**Password Diffusion**

**Introduction**

This project is an attempt to apply discrete diffusion models for password generation. The password cracking game has existed almost as long as passwords themselves and recent advancements in the generative algorithms can be applied to generate new password candidates that don’t appear on wordlists nor can be generated with rule-based augmentation.

**Previous work**

PassGAN(https://arxiv.org/abs/1709.00440) was the first major AI model trained to generate candidates and it was revolutionary it is able to generate billions of realistic password even with an OK speed. More recently a model called PassGPT seems to surpass PassGAN recovering more than 40% of wordlists using a billion samples, while also being more resistant to mode collapse.

**Password Diffusion**

One of the main problems with both of these models is mode collapse. In password cracking quantity is probably the most important and mode collapse is the greatest setback to get to a true AI password cracking this issue has to be resolved. Diffusion models are known to be very resilient to mode collapse, but they have their disadvantages.

1. Their ability to generate text is discovered recently and there is very little research that can help building such model
2. They tend to produce larger and slower models
3. They need very careful finetuning

**My idea**

Because character level attention doesn’t have a context to learn from (like a structured meaning of the combination of the words) my idea is that the gradual corruption of the diffusion model could help the attention capture patterns more effectively and it might be an interesting approach to create a character level embedding, but that is a topic for another discussion. Here the attention mechanism helps the diffusion model denoise better. On top of that there is length and time embeddings. Both embeddings are pretty straight forward the time embedding gives the model information where in the diffusion process is and the length embedding helps the model capture the natural length having in mind the denoising process.

**Implementation**

The model uses these embeddings

Token Embedding:

Each character (or token) in your password is first mapped to a vector representation. This embedding captures the semantic properties of the token—its “identity” as a letter, digit, or symbol—and serves as the basic building block for the model’s understanding of the password content.

Positional Embedding:

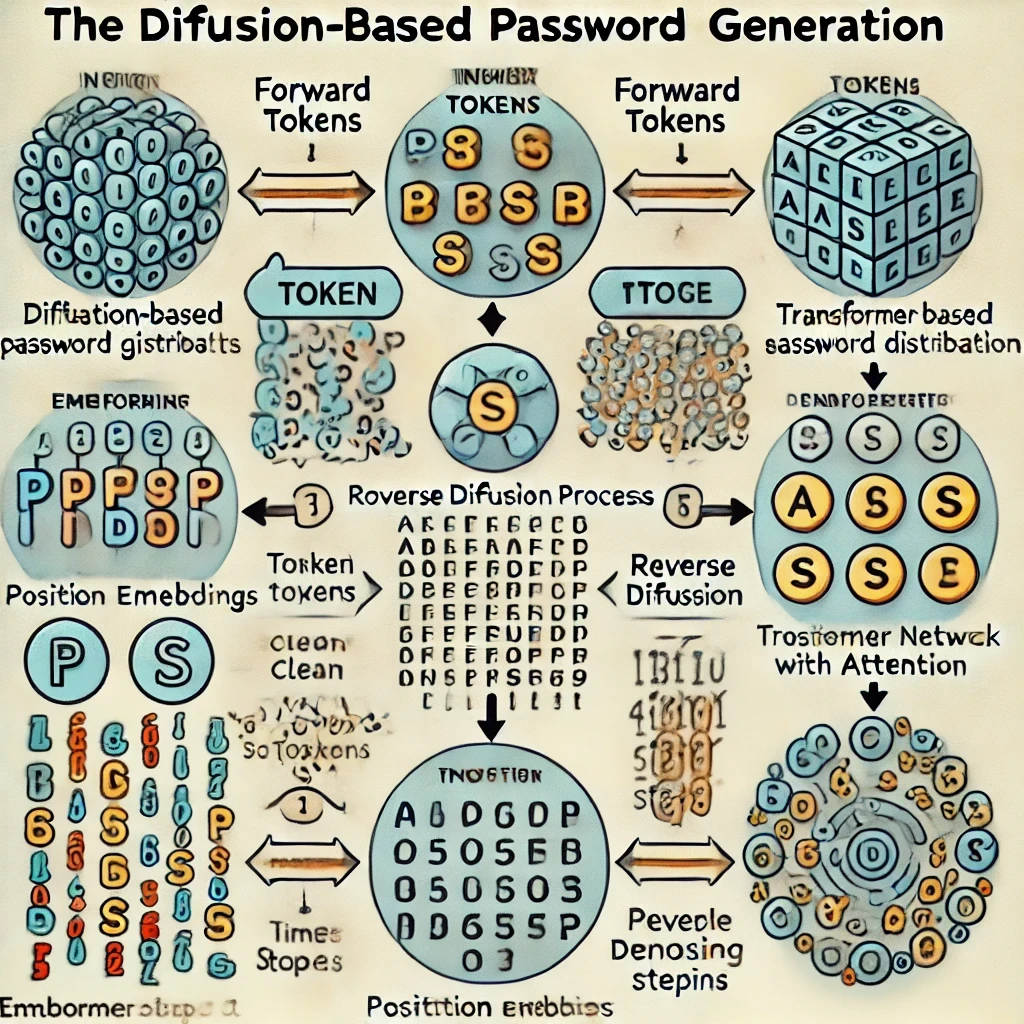
Since transformers have no inherent sense of sequence order, the positional embedding is added to the token embeddings to encode the position of each token within the sequence. This allows the model to know, for example, which token comes first or last, which is essential for generating coherent sequences.

Time Embedding:

In the diffusion process, the model must learn to denoise a sequence gradually. The time embedding provides information about the current diffusion timestep, so the network knows how “noisy” the input is and how much denoising is required. It effectively conditions the transformation on the progress of the reverse diffusion process.\

Ten denoising timesteps

****Loss function

Here I added penalties for numbers, repetition of characters and numbers at the teil of the password, because the model had problems with generalizing before these penalties. A noteworthy thing to say is that the loss function is calculated on the end output not for every diffusion step. This is done to improve coherency of the end passwords candidates. Here is an image of the whole process generated by a fellow diffusion model

Training data

The model is trained on the first 5 Million passwords of the famous list rockyou.txt for 5 epochs and for 1 epoch on the full rockyou.txt. I did it this way because the first 5 million passwords are more common and have better coherency and gave the model 1 additional epoch on the full rockyou.txt for better generalization.

**Results**

Unfortunately the model hasn’t matured enough to beat previously established models like PassGAN, but it proves that password diffusion generation is very resilient to mode collapse. With further training and refinement maybe the model could be able to outperform previous methods, but for now I have to admit it is heavily outperformed by PassGAN.

Here are some results.

Картина, която съдържа текст, екранна снимка, линия, Шрифт

Генерираното от ИИ съдържание може да е неправилно.

You can find the samples generated by the model here: <https://drive.google.com/file/d/1leQWEwNidniNvz7qOvsFohoQYnSnvbJQ/view?usp=sharing>

References

PassGAN: <https://arxiv.org/abs/1709.00440>

PassGPT: https://arxiv.org/abs/2306.01545