# PassDiff: A New Approach for Password Guessing Using Diffusion Model

Sheng Guo[1], Ming Duan[1,2(✉)], Yibin Du[1], Wei Wang[1], and Lulu Guo[1]

[1] College of Cyberspace Security, Information Engineering University, Zhengzhou 450001, China
mdscience@sina.com

[2] Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450001, China

**Abstract.** Password guessing models can be broadly divided into three classes: dictionary-based password guessing model, password guessing model based on probability statistics and password guessing model based on deep learning. Recurrent neural networks and generative adversarial networks are the main deep learning techniques used for password guessing in the past. In this paper, we propose a novel PassDiff method for password guessing using denoising diffusion probabilistic models (DDPMs). Considering the similarity between the password space and the text space, we incorporate a byte-level tokenizer in the input phase and optimize the sampling process by modifying the source code. We encode a special character, and it makes PassDiff can handle input of variable length and obtain variable output without manual truncation. The experimental results show that PassDiff produces high-quality passwords even with minimal denoising steps. We recommend setting the denoising steps to 5–50, which can increase the sampling speed by tens of times. Compared with PassGAN, the training process of PassDiff is more stable and the cracking rate is also significantly improved. Specifically, when the denoising steps is set to 10 and $10^8$ passwords are generated, PassDiff increases the cracking rate by 3.17%, 6.33% and 13.22% on 12306, CSDN and RockYou datasets, respectively.

**Keywords:** Denoising diffusion probabilistic models · Passwords · Deep learning

## 1 Introduction

Password authentication, as a security mechanism to protect user privacy, has become the most popular authentication method for a long time because of its simplicity, practicality and efficiency. At the same time, attackers and researchers tried to apply various emerging technologies to the password guessing model. This promoted the development of cryptanalytic techniques.

Password guessing models can be broadly divided into three classes: dictionary-based password guessing model, password guessing model based on probability statistics and password guessing model based on deep learning.


Y. Zhang et al. (Eds.): CENet 2023, LNEE 1125, pp. 29–40, 2024.
https://doi.org/10.1007/978-981-99-9239-3_3

The dictionary-based password guessing model usually converts the original password of the dictionary through predefined rules to generate a new extended set, that is, the guessing set. The quanty of the guessing set depends on the amount of the original dictionary and the number of rules. It is characterized by simplicity and speed, but the generation of rules depends on personal experience. HashCat [1] and John the Ripper (JTR) [2] both efficiently implement dictionary-based approaches.

The Markov model [3] and the Probabilistic Context Free Grammars (PCFG) model [4] are two classical representatives of the password guessing model based on probability statistics. The Markov model uses the statistical laws of the dictionary to calculate the probability of the next character according to the previous character or context, output to the guessing set in probability order, and guess as many passwords as possible with least small guessing set. The PCFG model firstly preprocesses the password structure, and divides the character types of the password into numbers D, letters L and special characters S, then it counts consecutive segments. For example, the password "abc@123456" will be recorded as L3S1D6. The PCFG model usually statistics the frequencies of all structures and strings, retains the high probability structures and then fills them with high probability strings, finally outputs the filled passwords as the guessing set.

In recent years, deep learning technology has made great progress, and password guessing models based on deep learning arise at the historic moment. Deep learning techniques such as Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), Variational Auto-Encoders (VAEs) have also been applied to password guessing models. In addition, many scholars have done a lot of research on the combination of deep learning and traditional password guessing techniques such as PCFG.

In 2016, Melicher et al. [5] proposed the FLA model. They firstly used RNN to extract and predict password features. Liu et al. [6] presented a PL model based on PCFG and Long Short-Term Memory network (LSTM). Wang et al. [7] constructed PR and PR+ models based on PCFG and RNN. In 2019, Hitaj et al. [8] proposed PassGAN, the first author to apply the generative adversarial network to the password guessing model. Nam et al. [8] proposed rPassGAN, which applied RNN to PassGAN and adopted a dual discriminator structure. Fu et al. [9] modified the generative and discriminant networks of PassGAN to DenseNet and named their model DenseGAN.

Unlike probability-based or rule-based password guessing models, models built on deep learning make no assumptions about the password structure. The guessing set generated by a deep learning method is not restricted to any specific subset of the password space. Inversely, neural networks can detect extensive password information beyond the power of traditional password guessing models.

Recently, diffusion model [10] has become one of the most popular generative models due to its powerful generative ability. In 2020, Ho et al. [11] published Denoising Diffusion Probabilistic Models (DDPMs), the first paper to give a rigorous mathematical derivation to prove that the diffusion model can produce high-quality images. Song et al. [12] accelerated the sampling and proposed DDIMs, which was a more efficient probabilistic model and had a same training process with DDPM. Peebles et al. [13] proposed Diffusion Transformers (DiTs), which replaced the commonly-used U-Net with Transformer. In addition to applications in computer vision, speech generation and

natural language processing, more applications of diffusion model are being explored by researchers [14].

### 1.1 Our Contribution

As far as we know, it is the first time to apply the diffusion model to the password guessing model. When generating $10^8$ passwords, our model can increase the cracking rate by 13.22% on RockYou dataset. Through a lot of experiments, we prove the great potential of the diffusion model in the field of password guessing. Our works expand the application scope of the diffusion model.

We innovatively use the byte-level tokenizer embedding for encoding the password into the input vector of the diffusion model. By modifying the model interface, the passwords can be better input to the diffusion model.

By adjusting the parameters and modifying the source code, we optimize the denoising process to better fit the characteristics of the passwords. Additionally, we improve the sampling speed of the model.

### 1.2 Organization

The rest of our paper is organized as follows: We briefly introduce the basic knowledge of generative adversarial network and diffusion model in Chap. 2. In this chapter, we present our model: PassDiff. The fourth chapter is the description of our experimental settings. In Chap. 5, we introduce and discuss our experimental results. In Chap. 6, we draw the experimental conclusions and prospect our future work.

## 2 Background and Related Works

### 2.1 Generative Adversarial Networks

Generative adversarial networks (GANs) are made up of two parts: a generator (G) and a discriminator (D), both of which are constructed using neural networks. G usually takes random features or noise as its input. In the training process, G learns the distribution of the input data and makes its output gradually approach the distribution of the input. D estimates the conditional probability of the examples given a set of tagged inputs. Throughout the training process, D spares no effort to distinguish the real samples and fake ones. After hundreds of thousands of such games, G and D eventually reach an equilibrium, with G generating a sample that is consistent with the distribution of input data, and D having no advantage in guessing the source of the samples. The process can be express by the formula as follows:

$$\min_G \max_D \left( \sum_{i=1}^{n} \log D(x_i) + \sum_{j=1}^{n} \log\left(1 - D\left(G\left(z_j\right)\right)\right) \right), \tag{1}$$

where G and D represent generator and discriminator, respectively.

### 2.2 Denoising Diffusion Probabilistic Models

The Denoising Diffusion Probabilistic Models (DDPMs) involve two Markov chains: a forward chain (diffusion process) for adding noise to the input data, and a backward chain (denoising process) for transforming the noise back into the original data. The diffusion process is often manually designed to convert any original distribution into a simple prior distribution such as standard Normal distribution, while the denoising process reverses this process using a parameterized transition kernel learned from a deep neural network. The denoising process continuously generate new data points in two steps: first sampling a random data from the prior distribution and then performing the original sampling.

Mathematically, given a data distribution $x_0 \sim q(x_0)$ and assuming that the transition kernel of the forward chain is $q(x_t|x_{t-1})$, the diffusion process will generate a sequence of random variables $x_1, x_2, \ldots, x_T$. We use $q(x_1, \ldots, x_T|x_0)$ to denote the joint distribution of $x_1, x_2, \ldots, x_T$ conditioned on $x_0$. According to the chain rule of conditional probability, it can be decomposed into the following formula:

$$q(x_1, \ldots, x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{2}$$

In DDPM, the transition kernel $q(x_t|x_{t-1})$ is usually manual designed to gradually convert the data distribution $q(x_0)$ to a identifiable prior distribution. Gaussian perturbation is often used as the transition kernel, and the most typical design of transition kernel $q(x_t|x_{t-1})$ is

$$x_t \sim N\left(\sqrt{1-\beta_t}x_{t-1}, \beta_t\right), \tag{3}$$

Among them, $\beta_t \in (0, 1)$ is a hyperparameter selected before training. We use this kind of transition kernel here to simplify our discussion.In addition, other types of transition kernels apply equally well. As we choose Gaussian perturbation as transition kernel, we can get expressions for the $q(x_t|x_0)$ of all $t \in \{0, 1, \ldots, T\}$ by marginalizing the joint distribution in the equality (2). Specifically, let $\alpha_t = 1 - \beta_t$, $\overline{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, we have

$$x_t \sim N\left(\sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha}_t)\right). \tag{4}$$

Given $x_0$, by choosing a random vector, $\varepsilon \sim N(0, 1)$, and transforming it by the following formula

$$x_t = \sqrt{\overline{\alpha}_t}x_0 + (1-\overline{\alpha}_t)\varepsilon, \tag{5}$$

we can easily obtain a sample of the $x_t$. When $\overline{\alpha}_T \approx 0$, the distribution of $x_T$ is almost Gaussian, so we have

$$q(x_T) = \int q(x_T|x_0)q(x_0)dx_0 \approx x_T \sim N(0, 1). \tag{6}$$

The diffusion process consistently destroies the input data with noise until it loses all the features. To produce new data points, DDPM firstly samples a random noise point

that follows the prior distribution, and then conducts a regular reverse Markov chain to increasingly eliminates the noise. Concretely, we use a prior distribution $p(x_T)$ (where $x_T \sim N(0, 1)$) and a knowable transition kernel $p_\theta(x_{t-1}|x_t)$ to parameterize the reverse Markov chain. The construction of the diffusion process makes the $q(x_T)$. Approximately satisfy $x_T \sim N(0, 1)$. The knowable transition kernel $p_\theta(x_{t-1}|x_t)$ can be calculated by the following formula

$$x_{t-1} \sim N(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), \tag{7}$$

where $\theta$ represents the model parameters, the mean $\mu_\theta(x_t, t)$ and the variance $\Sigma_\theta(x_t, t)$ are learned from deep neural networks, a very classical implementation is to learn noise using U-Net. Using the reverse Markov chain, we can firstly sample a noise point $x_T \sim p(x_T)$ and then iteratively calculate it from the knowable transition probability $p_\theta(x_{t-1}|x_t)$ until $t = 1$, and finally we will get a data point $x_0$.

Training the reversearkov chain to correspond to the forward Markov chain is very important to the denoising process. So, we need to select a proper parameter $\theta$ so that the reverse process joint distribution $p_\theta(x_0, x_1, \ldots, x_T) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$ is very close to the joint distribution $q(x_0, x_1, \ldots, x_T) = q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1})$ of the forward process (Fig. 1).

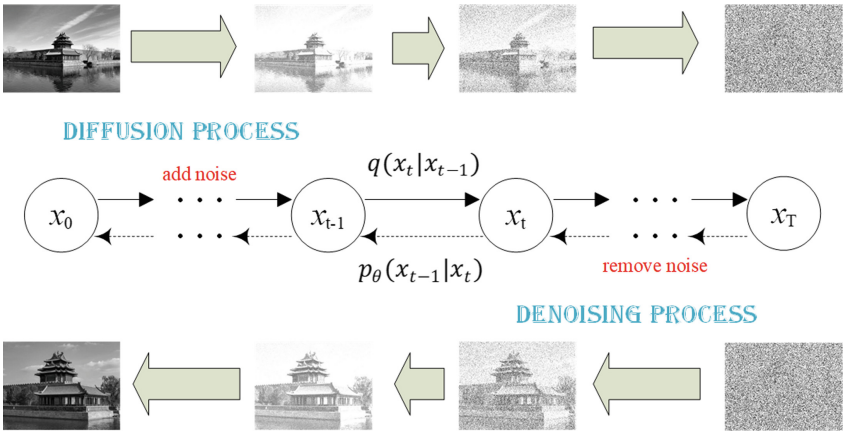


**Fig. 1.** Diffusion model architecture

## 3 Proposed Model: PassDiff

The standard diffusion model is mainly applied in the field of image generation, and both its input and output are 2D vectors. Li et al. [15] made some modifications on the standard diffusion model. They used Transformer to learn about noise distribution and added the text embedding step and rounding step. They named their model the Diffusion-LM model, which applies the diffusion model to the domain of controllable

text generation. Aman [16] simplified the source code of Diffusion-LM model to make it easier to use. The diffusion-LM model takes text as input and encodes the text into a 1D integral vector using word-level tokenizer embedding. In the diffusion process, the integral vector will be converted to a 1D floating vector. At the end of the denoising process, the rounding step converts a 1D floating vector back to a 1D integer vector and outputs semantically similar sentence text via the tokenizer. Diffusion-LM model, however, can only output fixed-length text, which needs to be truncated manually.

Word-level tokenizer embedding treats text as a sequence of words, typically divided by spaces or punctuation marks. Encoding each word or word with more than a certain frequency can preserve the semantic connection between words to a certain extent, but it is easy to cause the encoding range being too large.

Byte-level tokenizer embedding treats text as a sequence of bytes, encoding both a single byte and multiple bytes in a row. Single-byte encoding has poorer semantic connections, and multi-bytes encoding is a compromise between single-byte encoding and word-level encoding.

Considering the similarity between password space and text space, the semantic relationship between the preceding and following characters of the password is weaker than that in the corpus. Referring to the Diffusion-LM model, we encode the input password with character-level tokenizer, and propose a PassDiff password prediction model. In addition, we analyzed the code of the denoising process and found that it could only generate passwords of fixed length. We introduce a random factor so that users can generate passwords of any length within the required range according to their own needs, which makes the generated dictionary more close to the distribution characteristics of the training set and improves the password cracking rate to some extent.

We found no significant advantage of multi-bytes encoding over single-byte encoding. For simplicity, this article encodes only a single byte, i.e., only 1 special byte and 95 visible bytes. Our coding range is 0 to 95. The special byte is represented as 0, and the remaining visible bytes are represented in ASCII order from 1 to 95. The encoding of the special character can handle input of variable length and obtain variable output without manual truncation. We found that the PassDiff model produces high-quality passwords even with minimal denoising steps. We recommend setting the denoising steps to 5 to 50, which can increase the sampling speed by tens of times (Fig. 2).

## 4 Experiment Setup

Our experimental environment is a DELL Precision 5820 workstation with 128G memory, 10 CPU cores, and an NVIDIA GeForce RTX 3090 graphics card. All of our experiments were performed under Win10 and Ubuntu 22.04. We conducted PassGAN related experiments on Win10, using the source code of rnnPassGAN [8, 17]. Experiments related to diffusion model were carried out on Ubuntu 22.04, and the source code came from minimal-text-diffusion [16].

### 4.1 Dataset

We chose three classical datasets for the experiment: 12306, CSDN and RockYou. We start by suming up the amount of passwords in the three datasets. In the second step,

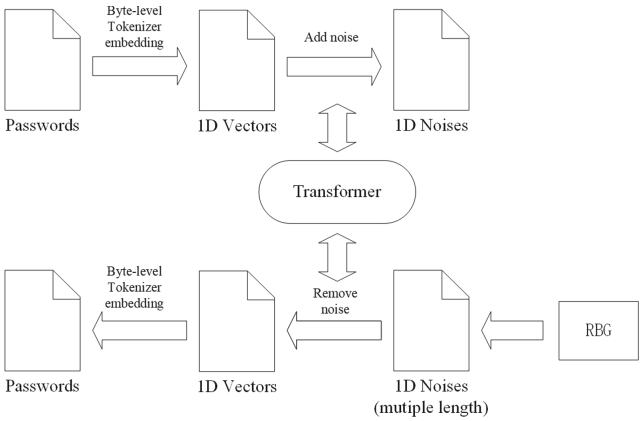


**Fig. 2.** PassDiff model architecture

we deleted passwords that contain invisible characters or are longer than 15 and marked them as Totals1. The number of unique passwords and the repetition rate were also recorded. We continued to divide the passwords in Totals1 into the train set and the test set at random ratios of 80% and 20%, respectively, and noted the count of unique passwords. The statistical results are as follows.

**Table 1.** Totals, unique totals and repetitive rate of 12306, CSDN and RockYou

| Datasets | Totals | Totals1 (Uniques) | Repetition rate (%) | Train set (Uniques) | Test set (Uniques) |
|---|---|---|---|---|---|
| 12306 | 131,653 | 131,652 (117,807) | 10.5 | 105,321 (95,283) | 26,331 (24,974) |
| CSDN | 6,428,632 | 6358,467 (3973,939) | 37.5 | 5086,773 (3256,919) | 1271,694 (910,100) |
| RockYou | 32,584,847 | 32,330,944 (12,350,913) | 61.8 | 25,864,755 (10,562,791) | 6466,189 (3498,440) |

We counted the ratios of various length interval of the three datasets as follows.

It can be seen from Tables 1 and 2, 12306 dataset is small in scale with low repetition rate, while RockYou dataset has a large size with high repetition rate. The length intervals of the three datasets are all relatively concentrated.

### 4.2 Hyperparameters

The main hyperparameters of our PassDiff model include: training rounds, batch size, diffusion steps, denoising steps, input length, output length, learning rate and so on. At present, we set some parameters to the default values of the original diffusion

**Table 2.** The ratio of length intervals of 12306, CSDN and RockYou

| Datasets | Totals1 | Ratio of various length intervals | | | | The primary length interval (ratio) |
|---|---|---|---|---|---|---|
| | | 1–5 (%) | 6–10 (%) | 11–15 (%) | >15 (%) | |
| 12306 | 131,652 | 0 | 94.11 | 5.89 | 0 | 6–10 (94.11%) |
| CSDN | 6,358,467 | 0.62 | 76.56 | 21.72 | 1.1 | 8–12 (90.54%) |
| RockYou | 32,330,944 | 4.32 | 86.54 | 8.36 | 0.78 | 5–11 (94.18%) |

model. Moreover, we fixed the diffusion steps and the learning rate at 2000 and 0.0001, respectively.

Using the PassDiff model, we generate $10^5$ to $10^8$ unique passwords, and calculated the cracking rate between the generated dictionary and the test set. Moreover, we compare our cracking rate with PassGAN's dictionary of the same size. When calculating the cracking rate, the passwords in the generated dictionary are unique ones, while the train and the test sets contain duplicate passwords.

## 5 Evaluation

### 5.1 Experiments on the PassDiff Model

For the 12306 dataset, we conducted a lot of experiments of PassDiff model with different training rounds, batch sizes and denoising steps. Due to the slow sampling speed of the diffusion model, we mainly generated dictionaries of size of $10^5$ and $10^6$ for comparison.

The two dot plots below show how the cracking rate of the 12306 dataset varies with the number of denoising steps. The quantity of training rounds in both experiments was 200,000, and both generated $10^5$ and $10^6$ passwords, but their batch sizes were 1024 and 256, respectively (Figs. 3 and 4).

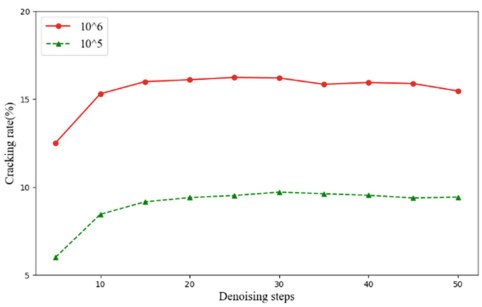


**Fig. 3.** Cracking rate of 12306 with batch size = 1024

The above plots show that the cracking rate is positively correlated with the batch size and the amount of the generated dictionary. As the number of denoising steps raises,

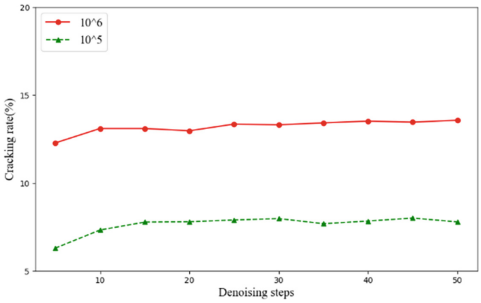


**Fig. 4.** Cracking rate of 12306 with batch size = 256

the cracking rate gradually increases and stabilizes around 20–25 steps, indicating that the PassDiff model requires only a small number of restore steps to produce a good dictionary.

Figure 5 illustrates the variation pattern of cracking rate with training rounds for the 12306 dataset. In this experiment, the batch size is 1024 and the number of denoising steps is 20. Respectively, we generated $10^5$ and $10^6$ passwords.

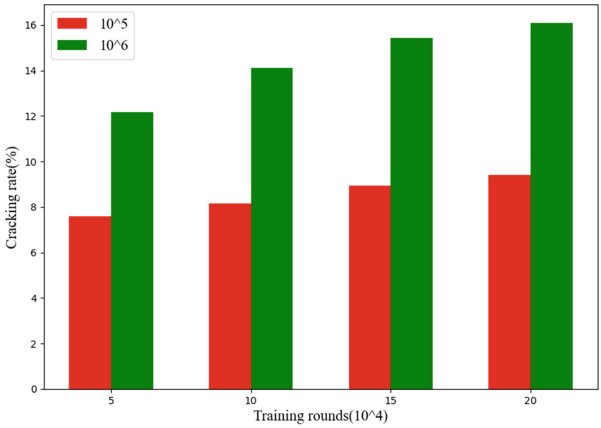


**Fig. 5.** Cracking rate of 12306 varies with the training rounds

As shown in the above graph that the cracking rate gradually increases with the increase of training rounds. Through other experiments, we found that the cracking rate had not reached the peak when training 200,000 rounds. If we increase the training rounds, we will acquire a higher cracking rate.

For CSDN and RockYou datasets, we also conducted experiments to generate dictionaries of different sizes under the conditions of different training rounds, different batch sizes and different denoising steps. The datasets showed roughly the same rule as 12306. Here, we present only our representative experiments. Figures 6 and 7 show the effect of the number of denoising steps on the cracking rate of CSDN and RockYou when

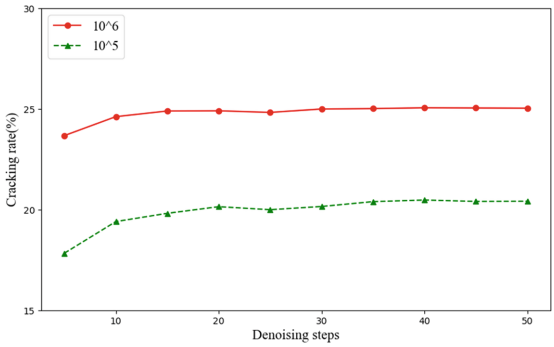


**Fig. 6.** Cracking rate of CSDN varies with the denoising steps

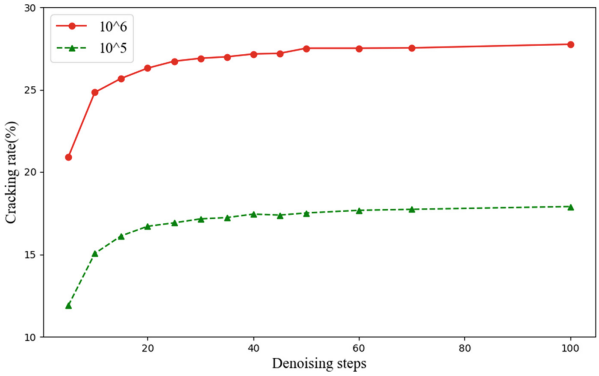


**Fig. 7.** Cracking rate of RockYou varies with the denoising steps

generating $10^5$ and $10^6$ passwords, respectively, under the condition that the quantity of training rounds is set to 200,000 and the batch size is 1024.

It is visible that the cracking rate of CSDN and RockYou is higher than 12306 on the whole. We think this is due to their repetition rate being higher than 12306. The cracking rates of CSDN and RockYou datasets gradually stabilized at about 40 denoising steps, exceeding the denoising steps required for 12306. Upon further analysis, we attributed it to inadequate training and larger data size.

From the experimental results of PassDiff model above, we analyze that increasing the quantity of both training rounds and batch size can improve the cracking rate. With the increase of the number of denoising steps, the crack first gradually rises to the maximum value and then tends to be stable.

Additionally, we explored the variation of repetition rate, training time and sampling time. We find that the repetition rate is positively correlated with the number of training rounds and denoising steps. The higher the repetition rate, the longer it takes to generate a unique password dictionary of the same size. The training time is proportional to the quantity of training rounds and batch size. Regardless of the repetition rate, the sampling time is proportional to the number of denoising steps and the amount of the generated

dictionary. Take the 12306 dataset as an example. When the batch size is 1024, it takes about 31 h to train 200,000 rounds. If the denoising step is set to 20, we will need about 11 h to generate $10^7$ unique passwords.

### 5.2 Comparison with PassGAN

We tested the cracking rate of PassGAN under the same conditions for three datasets: 12306, CSDN, and RockYou. In the paper [7], the authors show that for the RockYou dataset, the model gets the best dictionary at 199,000 rounds of training. This is consistent with our experiment. Therefore, we trained 200,000 rounds on each of the three datasets for comparison. For our PassDiff model, we sacrifice some cracking rates to generate a larger dictionary, and the parameters of our model are 200,000 training rounds, 1024 batch size, and 10 denoising steps. Our results are shown in Table 3.

**Table 3.** Comparison of cracking rate of PassGAN and PassDiff for 12306, CSDN and RockYou

| Dataset | Model | Size of dictionary (%) | | | |
|---|---|---|---|---|---|
| | | $10^5$ (%) | $10^6$ (%) | $10^7$ (%) | $10^8$ (%) |
| 12306 | PassGAN | 6.85 | 13.77 | 21.84 | 34.91 |
| | PassDiff | 8.45 | 15.30 | 24.70 | 38.08 |
| CSDN | PassGAN | 17.12 | 21.27 | 25.31 | 30.99 |
| | PassDiff | 19.41 | 24.61 | 30.03 | 37.32 |
| RockYou | PassGAN | 9.44 | 16.80 | 26.32 | 37.27 |
| | PassDiff | 15.45 | 24.83 | 37.36 | 50.49 |

From Table 3, we can clearly see that our PassDiff model has significant advantages over PassGAN. When generating $10^8$ passwords, the cracking rate of our model is 3.17%, 6.33% and 13.22% higher than that of PassGAN on 12306, CSDN and RockYou data sets, respectively. In addition, it is not our best performance in cracking rate. However, there is still some gap between our model and the traditional password guessing model (such as PCFG). We will keep improving our model and try to combine it with traditional methods.

## 6 Conclusion

In this paper, by optimizing the diffusion model parameters and increasing the number of training rounds, we propose PassDiff model with shorter training time and more stable training process than PassGAN. However, due to the slow sampling speed of diffusion model, it is difficult for the model used in this paper to generate billion-level dictionaries. The diffusion model has great potential in constructing password guessing models, and can also make great breakthroughs in theory.

In future work, we will further optimize parameters and debug other parameters such as diffusion steps and learning rate. We plan to use other neural networks to learn from the noise to improve the cracking rate. We will also try other encoding methods, such as encoding more than two characters and trying to combine them with PCFG.

## References


1. HashCat: https://hashcat.net. Last accessed 21 May 2023
2. John the Ripper: https://www.openwall.com/john/. Last accessed 15 Apr 2023
3. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 364–372. ACM (2005)
4. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 30th IEEE Symposium on Security and Privacy, pp. 391–405. IEEE (2009)
5. Liu, Y., Xia, Z., Yi, P., Yao, Y., Xie, T., Wang, W., Zhu, T.: Genpass: a general deep learning model for password guessing with PCFG rules and adversarial generation. In: Proceedings of the 2018 IEEE International Conference on Communications (ICC), pp. 1–6 (2018)
6. Wang, D., Zou, Y., Tao, Y., Wang, B.: Password guessing based on recurrent neural networks and generative adversarial networks. Chin. J. Comput., 1519–1534 (2021)
7. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: a deep learning approach for password guessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 217–237. Springer, Cham (2019)
8. Nam, S., Jeon, S., Moon, J.: Recurrent GANs password cracker for IoT password security enhancement. In: Proceedings of the International Workshop on Information Security Applications, pp. 247–258. Jeju Island, Korea (2019)
9. Fu, C., Duan, M., Dai, X., Wei, Q., Wu, Q., Zhou, R.. DenseGAN: A Password Guessing Model Based on DenseNet and PassGAN. In: Information Security Practice and Experience. ISPEC 2021. Lecture Notes in Computer Science, vol. 13107, pp. 296–305. Springer, Cham (2021)
10. Anderson, B.D.: Reverse-time diffusion equation models. Stochast. Processes Appl. **12**(3), 313–326 (1982)
11. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Adv. Neural. Inf. Process. Syst. **33**, 6840–6851 (2020)
12. Song J, Meng C, Ermon S.: Denoising Diffusion Implicit Models. arXiv preprint arXiv:2010.02502 (2020)
13. Peebles W, Xie S.: Scalable Diffusion Models with Transformers. arXiv preprint arXiv:2212.09748 (2022)
14. Yang L, Zhang Z, Song Y, et al.: Diffusion Models: A Comprehensive Survey of Methods and Applications. arXiv preprint arXiv:2209.00796 (2022)
15. Li, X., Thickstun, J., Gulrajani, I., et al.: Diffusion-lm improves controllable text generation. Adv. Neural. Inf. Process. Syst. **35**, 4328–4343 (2022)
16. Aman, M.: Minimal Text Diffusion. https://github.com/madaan/minimal-text-diffusion. Last accessed 10 Apr 2023
17. Peng, K.: GAN-Based Password Guessing. https://github.com/ponedo/rnnPassGAN-password-cracking. Last accessed 10 Apr 2023
18. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: USENIX Security Symposium, pp. 175–191 (2016)