character-level password locality. We iteratively learn and generate password guesses at the feature level and character level based on co-training to further improve generalization. We will introduce the co-training algorithm in the next subsection.

C. Co-training

Co-training [60], [61], [62] is a multi-view classification algorithm that improves the generalization performance of models by combining two (or more) classifiers trained from different views. Different views exchange the prediction labels of unlabeled samples to realize the information exchange. The co-training algorithm is based on two key assumptions. The one assumption is that each view contains enough information to build the optimal learner. The other assumption is that the two views are independent under the condition of a given class label. Although the process of the co-training algorithm is simple, the theory proves that if the two views satisfy the two key assumptions, the generalization of weak classifiers can be improved to any high level through co-training using unlabeled data [63]. Qiao et al. [64] present Deep Co-Training (DCT) based on the co-training framework for semi-supervised image recognition, which improves the accuracy of models. Katz et al. [65] propose an ensemble-based co-training approach that makes use of unlabeled text data to improve text classification when labeled data is very small.

The goal of co-training is to improve overall task performance through complementary learning of multiple base models. Although traditional co-training is applied to classification tasks, its theory can be extended to tasks in other domains. In this paper, we adopt **co-training for dynamic password guessing** stemming from the following intuitions: (1) Character-level password locality and feature-level password locality are different views of password distribution properties. (2) Feature and character level localities are independent of each other, and it is possible to build a password generation model from a single level of locality. (3) Feature and character-level locality information can complement each other during dynamic password guessing. That is, after guessing the target set at the feature level, the hit passwords can be provided to the character-level model for password transformation. Similarly, hit passwords at the character level can be provided to the feature-level model for incremental learning. (4) The above process can be carried out iteratively to improve the performance of the models at both levels, thus improving the overall dynamic guessing cracking rate.

III. Method

Threat Model. Password cracking attacks mainly include online targeted attacks [30] and offline trawling attacks [23], [24]. In this paper, we primarily consider the case of dynamic password guessing in the offline trawling attack scenario. In practice, the attacker obtains the password hash database of the target site by means of SQL injection, social worker database leakage, etc., expecting to use the existing password data to guess the target. The attacker uses the password generation model to generate a guess list and hashes these guesses using the hashing algorithm (e.g., MD5, Argon2) to compare with the cipher-text password. The attacker then adapts their model based on matching (hit) passwords. The above process is carried out iteratively so that the model fits the target data distribution better and better. Finally, the attacker uses the dynamically adapted model to generate a large amount of guesses ($>10^8$) to maximize the number of guessed passwords. $\beta-success-rate$ [66] is used to measure the average success rate when an attacker is limited to a maximum number of β guesses.

$$s_\beta(X) = \sum_{i=1}^{\beta} P(x_i) \tag{1}$$

Overview of CoT-DPG. In this section, we present CoT-DPG, our dynamic password guessing framework based on co-training. The framework of the proposed method is illustrated in Fig. 2. The framework includes three modules: *Password Incremental Training*, *Policy Distribution Optimization with PSO*, and *Co-Training*. In the password incremental training module, we design a feedback-based password learning approach based on incremental training at the feature level. We dynamically learn the distribution of the target set by continuously adding newly cracked passwords to the training set and incrementally training the password generation model with feedback information. In the policy distribution optimization with PSO module, we design the password transformation policy automatic generation algorithm and PSO-based policy distribution optimization algorithm at the character level. We optimize policy distribution during dynamic guessing to generate more efficient password guesses. In the co-training module, model incremental learning and policy distribution

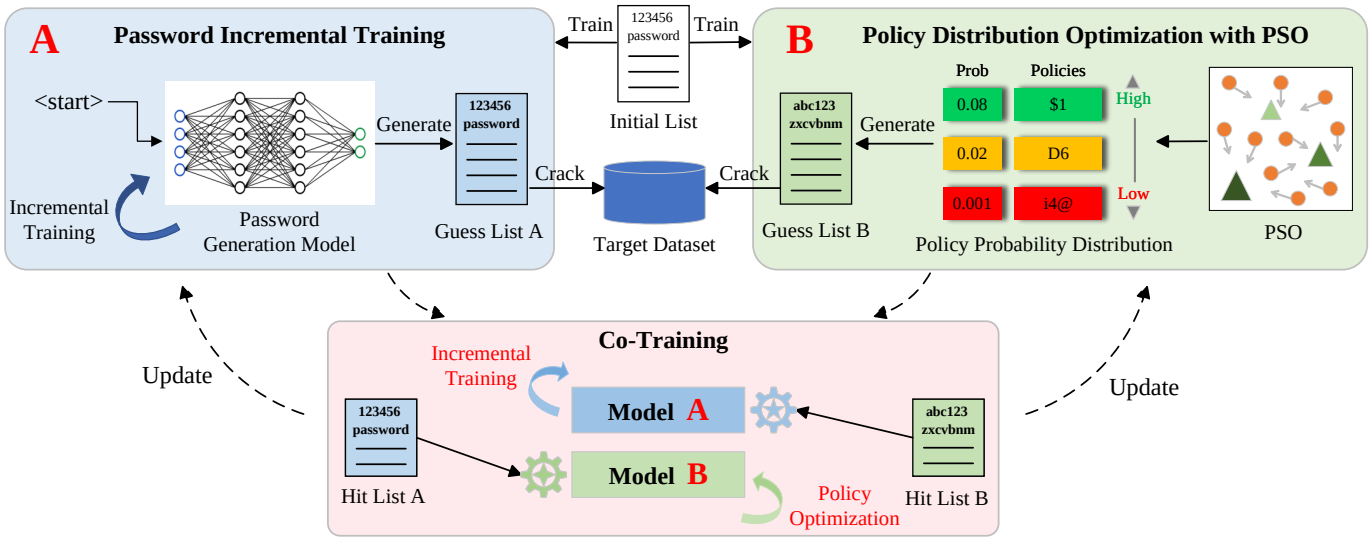


Fig. 2. The dynamic password guessing framework CoT-DPG.

optimization alternate to complement each other's knowledge, and dynamic password guessing is achieved through multiple rounds of iterations. The proposed method takes full advantage of the dual locality of passwords at the feature level and character level.

A. *Password Incremental Training*

Deep neural networks are generalized for generative tasks and can generate text that does not appear in the training set in addition to fitting the training set distribution. In addition to this, the weights of a deep learning network can be dynamically learned and updated. That is, if we crack more passwords and extend the training set, the neural network can fit the new training set distribution to encode more password knowledge. Therefore, it is feasible to dynamically update the parameters of the password generation model through incremental learning and gradually generate more hit password guesses.

Feature-Level Password Locality. Deep neural networks represent passwords in potential space, and passwords with similar features are similar in potential space. Thus when a particular password is generated with a high probability, passwords that are close to it in the potential space may also be generated. Even if the password does not appear in the training set, the neural network will have some generalization ability to generate it. This is the locality of passwords at the feature level.

Incremental Training. In general-purpose password guessing, the model does not change once it is trained, but such a static model is not applicable to dynamic password guessing. Incremental training [67], [68], [69] allows the model to be changed and updated dynamically. When new data arrives, the model adjusts its parameters to the new data distribution by learning the features of the new samples. Incremental training is divided into full incremental training and partial incremental training. Full incremental training refers to adding new samples to the training set and training on the expanded training set. Partial incremental training refers to training on newly acquired samples. Since the amount of password data used for training in the dynamic password guessing scenario is not very large, and in order to prevent the model from forgetting the old knowledge, we adopt the full incremental training approach. For neural network models, a parametric function is used to estimate the distribution of passwords:

$$p(X) = p(X;\theta) \tag{2}$$

Where θ is the set of parameters of the neural network, $p(X)$ is the distribution of the train set. In the case of recurrent neural networks, the model predicts the next character in the string. Assuming the input password x to the network is '<s>abc123' and <s>' is the start symbol. Then the correct output (label) of the network is 'abc123<e>' with '<e>' as the end symbol. The label is compared with the predictions of the model and the loss is calculated by cross entropy. After obtaining the losses, the parameters of the network are optimized using gradient descent. All training samples play a role in the optimisation of the network parameters. In this way, as we obtain more new cracked passwords, we can update θ to θ' through the learning process of the neural network. The model can adapt to the new training set distribution $p(X')$.

$$p(X') = p(X';\theta') = p(X + \Delta X;\theta') \tag{3}$$

Where ΔX is the incremental part of passwords, θ' is the updated model parameter, and $p(X')$ is the updated distribution of passwords. This feedback-based incremental training allows us to continuously generalize new cracked passwords, making the distribution of passwords learned by the model more and more adapted to the target dataset. The generalization ability of the model is improved by learning more samples. In combination with feature-level password locality, the model will be able to generate more hit password guesses after each round of iterations.

The feedback-based incremental training approach is applicable to almost all data-driven password guessing models. Data-driven models almost invariably fit the distribution of the training set by building a probabilistic model. Then the probability distribution function can be changed by changing the training set. In the subsequent experimental validation of this paper, we explore the effectiveness of the incremental training approach on four models, Markov, PCFG, FLA, and GPT. After incremental training, all four models are able to generate more hit password guesses, and the generalization ability of the models is further enhanced. During incremental training, the expansion of the training set and the timely addition of newly hit passwords enable the model to better fit the target distribution.

B. Policy Distribution Optimization with PSO

At the character level, we aim to further extend the generation through password locality. For the surrounding passwords that cannot be generated at the feature level, we generate them at the character level by means of password policy transformation/mangling rule extension.

Character-Level Password Locality. Character-level password locality means that passwords that are similar in structure and character are clustered together in character space. Some of the passwords in this cluster can be transformed into other passwords by the password transformation policy. The password transformation policy, also known as the mangling rule, refers to changing the password string according to a certain transformation. For example, 'pass123' can be transformed into 'p@ss123' by the policy 'sa@' (replacing 'a' with '@'). The reason for this locality of passwords at the character level is that users tend to construct passwords with local modifications on popular passwords that are easy to remember. This is why dictionary attacks can be successful. Meanwhile, this habit of local modification by users is common enough to make dictionary attacks feasible in the dynamic password guessing scenario. This is because we can extract these local transformation policies on other leaked datasets and apply them by cracking the unknown target set. In the following, we describe how our work automatically generates password transformation policies based on data-driven without expert predefinition.

Password Transformation Policy Generation. The algorithm we designed for the automatic generation of password transformation policies is data-driven, so we analyze password data from other leaked datasets. These passwords are clustered together in character space and are similar in character structure. Therefore, we first find these similar password pairs by calculating the string similarity between the passwords, which is used for the next step of extracting password transformation policies. Levenshtein distance [70], [71], also known as edit distance, refers to the minimum number of edit operations required to convert from one to the other between two strings. Levenshtein distance is commonly used to calculate the similarity between two strings, and the Levenshtein distance is defined as follows:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j), & if \quad min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+l_{a_i \neq b_j} \end{cases} & else. \end{cases} \tag{4}$$

Where $l_{a_i \neq b_j}$ is an indicator function equal to 0 when $a_i = b_j$ and equal to 1 in all other cases. $lev_{a,b}(i,j)$ represents the Levenshtein distance from the first i characters of string a to the first j characters of string b.

We then use the string analysis tool SequenceMatcher, a sequence-matched library in Python. This tool helps us to analyze the differences in similar password pairs. Inputs two strings to be compared and outputs the result of the comparison, which contains four types of tag information: 'equal', 'insert', 'delete', 'replace' and four positional information i_1, i_2, j_1, j_2. These tag information and position information can help us to construct password transformation policies. For example, input the strings 'pass123' and 'pass123!' to the SequenceMatcher function, and we get an output '[('equal', 1, 7, 1, 7), ('insert', 7, 7, 7, 8)]'. This means that both strings are the same from the first to the seventh position, with the character '!' inserted in the eighth position. It is possible to derive the policy '$!', and this policy refers to inserting the character '!' at the end of the password. Similarly, we extract password transformation policies on each pair of similar passwords.

Policy Distribution Optimization with PSO. How to sample the appropriate policy set from a large number of policies is an issue worth investigating. Obviously, a uniform distribution of all policies is not appropriate, because the efficiency of each policy is different. In addition to this, we have no information about the target set, we don't know which policies are effective and which ones are not. Therefore the policy distribution needs to be optimized and the sampling probability of effective policies should be increased while the sampling probability of ineffective policies should be decreased.

Swarm Intelligence (SI) has attracted the interest of many researchers in various fields [72], showing the potential to solve many optimization problems. Particle Swarm Optimization (PSO) is an optimization technique proposed by Kennedy and Eberhart in 1995 [59]. It uses a simple mechanism that mimics the flocking behavior of birds and fishes to guide the particles in their search for a globally optimal solution. The PSO algorithm first initializes the population. The second step is to calculate the fitness value of each particle, update the individual and global optimal values, and update the velocity and position of the particles. The second to fourth steps are repeated until the termination condition is satisfied [73], [74]. We apply the PSO algorithm to the task of optimizing password policy distribution, with the aim of generating more valid password guesses by choosing an optimal policy probability distribution for a given list of words. The PSO-based policy distribution optimization algorithm is shown in Fig. 3.

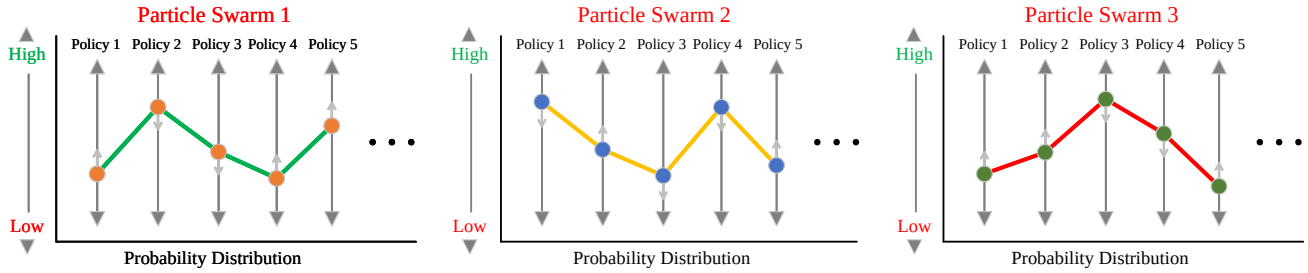


Fig. 3. The policy distribution optimization algorithm based on PSO.

Firstly we need to initialize multiple particle swarms. We define each particle to represent a password transformation policy, and the position of the particle represents the sampling probability of the corresponding policy. The particle swarm is for all password policies, and the sum of the positions of all particles in the swarm is 1, that is, the sum of the sampling probabilities of all policies is 1. We use multiple particle swarms to explore the globally optimal probability distribution. Each particle updates its position during the iteration process, i.e., the sampling probability. We set the position $x[i][j]$ of each particle in each swarm to a random value and normalize the probability distribution $x[i]$ of all particles in each swarm to 1. In order to evaluate the goodness of the policy distribution, we need to set up a password generation function $Generate$ and an evaluation function $Object\_function$. The generator function samples the policies based on the policy distribution $x[i]$ and applies them with the initial password list $pwds$ to generate password guesses $Gen[i] = Generate(x[i], pwds)$. The evaluation function is used to evaluate the cracking rate of the generated guess list.

Secondly, we generate password guesses and evaluate the cracking rate. We use multiple particle swarms to explore different password transformation policy probability distributions. We sequentially generate password guesses for each particle swarm and evaluate them. Specifically, we sample a fixed number of policies based on the probability distribution of each particle swarm without playback and generate password guesses. We then verify the cracking rate of these password guesses on the validation set. The local score for each particle will be obtained by multiplying the score of the particle swarm by the probability of that particle. Naturally, the higher the sampling probability of a good policy, the higher its contribution, the higher its score, and the higher the overall crack rate will be. Meanwhile, the particle will locate its optimal position $L_{best}[i][j]$ based on the score of each round. After evaluating each particle swarm sequentially, the particle swarm with the highest cracking rate S_{best} and the optimal policy distribution $G_{best}[i]$ are obtained.

Thirdly, we update the position for each particle in the swarm based on the evaluation results:

$$v[i][j] = w \times v[i][j] + c_1 \times r_1 \times (L_{best}[i][j] - x[i][j]) + c_2 \times r_2 \times (G_{best}[j] - x[i][j]) \quad (5)$$

$$x[i][j] = x[i][j] + v[i][j] \quad (6)$$

Where $v[i][j]$ is the velocity of the particle, w is the habitual factor and it controls the inertia of the particle to maintain its current state of motion. The habitual factor w is a constant that we can initialize to 0.9. c_1 is the cognitive coefficient and c_2 is the social coefficient. c_1 and c_2 are two constants that control the rate at which the particles move towards the local optimum and the population optimum. r_1 and r_2 are two random numbers between 0 and 1. The PSO-based policy distribution optimization algorithm is shown in Algorithm 1.

After the PSO-based policy distribution optimization algorithm is executed, we can then obtain the optimal policy sampling probability distribution. We sample a fixed number of password transformation policies based on this sampling probability distribution and apply them to a password list to generate password guesses.

C. Co-training

Based on the co-training algorithm, incremental learning and policy distribution optimization are performed iteratively, and both complement each other. The co-training process is shown in Fig. 4.

The process is as follows: (1) At the feature level, the password generation model generates guess list A. Here we can choose password generation models such as FLA, Markov, PCFG, etc. (2) The guess list A is used to crack the target dataset and and the hit passwords compose hit list A. (3) We optimize the distribution of password policies based on the hit list A. Here we use the PSO algorithm for optimization. (4) At the character level, the guess list B is generated based on the updated policy distribution. (5) The guess list B is used to crack the target dataset and the hit passwords compose hit list B. (6) We incrementally train the password generation model based on the hit list B. (7) The above process is repeated iteratively.

The co-training process makes full use of feature and character-level password locality. Based on the password guesses generated at the feature level, more hit passwords can be extended at the character level using policy transformations. Similarly, character-level extended passwords can be used for training the password generation model. The model encodes more password information and uses its powerful generalization capabilities to further generate more hit password guesses. Executing the PSO algorithm on more hit passwords makes the distribution of password transformation policies more

Algorithm 1 The PSO based Policy Distribution Optimization Algorithm.

Input:
Number of particle swarms: m,
Number of particles in each swarm: n,
The habitual factor: w,
The cognitive coefficient: r_1,
The social coefficient: r_2,
Number of iterations: N,
Initial password list: $pwds$.
Output:
The best score: S_{best},
The optimal policy distribution: G_{best}.
Process:

for $iter = 0; iter < N; iter++$ **do**
  for $i = 0; i < m; i++$ **do**
    $Gen[i] = Generate(x[i], pwds)$
    $S[i] = Object\_function(Gen[i])$
    if $S[i] > S_{best}$ **then**
      $S_{best} = S[i]$
      $G_{best} = x[i]$
    end if
    $x_{sum} = 0$
    for $j = 0; j < n; j++$ **do**
      $s[i][j] = S[i] \times x[i][j]$
      if $s[i][j] > s_{best}[i][j]$ **then**
        $s_{best}[i][j] = s[i][j]$
        $L_{best}[i][j] = x[i][j]$
      end if
      $r_1 = random(), r_2 = random()$
      $v[i][j] = w \times v[i][j] + c_1 \times r_1 \times (L_{best}[i][j] - x[i][j]) + c_2 \times r_2 \times (G_{best}[j] - x[i][j])$
      $x[i][j] = x[i][j] + v[i][j]$
      $x_{sum} = x_{sum} + x[i][j]$
    end for
    for $j = 0; j < n; j++$ **do**
      $x[i][j] = x[i][j]/x_{sum}$
    end for
  end for
end for

complete. Also, larger and more efficient word lists enable the policy transformation to generate more valid password guesses. Because dictionary attacks are sensitive to the word list and rule list.

IV. Evaluation

In this section, we conduct a systematic experiment to evaluate the effectiveness of CoT-DPG, the co-training based dynamic password guessing framework. Our method generates password guesses through co-training at the feature and character level for dynamic password guessing. We compare four existing models (Markov, PCFG, FLA, and PassGPT) for password generation at the feature level. We use a fourth-order Markov model, and in the FLA model, we use a three-layer 128-neuron network architecture. We also compare with the state-of-the-art methods **DPG** [16] and **ARPG** [18] in dynamic password guessing. In our experiments, we compare the cracking performance of dynamic password guessing, which is adapted through feedback, with that of static password guessing. We also compare the performance of password cracking with co-training at the feature and character level and feedback-based adapting at a single level. We analyze the effectiveness of the PSO-based policy distribution optimization algorithm and show its optimization process.

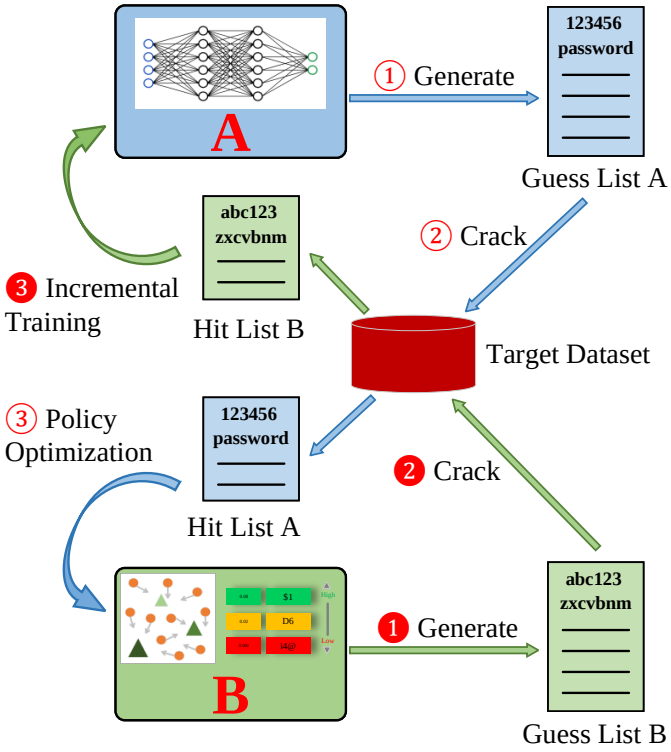


Fig. 4. The co-training process for incremental training and policy distribution optimization.

A. Experiment Setup

Datasets. Eight leaked password datasets are used in this paper, three from Chinese websites and five from English websites, containing a total of 66.9 million plain-text passwords. The detailed information of the eight password datasets is shown in Table I. We divide each password dataset into training and testing sets in the ratio of 1:1. When guessing on one of the datasets, only the training sets of the other datasets are utilized, guaranteeing that the target is unknown. Our experiments are conducted on a server with Intel(R) Xeon(R) Gold 6139 CPU, 377G memory, and TITAN RTX3090 GPU, 24G video memory.

Comparison Baseline. We fairly compare our CoT-DPG with its foremost counterparts (i.e., PassGAN (DPG) [17], PassBERT (ARPG) [18], GuessFuse [58], FLA [14], Markov [13], PCFG [12], and PassGPT [55]). Since the PassGAN and PassBERT are dynamic password guessing

methods, and GuessFuse is the hybrid guessing method, they can be compared directly. For the FLA, Markov, PCFG, and PassGPT methods, we train it incrementally and also put it into our CoT-DPG framework to form methods CoT-F, CoT-M, CoT-P, and CoT-G for comparison.

TABLE I
THE INFORMATION OF DIFFERENT DATASETS.

Dataset	Year	Language	Service type	Website	Size
000webhost	2015	English	Web Hosting	https://000webhost.com	14,451,798
Linkedin	2012	English	Social Platform	https://linkedin.com	7,660,392
ClixSense	2016	English	Paid to click	https://www.ysense.com	3,634,274
7k7k	2011	Chinese	Games	http://www.7k7k.com	14,785,569
Csdn	2011	Chinese	Programmer Forum	https://csdn.net	6,427,828
Dodonew	2011	Chinese	E-commerce	http://www.dodonew.com	16,103,478
BookCrossing	2022	English	Book site	https://bookcrossing.com	1,465,568
CraftRise	2023	English	Games	http://www.craftrise.com.tr	2,375,526

Ethical Considerations. The datasets used in this paper are publicly available and are widely used in research [75], [76], [77]. Although these data are widely used, they remain private. We therefore do not show the actual passwords associated with individual users in these datasets and use them for research purposes only. We report only the results of our experiments.

B. Evaluation of Dynamic and Static Guessing

Firstly, we compare static password guessing and dynamic password guessing. We use the other datasets to form the initial password guess list to crack the target dataset, and the hit passwords are used as the initial training set. Static password guessing refers to a cracking method that is trained only once. A large number of password guesses are generated to crack the target set directly after training the password generation model using the initial password list as the training set. Dynamic password guessing refers to multiple rounds of incremental training and updating the model based on feedback. In each iteration, the model generates password guesses to crack the target, and the hit passwords are added to the training set for incremental training. To speed up the dynamic password guessing, only a small number of password guesses are generated in each round (one-tenth of the final number), and a large number of guesses are generated for cracking in the final round. Duplicate password guesses are removed during the iteration process and only the non-duplicate password guesses generated in the last round are retained. We use the FLA model as the comparative password generation model. The complete guessing curves of dynamic and static guessing approaches are shown in Fig. 5.

We can see from Fig. 5 that dynamic password cracking has a much higher cracking rate than static password guessing. This indicates that the newly cracked passwords in each round provide the neural network with important information about the distribution of the target set. The neural network can encode more password information in each round of incremental learning. Also, its strong generalization capability allows the model to generate more hit passwords. Even with many passwords that are not in the training set, the neural network model has the ability to generate them, and the increase in cracking rates confirms this view.

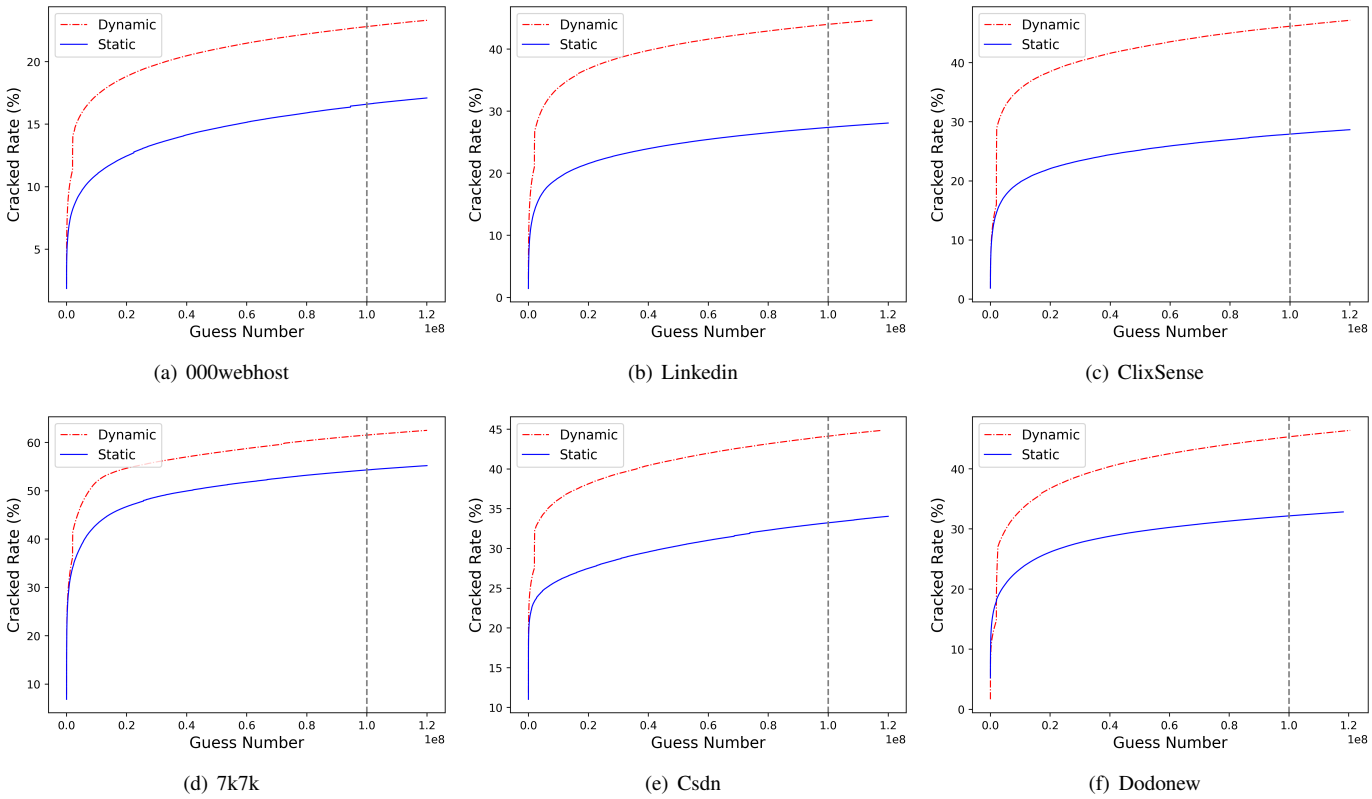
C. Evaluation of Dual-locality and Single-locality

In this subsection, we discuss the effectiveness of dual password locality. We compared the four methods as shown in the Table II. Where the FLA, Markov, PCFG, and PassGPT models are trained using dynamic incremental training. CoT-F, CoT-M, CoT-P, and CoT-G are co-training methods in the CoT-DPG framework using FLA, Markov, PCFG, and PassGPT with policy transformations respectively. Incremental training is performed at the feature level in rounds 1,3,5,7. Policy distribution optimization is performed at the character level in rounds 2,4,6. When guessing the target dataset, the initial training set is sampled from other datasets in order to ensure that the distribution information of the target dataset is completely unknown. The newly cracked passwords in each round are used as feedback and delivered to the model for incremental training and policy distribution optimization. Duplicate password guesses during iteration are removed. That is, only the cracking rate of non-repeated password guesses will be calculated in each iteration. The cracking rates of these methods for each round are recorded in Table II.

From the experimental results in Table II. we can draw the following conclusions:

(1) The dynamic password guessing framework CoT-DPG outperforms any single method. On all eight datasets and all four methods, the performance of password cracking using the co-training methods (CoT-F, CoT-M, CoT-P, and CoT-G) is better than the performance using incremental training alone. The password cracking rate with co-training relatively improves by 1.77% to 35.71% compared to no co-training. This illustrates the importance of dual locality of password and the single level of locality is deficient. In addition to this, we find that as the number of iterations increases, the cracking rate of the model grows larger at first and then grows slowly. This suggests that during the dynamic guessing process, when the target is unknown at the beginning, the newly cracked password can provide rich information enabling the model to quickly fit the distribution of the target dataset. Afterwards the newly cracked passwords become fewer and can provide less information. This is as illustrated in Fig. 1, after the model has been incrementally trained for several rounds, the generation of surrounding passwords for the target password reaches an upper bound. At this point generating more hit passwords through policy transformation is a solution to improve this upper bound. This is the reason why more passwords can be cracked using the co-training approach compared to the incremental training approach only.

(2) The password guessing performance varies between different models and different datasets. After incremental training, CoT-P performs better than the other models on the 000webhost, ClixSense, and Craftrise datasets. And CoT-F outperforms on the other five datasets. We note that the password cracking performance of the PCFG model is better than the other models on seven datasets when trained for only one round using the initial set of passwords. After incremental training, the FLA model outperforms the other



(a) 000webhost (b) Linkedin (c) ClixSense

(d) 7k7k (e) Csdn (f) Dodonew

Fig. 5. Complete guessing curves of dynamic and static guessing approaches.

models on five datasets. This indicates that the neural network is sensitive to the training samples. The neural network does not fit the target distribution well when the training data is small, and the generalization ability of the model is enhanced with more training data. The password cracking rate has the largest absolute improvement rate of 8.77% on the CoT-G model, exceeding the 4.20% on CoT-M and the 3.35% on CoT-P. This suggests that in the co-training framework, while FLA, Markov, PCFG, and PassGPT methods, and policy transformations are all able to complement each other, neural networks are better integrated with policy transformations. This also confirms the view of this paper that feature-level and character-level password locality can complement each other well.

Our CoT-DPG also achieved significant success on the BookCrossing and Craftrise datasets in 2022 and 2023. Although the leak time is different for each dataset and the password distribution may change, the password locality is an invariant property. This is the reason why dynamic password guessing can be successful. In addition, when guessing on unknown sites, the other datasets are only used to provide the initial set of passwords, while fitting the target distribution relies on dynamic learning and updating of the model. Therefore, the time attribute of the dataset has little effect on the experimental results. Our framework is robust and scalable.

(3) The proposed CoT-DPG is highly applicable and transferable. The CoT-DPG proposed in this paper is a dynamic password guessing framework that allows multiple models to learn and complement each other. The models from different views learn from each other and iteratively train to achieve better dynamic password guessing performance. Therefore, in CoT-DPG, as long as effective password guessing models can be constructed from different views, co-training is able to integrate the two well. In this paper, feature level and character level locality are two different views. CoT-DPG can even be applied to the hybrid password guessing scenario, fusing knowledge from multiple models to take full advantage of the strengths of multiple models.

We compare CoT-DPG with the state-of-the-art dynamic guessing methods **PassGAN (DPG)** proposed in [16] and **PassBERT (ARPG)** proposed in [18]. We also compare CoT-DPG with the hybrid guessing method **GuessFuse**. After the 7-th round of training, we generate more than 10^8 password guesses to observe the password cracking performance of various methods under a large number of guesses. The complete guessing curves of different methods in dynamic password guessing after 7 iterations are shown in Fig. 6. We can see that the password cracking performance of CoT-DPG exceeds the three methods on all eight datasets. And the password cracking rates of co-training methods are all higher than that of single model. The policy transformation at the character level has the lowest cracking rate after multiple rounds of feedback optimization. The highest password cracking rate is achieved after multiple rounds of co-training (CoT-F) of the

TABLE II
PASSWORD CRACKING RATES FOR DIFFERENT NUMBER OF ITERATIONS FOR CO-TRAINING USING PASSWORD DUAL LOCALITY AND DYNAMIC UPDATING USING SINGLE LOCALITY.

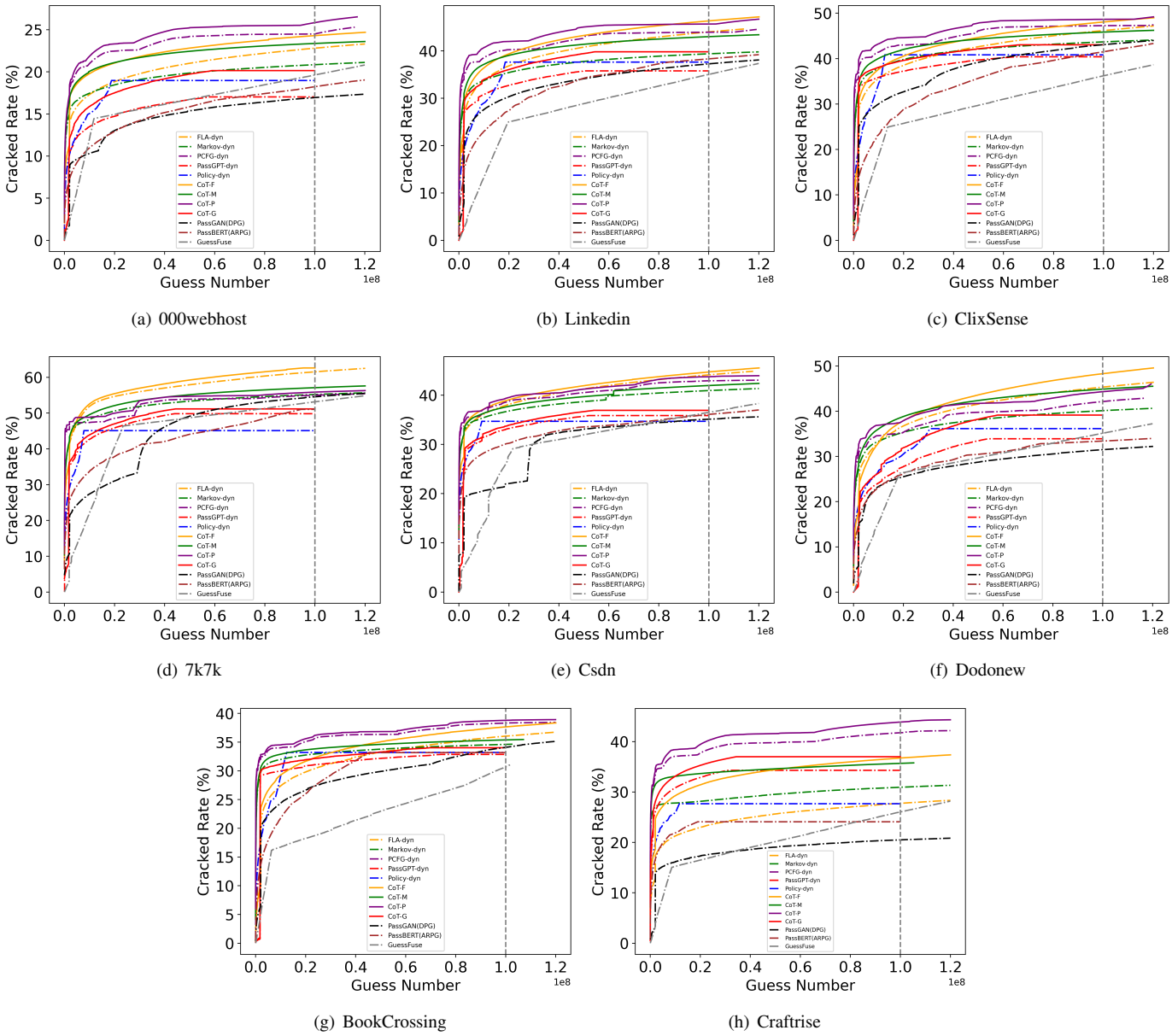
000webhost	FLA	CoT-F	Markov	CoT-M	PCFG	CoT-P	PassGPT	CoT-G	Linkedin	FLA	CoT-F	Markov	CoT-M	PCFG	CoT-P	PassGPT	CoT-G
iter_1	0.1589	0.1589	0.1721	0.1721	0.1922	0.1922	0.1554	0.1554	iter_1	0.3474	0.3474	0.3508	0.3508	0.3835	0.3835	0.3347	0.3347
iter_2	0.1749	0.1812	0.1767	0.1931	0.2064	0.2062	0.1588	0.1802	iter_2	0.3668	0.3818	0.3562	0.3819	0.3965	0.4019	0.3409	0.3703
iter_3	0.1832	0.1915	0.1776	0.1995	0.2088	0.2180	0.1598	0.1827	iter_3	0.3789	0.4023	0.3573	0.3906	0.3978	0.4129	0.3429	0.3749
iter_4	0.1879	0.2002	0.1779	0.2046	0.2093	0.2220	0.1605	0.1903	iter_4	0.3854	0.4091	0.3576	0.3973	0.3980	0.4189	0.3441	0.3831
iter_5	0.1927	0.2066	0.1781	0.2062	0.2094	0.2246	0.1609	0.1912	iter_5	0.3911	0.4176	0.3577	0.3997	0.3981	0.4205	0.3448	0.3845
iter_6	0.1969	0.2088	0.1782	0.2082	0.2143	0.2257	0.1612	0.1941	iter_6	0.3951	0.4198	0.3577	0.4012	0.3981	0.4218	0.3454	0.3877
iter_7	0.1997	0.2164 (8.36%↑)	0.1782	0.2088 (17.17%↑)	0.2145	**0.2264 (5.55%↑)**	0.1615	0.1946 (20.50%↑)	iter_7	0.3984	**0.4257 (6.85%↑)**	0.3584	0.4020 (12.17%↑)	0.3981	0.4221 (6.03%↑)	0.3459	0.3884 (12.29%↑)
ClixSense	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**	**7k7k**	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**
iter_1	0.4042	0.4042	0.3984	0.3984	0.4227	0.4227	0.3903	0.3903	iter_1	0.5023	0.5023	0.4781	0.4781	0.4787	0.4787	0.4586	0.4586
iter_2	0.4114	0.4235	0.4012	0.4180	0.4274	0.4385	0.3936	0.4124	iter_2	0.5279	0.5098	0.4908	0.4885	0.4854	0.4903	0.4730	0.4735
iter_3	0.4190	0.4329	0.4016	0.4220	0.4286	0.4437	0.3944	0.4149	iter_3	0.5420	0.5433	0.4942	0.5026	0.4861	0.4979	0.4752	0.4854
iter_4	0.4228	0.4375	0.4018	0.4263	0.4288	0.4478	0.3948	0.4200	iter_4	0.5515	0.5536	0.4954	0.5049	0.4862	0.5007	0.4760	0.4886
iter_5	0.4252	0.4429	0.4018	0.4272	0.4288	0.4489	0.3951	0.4206	iter_5	0.5557	0.5642	0.4959	0.5091	0.4862	0.5016	0.4765	0.4904
iter_6	0.4274	0.4446	0.4018	0.4289	0.4288	0.4498	0.3953	0.4222	iter_6	0.5579	0.5672	0.4961	0.5099	0.4862	0.5023	0.4769	0.4912
iter_7	0.4290	0.4482 (4.48%↑)	0.4044	0.4308 (6.53%↑)	0.4288	**0.4501 (4.97%↑)**	0.3955	0.4226 (6.85%↑)	iter_7	0.5591	**0.5698 (1.91%↑)**	0.5009	0.5164 (3.09%↑)	0.4862	0.5025 (3.35%↑)	0.4771	0.4919 (3.10%↑)
Csdn	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**	**Dodonew**	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**
iter_1	0.3587	0.3587	0.3568	0.3568	0.3637	0.3637	0.3504	0.3504	iter_1	0.3226	0.3226	0.3205	0.3205	0.3304	0.3304	0.2969	0.2969
iter_2	0.3718	0.3663	0.3618	0.3659	0.3688	0.3726	0.3523	0.3596	iter_2	0.3601	0.3606	0.3370	0.3585	0.3463	0.3615	0.3079	0.3452
iter_3	0.3802	0.3800	0.3633	0.3711	0.3699	0.3774	0.3528	0.3610	iter_3	0.3746	0.3877	0.3424	0.3731	0.3514	0.3735	0.3109	0.3528
iter_4	0.3848	0.3820	0.3640	0.3729	0.3702	0.3797	0.3531	0.3633	iter_4	0.3864	0.3982	0.3446	0.3826	0.3533	0.3819	0.3125	0.3673
iter_5	0.3890	0.3889	0.3643	0.3743	0.3703	0.3806	0.3532	0.3637	iter_5	0.3981	0.4152	0.3456	0.3869	0.3540	0.3848	0.3134	0.3691
iter_6	0.3922	0.3893	0.3644	0.3749	0.3703	0.3814	0.3534	0.3644	iter_6	0.3998	0.4183	0.3462	0.3904	0.3544	0.3882	0.3141	0.3730
iter_7	0.3948	**0.4018 (1.77%↑)**	0.3673	0.3783 (2.99%↑)	0.3703	0.3816 (3.05%↑)	0.3535	0.3646 (3.14%↑)	iter_7	0.4078	**0.4282 (5.00%↑)**	0.3472	0.3923 (12.99%↑)	0.3546	0.3892 (9.76%↑)	0.3147	0.3740 (18.84%↑)
BookCrossing	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**	**Craftrise**	**FLA**	**CoT-F**	**Markov**	**CoT-M**	**PCFG**	**CoT-P**	**PassGPT**	**CoT-G**
iter_1	0.3300	0.3300	0.3275	0.3275	0.3445	0.3445	0.3226	0.3226	iter_1	0.2062	0.2062	0.2613	0.2613	0.3170	0.3170	0.2281	0.2281
iter_2	0.3360	0.3499	0.3286	0.3364	0.3483	0.3506	0.3240	0.3333	iter_2	0.2098	0.2580	0.2755	0.2943	0.3515	0.3378	0.2543	0.2650
iter_3	0.3391	0.3553	0.3288	0.3380	0.3489	0.3541	0.3243	0.3348	iter_3	0.2214	0.2724	0.2784	0.3172	0.3613	0.3699	0.2727	0.2935
iter_4	0.3407	0.3580	0.3288	0.3388	0.3489	0.3547	0.3244	0.3357	iter_4	0.2342	0.2981	0.2792	0.3239	0.3651	0.3732	0.2872	0.3044
iter_5	0.3415	0.3598	0.3288	0.3390	0.3490	0.3552	0.3245	0.3360	iter_5	0.2386	0.3101	0.2795	0.3305	0.3707	0.3819	0.2977	0.3225
iter_6	0.3424	0.3602	0.3288	0.3391	0.3490	0.3552	0.3246	0.3362	iter_6	0.2435	0.3208	0.2795	0.3326	0.3725	0.3829	0.3046	0.3262
iter_7	0.3431	**0.3614 (5.33%↑)**	0.3288	0.3391 (3.13%↑)	0.3490	0.3553 (1.81%↑)	0.3247	0.3363 (3.57%↑)	iter_7	0.2456	0.3333 (35.71%↑)	0.2795	0.3349 (19.82%↑)	0.3733	**0.3910 (4.74%↑)**	0.3095	0.3361 (8.59%↑)

FLA model and policy transformation on the four datasets. The highest cracking rate is achieved after the PCFG and policy transformation are co-trained (CoT-P) on the other four datasets. This shows that by cracking more passwords at the feature level and character level, the model encodes more information and is able to fit the target set distribution better.

D. Evaluation of Policy Distribution Optimization using PSO

At the character level, we use the PSO algorithm to optimize the policy distribution, thereby mitigating the blindness of the selection process of password transformation policies. Intuitively sampling on a more optimal distribution of policies will generate a more efficient list of password guesses, which will then have a correspondingly higher cracking rate. We execute the policy distribution optimization algorithm on the set of hit passwords generated by FLA for the first time and show the optimization results in Fig. 7. The figure shows the growth curve of the password cracking rate over a total of 50 iterations and the probability distribution of the 300 password transformation policies after the PSO algorithm has been executed. From the figure, we can see that the password cracking rate is gradually increasing with the execution of the PSO-based policy optimization algorithm. We set up a total of 5 particle swarms, each with 300 particles. Each particle represents a policy, and its position represents the sampling probability of the policy, and all the particle positions are the probability distribution of the policy sampling. Each particle explores its local optimal position and all particles approach the global optimal position. Our objective function is the cracking rate, so during the execution of the algorithm, the cracking rate gets higher and higher. It can also be seen from the figure that the set of policies applicable to each dataset is different. When we are unknown about the target dataset, it is necessary to perform the optimization of the policy distribution so as to quickly obtain the policy set adapted to the target. Since our policies are data-driven and automatically generated, most of them are valid. Thus a small number of policies have a very low probability of being sampled in the policy probability distributions shown in the figure. In summary, the PSO-based policy distribution optimization algorithm is effective in optimizing the policy distribution, generating more hit passwords, and further improving the cracking rate. In addition, by increasing the search space (600 policies × 10 swarms) from the original 300 policies × 5 swarms, the success rate of guessing increases slightly, but significantly increases the overhead (+400%). To balance the overhead and performance, we chose the parameters of 300 policies and 5 swarms.

Overhead analysis. We conduct rigorous cost analysis covering training time and generation rate on a server with a single RTX 3090 GPU. The results are shown in Table III.



(a) 000webhost (b) Linkedin (c) ClixSense

(d) 7k7k (e) Csdn (f) Dodonew

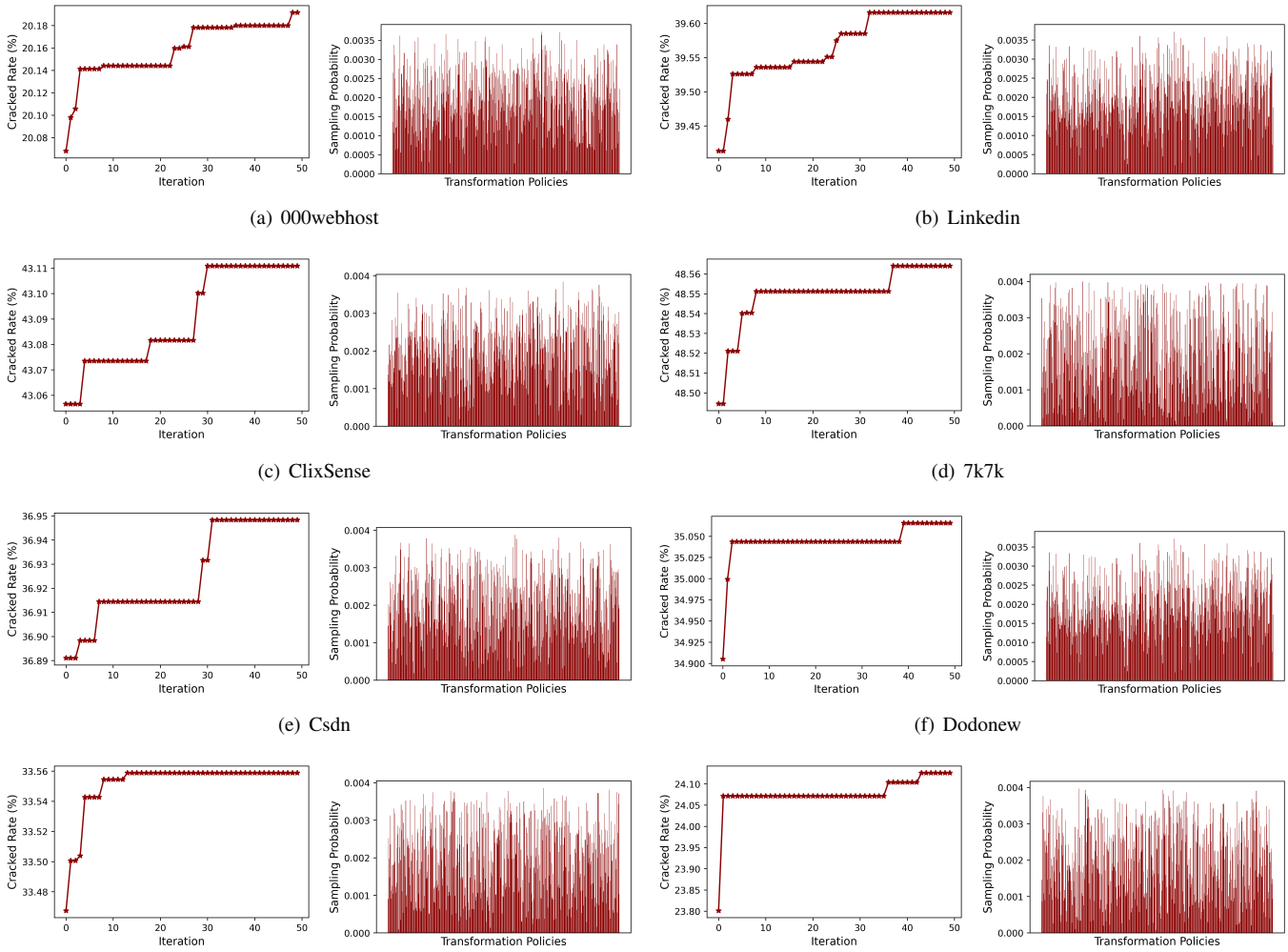
(g) BookCrossing (h) Craftrise

Fig. 6. Complete guessing curves of different methods in dynamic password guessing after 7 iterations. We compare two approaches to dynamic password guessing that use only feature-level password locality for incremental learning and iterative learning using the dual-locality. We also compare our method with the state-of-the-art **PassGAN (DPG)**, **PassBERT (ARPG)**, and **GuessFuse**.

CoT-F, CoT-M, CoT-P, and CoT-G methods under the CoT-DPG framework are 1.2 to 2.4 times faster than the iteratively updated FLA, Markov, PCFG and PassGPT methods. This is due to the fast training process of PSO-based policy optimization and the fast password generation process based on the transformation policy. The CoT-DPG framework does not introduce more overhead, but instead reduces the time overhead while improving the guessing performance.

V. Discussion

Password dual-locality. The dual locality of passwords is an important property found in this paper. Existing work has presented the locality of passwords at the feature level, which guides dynamic password guessing through GANs. However real-world dictionary attacks and transformations via mangling rules actually exploit the locality of passwords at the character level. Thus more passwords can be guessed by local transformations at the character level. The experiments in this paper also confirm that single-level locality is not comprehensive enough and that fusion of feature and character level locality can achieve better DPG performance. The locality of passwords stems from the fact that the distribution of passwords is clustered rather than uniformly distributed. The distance between the clustered passwords in character space and feature space is small.



(a) 000webhost (b) Linkedin

(c) ClixSense (d) 7k7k

(e) Csdn (f) Dodonew

(g) BookCrossing (h) Craftrise

Fig. 7. Crack rate increase curves during a total of 50 iterations of the PSO algorithm and the probability distribution figures of 300 password transformation policies after the algorithm has been executed.

TABLE III
THE TIME OVERHEAD OF DIFFERENT METHODS.

Method	Item	Rate	Total Time	Method	Total Time	Speedup
FLA-dyn	Training Generation	2hours 1.6×10^3 pw/s	26hours (7iters)	**CoT-F**	11hours (7iters)	2.4times
Markov-dyn	Training Generation	1hour 7.1×10^3 pw/s	10hours (7iters)	**CoT-M**	6hours (7iters)	1.7times
PCFG-dyn	Training Generation	10mins 6.7×10^4 pw/s	1.5hours (7iters)	**CoT-P**	1.3hours (7iters)	1.2times
PassGPT-dyn	Training Generation	30mins 4.8×10^3 pw/s	8hours (7iters)	**CoT-G**	5hours (7iters)	1.6times
Policy-dyn	Training Generation	10mins 1.4×10^6 pw/s	2hours (7iters)	**PassGAN (DPG)** **PassBERT (ARPG)**	4hours 8hours	—— ——

Co-training framework. The co-training based DPG framework proposed in this paper makes full use of password dual-locality. The co-training algorithm utilizes models constructed from different views for iterative learning to complement each other's knowledge. The locality of passwords at the feature level and the character level are exactly the properties that passwords exhibit in the two different views, which can complement each other. This naturally applies to co-training. Co-training is a multi-view learning method, and multi-view learning aims to improve learning performance by exploiting different views of the same input data. From this perspective, the password dataset can then be observed from different

views. The representation of password data in different feature spaces is from different views, and different password guessing models also model the password distribution from different views.

View independence. The co-training method requires two views that are sufficiently redundant and satisfy conditional independence. The sufficiently redundancy of the two views is obvious, as password generation models can be constructed from both character space and feature space. For view independence, we analyse as follows: (1) We perform statistical tests by calculating the *Jaccard* correlation coefficients of generated and hit passwords from different views, as shown in Table IV. The Jaccard coefficient is a measure of the similarity of sets, calculating the ratio of the intersection size to the union size, with values ranging from 0 to 1, with larger values indicating a higher degree of similarity. From the 7 rounds of iterations, the Jaccard coefficients of the password guesses generated by the policy transformation based method and the FLA method are very low, indicating that the percentage of same passwords generated is very small. The hit passwords only overlap by 50.6% to 64%. The above results illustrate the mutual independence property of character space and feature space in generating passwords. (2) Empirically, the feature-level locality captures the proximity of the password in vector space, whereas the character-level locality acts directly on the original character sequence, and their sources of information are different. (3) As the results show in Table II and Fig. 6, our co-training framework significantly outperforms the model using only a single view. This improvement strongly suggests that the two views carry complementary information, which is an important indication of view (approximate) independence and a necessary condition for co-training to be effective.

TABLE IV
JACCARD CORRELATION COEFFICIENTS OF THE GENERATED PASSWORD SETS AND HIT PASSWORD SETS FOR THE FLA METHOD AND THE POLICY METHOD.

$Jaccard$	**iter_1**	**iter_2**	**iter_3**	**iter_4**	**iter_5**	**iter_6**	**iter_7**
J_{gen}	0.018	0.015	0.024	0.026	0.025	0.026	0.027
J_{hit}	0.640	0.546	0.524	0.516	0.511	0.508	0.506

Suggestions. Our work explores the process of how to perform dynamic password guessing on unknown websites. Even if the distribution of the target is unknown, it is still possible to continuously expand the threat to the target by relying on other information as well as model learning. This is because there are some habitual behaviors of people in constructing passwords: (1) Users don't update their passwords for a long time. (2) Users tend to use the same or similar passwords at multiple sites. (3) Users are accustomed to making simple modifications to popular passwords. Therefore, users need to avoid several of the above mentioned behaviors of constructing passwords. In addition to this, users need to be aware of common passwords and their substrings to avoid adopting them when constructing passwords..

Future work. In future work, we will explore the following components: (1) Further explore the essential properties of password data and investigate the applicable methods based on the properties. (2) Explore approaches for better integration of multi-view features of passwords. (3) Co-training is actually a semi-supervised learning approach, and we will further explore how to better introduce semi-supervised learning into password guessing.

VI. CONCLUSION

In this paper, we propose CoT-DPG, a co-training based dynamic password guessing framework for efficient guessing of unknown sites. We make full use of the password dual-locality at the feature level and the character level, and construct password generation models from different views. At the feature level, we design a feedback-based password learning approach based on incremental training. At the character level, we design an automatic password policy generation algorithm and a PSO-based password policy distribution optimization algorithm. Based on co-training algorithm, models from different views iteratively learn and complement each other. CoT-DPG effectively combines multiple password guessing models and employs co-training, a semi-supervised learning method, to achieve efficient dynamic password guessing.

REFERENCES

[1] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE transactions on dependable and secure computing*, vol. 15, no. 4, pp. 708–722, 2016.

[2] W. Li, J. Tan, W. Meng, and Y. Wang, "A swipe-based unlocking mechanism with supervised learning on smartphones: Design and evaluation," *Journal of Network and Computer Applications*, vol. 165, p. 102687, 2020.

[3] A. K. Jain, A. Ross, and S. Pankanti, "Biometrics: a tool for information security," *IEEE transactions on information forensics and security*, vol. 1, no. 2, pp. 125–143, 2006.

[4] A. Z. Zaidi, C. Y. Chong, Z. Jin, R. Parthiban, and A. S. Sadiq, "Touch-based continuous mobile device authentication: State-of-the-art, challenges and opportunities," *Journal of Network and Computer Applications*, vol. 191, p. 103162, 2021.

[5] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[6] ——, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 553–567.

[7] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto, "Who are you? a statistical approach to measuring user authenticity." in *NDSS*, vol. 16, 2016, pp. 21–24.

[8] C. Herley and P. Van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & privacy*, vol. 10, no. 1, pp. 28–36, 2011.

[9] S. Oesch and S. Ruoti, "That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020, pp. 2165–2182.

[10] J. Steube, "Hashcat," 2018. [Online]. Available: https://hashcat.net/hashcat/

[11] A. Peslyak, "John the ripper password cracker," 2021. [Online]. Available: https://www.openwall.com/john/

[12] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *2009 30th IEEE symposium on security and privacy*. IEEE, 2009, pp. 391–405.

[13] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 364–372.

[14] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean, and accurate: Modeling password guessability using neural networks," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 175–191.

[15] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*. Springer, 2019, pp. 217–237.

[16] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1382–1399.

[17] D. Pasquini, M. Cianfriglia, G. Ateniese, and M. Bernaschi, "Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 821–838.

[18] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han, "Improving real-world password guessing attacks via bi-directional transformers," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1001–1018.

[19] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[20] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*. Springer, 2003, pp. 617–630.

[21] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: Empirical results," *IEEE Security & privacy*, vol. 2, no. 5, pp. 25–31, 2004.

[22] M. Dell'Amico and M. Filippone, "Monte carlo strength evaluation: Fast and reliable password checking," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 158–169.

[23] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring {Real-World} accuracies and biases in modeling password guessability," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 463–481.

[24] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzypsm: A new password strength meter using fuzzy probabilistic context-free grammars," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 595–606.

[25] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur, "Reasoning analytically about password-cracking software," in *2019 IEEE SYMPOSIUM ON SECURITY AND PRIVACY (SP 2019)*, ser. IEEE Symposium on Security and Privacy. IEEE; IEEE Comp Soc; CS Financial, 2019, pp. 380–397.

[26] A. M. Di Campi, R. Focardi, and F. L. Luccio, "The revenge of password crackers: Automated training of password cracking tools," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 317–336.

[27] J. Eckroth, L. Hough, and H. ElAarag, "Oneruletofindthem: Efficient automated generation of password cracking rules," *Journal of Computing Sciences in Colleges*, vol. 39, no. 3, pp. 226–248, 2023.

[28] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 965–982.

[29] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane, "Omen: Faster password guessing using an ordered markov enumerator," in *Engineering Secure Software and Systems: 7th International Symposium, ESSoS 2015, Milan, Italy, March 4-6, 2015. Proceedings 7*. Springer, 2015, pp. 119–132.

[30] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1242–1254.

[31] X. Guo, Y. Liu, K. Tan, W. Mao, M. Jin, and H. Lu, "Dynamic markov model: Password guessing using probability adjustment method," *Applied Sciences*, vol. 11, no. 10, p. 4607, 2021.

[32] H. Cheng, W. Li, P. Wang, and K. Liang, "Improved probabilistic context-free grammars for passwords using word extraction," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 2690–2694.

[33] H.-C. Chou, H.-C. Lee, H.-J. Yu, F.-P. Lai, K.-H. Huang, C.-W. Hsueh *et al.*, "Password cracking based on learned patterns from disclosed passwords," *IJICIC*, vol. 9, no. 2, pp. 821–839, 2013.

[34] R. Veras, C. Collins, and J. Thorpe, "A large-scale analysis of the semantic password model and linguistic patterns in passwords," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 3, pp. 1–21, 2021.

[35] S. Houshmand, S. Aggarwal, and R. Flood, "Next gen pcfg password cracking," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1776–1791, 2015.

[36] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 07 2019. [Online]. Available: https://doi.org/10.1162/neco_a_01199

[37] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[38] Z. Xia, P. Yi, Y. Liu, B. Jiang, W. Wang, and T. Zhu, "Genpass: A multi-source deep learning model for password guessing," *IEEE Transactions on Multimedia*, vol. 22, no. 5, pp. 1323–1332, 2019.

[39] M. Zhang, Q. Zhang, X. Hu, and W. Liu, "A password cracking method based on structure partition and bilstm recurrent neural network," in *Proceedings of the 8th International Conference on Communication and Network Security*, 2018, pp. 79–83.

[40] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *neural information processing systems*, 2014.

[41] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 417–434.

[42] L. Xu, C. Ge, W. Qiu, Z. Huang, Z. Gong, J. Guo, and H. Lian, "Password guessing based on lstm recurrent neural networks," in *Computational Science and Engineering*, 2017.

[43] Y. Fang, K. Liu, F. Jing, and Z. Zuo, *Password Guessing Based on Semantic Analysis and Neural Networks: 12th Chinese Conference, CTCIS 2018, Wuhan, China, October 18, 2018, Revised Selected Papers*. Trusted Computing and Information Security, 2019.

[44] J. Luo, J. Deng, C. Lu, and H. Liu, "Recurrent neural network based password generation for group attribute context-ware applications," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019.

[45] Y. Zhang, H. Xian, and A. Yu, "Csnn: Password guessing method based on chinese syllables and neural network," *Peer-to-peer Networking and Applications*, vol. 13, no. 6, pp. 2237.0–2250.0, 2020.

[46] J. Zhang, C. Yang, Y. Zheng, W. You, R. Su, and J. Ma, "A preliminary analysis of password guessing algorithm," in *International Conference on Computer Communications and Networks*, 2020, pp. 1–9.

[47] X. Guo, Y. Liu, K. Tan, M. Jin, and H. Lu, "Pggan: Improve password cover rate using the controller," in *Journal of Physics: Conference Series*, vol. 1856, no. 1. IOP Publishing, 2021, p. 012012.

[48] D. Huang, Y. Wang, and W. Chen, "Rlpassgan: Password guessing model based on gan with policy gradient," in *International Conference on Security and Privacy in New Computing Environments*. Springer, 2021, pp. 159–174.

[49] S. Nam, S. Jeon, and J. Moon, "Generating optimized guessing candidates toward better password cracking from multi-dictionaries using relativistic gan," *APPLIED SCIENCES-BASEL*, vol. 10, no. 20, 2020.

[50] T. Zhou, H.-T. Wu, H. Lu, P. Xu, and Y.-M. Cheung, "Password guessing based on gan with gumbel-softmax," *Security and Communication Networks*, vol. 2022, 2022.

[51] Y. Liu, Z. Xia, P. Yi, Y. Yao, T. Xie, W. Wang, and T. Zhu, "Genpass: A general deep learning model for password guessing with pcfg rules and adversarial generation," in *Intelligent Cloud Computing*, 2018.

[52] T. Yang and D. Wang, "Rankguess: Password guessing using adversarial ranking," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 40–40.

[53] K. Xiu and D. Wang, "{PointerGuess}: Targeted password guessing model using pointer mechanism," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5555–5572.

[54] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[55] J. Rando, F. Perez-Cruz, and B. Hitaj, "Passgpt: Password modeling and (guided) generation with large language models," in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 164–183.

[56] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[57] X. Su, X. Zhu, Y. Li, Y. Li, C. Chen, and P. Esteves-Veríssimo, "Pagpassgpt: Pattern guided password guessing via generative pretrained transformer," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2024, pp. 429–442.

[58] Z. Xie, F. Shi, M. Zhang, H. Ma, H. Wang, Z. Li, and Y. Zhang, "Guessfuse: hybrid password guessing with multi-view," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4215–4230, 2024.

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

[60] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[61] Z.-H. Zhou, M. Li *et al.*, "Semi-supervised regression with co-training." in *IJCAI*, vol. 5, 2005, pp. 908–913.

[62] Y. Wang and T. Li, "Improving semi-supervised co-forest algorithm in evolving data streams," *Applied Intelligence*, vol. 48, no. 10, pp. 3248–3262, 2018.

[63] Z.-H. Zhou and M. Li, "Semi-supervised learning by disagreement," *Knowledge and Information Systems*, vol. 24, no. 3, pp. 415–439, 2010.

[64] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille, "Deep co-training for semi-supervised image recognition," in *Proceedings of the european conference on computer vision (eccv)*, 2018, pp. 135–152.

[65] G. Katz, C. Caragea, and A. Shabtai, "Vertical ensemble co-training for text classification," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 2, pp. 1–23, 2017.

[66] J. Bonneau, "The science of guessing: analyzing an anonymized corpus of 70 million passwords," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 538–552.

[67] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[68] J. He, R. Mao, Z. Shao, and F. Zhu, "Incremental learning in online scenario," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[69] K. Huang, P. Li, J. Ma, T. Yao, and Y. Liu, "Knowledge transfer in incremental learning for multilingual neural machine translation," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 15 286–15 304.

[70] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[71] B. Berger, M. S. Waterman, and Y. W. Yu, "Levenshtein distance, sequence comparison and biological database search," *IEEE transactions on information theory*, vol. 67, no. 6, pp. 3287–3294, 2020.

[72] E. Bonabeau, "Swarm intelligence: From natural to artificial systems," *Oxford University Press google schola*, vol. 2, pp. 25–34, 1999.

[73] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, "Particle swarm optimization: basic concepts, variants and applications in power systems," *IEEE Transactions on evolutionary computation*, vol. 12, no. 2, pp. 171–195, 2008.

[74] S. Li, X. Chi, and B. Yu, "An improved particle swarm optimization algorithm for the reliability–redundancy allocation problem with global reliability," *Reliability Engineering & System Safety*, vol. 225, p. 108604, 2022.

[75] Z. Parish, C. Cushing, S. Aggarwal, A. Salehi-Abari, and J. Thorpe, "Password guessers under a microscope: an in-depth analysis to inform deployments," *International Journal of Information Security*, vol. 21, no. 2, pp. 409–425, 2022.

[76] D. Wang, Y. Zou, Y. Tao, and B. Wang, "Password guessing based on recurrent neural networks and generative adversarial networks," *Chin. J. Comput*, pp. 1519–1534, 2021.

[77] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han, "Chunk-level password guessing: Towards modeling refined password composition representations," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 5–20.