PassGPT: Password Modeling and (Guided) Generation with Large Language Models

Javier Rando[1[0000−0002−2723−7660]], Fernando Perez-Cruz[1,2[0000−0001−8996−5076]], and Briland Hitaj[3[0000−0001−5925−3027]]

[1] ETH Zürich, Turnerstrasse 1, 8092 Zürich
jrando@ethz.ch
[2] Swiss Data Science Center, Turnerstrasse 1, 8092 Zürich
fernando.perezcruz@sdsc.ethz.ch
[3] SRI International, New York, NY 10165 USA
briland.hitaj@sri.com

Abstract. Large language models (LLMs) successfully model natural language from vast amounts of text without the need for explicit supervision. In this paper, we investigate the efficacy of LLMs in modeling passwords. We present **PassGPT**, an LLM trained on password leaks for password generation. PassGPT outperforms existing methods based on generative adversarial networks (GAN) by guessing twice as many *previously unseen* passwords. Furthermore, we introduce the concept of *guided password generation*, where we leverage PassGPT sampling procedure to generate passwords matching arbitrary constraints, a feat lacking in current GAN-based strategies. Lastly, we conduct an in-depth analysis of the entropy and probability distribution that PassGPT defines over passwords and discuss their use in enhancing existing password strength estimators.

Keywords: Password Guessing · LLMs · Generative AI

1 Introduction

Passwords still retain their status as the authentication mechanism of choice despite the ever-increasing number of alternative technologies [46,37], primarily thanks to passwords being easy to deploy and remember. Furthermore, most applications rely on passwords as a fallback mechanism when other methods do not succeed. Considering their prevalence, password leaks [22,49] are one of the main threats institutions (and individuals) face. Not only do password leaks enable adversaries to break into systems but they also make it possible to research and identify hidden patterns within human-generated passwords that guide creation and refinement of effective password cracking tools [1,31].

Machine learning (ML) has played (and continues to play) a prominent role in extracting and learning meaningful features from vast password leaks,

Code and models can be accessed at https://github.com/javirandor/passgpt

resulting in major contributions primarily towards two main areas of research: (1) password guessing [26,25,32,35,34] and (2) password strength estimation mechanisms [44,26,10,19,20].

At the same time, Large Language Models (LLMs), a family of ML models, has demonstrated tremendous effectiveness in natural language processing (NLP) and understanding (NLU). These models are based on the *Transformer* architecture; well-known examples include the Generative Pre-trained Transformer (GPT) models [9,29], PaLM [13] or LLaMA [43]. Given their recent success, we pose the following question: **How effectively can *LLMs* capture the underlying characteristics and cues hidden in the complex nature of human-generated passwords?**

To answer this question, we present and thoroughly evaluate an LLM-based password-guessing model called **PassGPT**. Suitable for both password guessing and password strength estimation, PassGPT is an *offline* password-guessing model based on the GPT-2 architecture [38]. When compared with prior work on deep generative models [25,34], PassGPT guesses 20% more *unseen passwords*, and demonstrates good generalization capabilities to novel leaks. Moreover, we enhance PassGPT with vector quantization [55]. The resulting architecture is **PassVQT**, which can increase the perplexity of generated passwords.

Unlike previous deep generative models that generate passwords as a whole, PassGPT sequentially samples each character,thus introducing the novel task of *guided password generation*. This method ensures a more granular (character-level) guided exploration of the search space, where generated passwords are sampled based on arbitrary constraints. Finally, PassGPT, in constrast with GANs, provides an explicit representation of the probability distribution over passwords. We show that password probabilities align with state-of-the-art password strength estimators: PassGPT assigns lower probabilities to stronger passwords. We also look for passwords that can be easily guessed by generative approaches, even though they are considered "strong" by strength estimators. We discuss how password probabilities under PassGPT can be valuable for enhancing existing strength estimators.

1.1 Contributions and Remarks

Given the nature of LLMs, as well as the probabilistic nature of DL-based password generation, we position ourselves in **offline password guessing**. Such a setup is in line with prior work in the domain [25,36,35,32,17]. In these scenarios, the adversary is in possession of one or more password hashes obtained from a target system and their primary goal is to obtain the plaintext version corresponding to the respective password/s hash [7]. In general, a powerful adversary can employ a series of heuristics [3,4,2,30,47,17] using a combination of tools [1,31] While doing so, the adversary seeks to avoid a worst-case scenario in which they would need to enumerate all potential guesses, i.e., brute-force.

In this work, we narrow down our experiments and comparisons to *deep generative models* for offline password guessing.[4] We acknowledge the field is broader than that (e.g., Markov models and online guessing), but we restrict ourselves to comparable architectures. Our goal is to provide additional tools in the password-guessing landscape rather than establishing a default go-to architecture. We summarize our contributions as follows:

- We introduce PassGPT, an autoregressive transformer that obtains state-of-the-art results in password generation and generalization to unseen datasets.
- We show how PassGPT enables a novel approach to password generation under arbitrary constraints: *guided password generation*.
- We examine password probabilities under PassGPT and how they align with strength. We discuss how this metric could be used to improve current strength estimators.
- We present PassVQT, a similar architecture enhanced with vector quantization to increase generation perplexity.

2 Background and Related Work

In this paper, we make heavy use of LLMs and generative AI. In this section, we introduce the concepts relevant to generative models (Section 2.1) and *transformer* models (Section 2.2). We conclude discussing progress in password guessing and strength estimation, focusing on works that use deep generative models (Section 2.3).

2.1 Deep Generative Models

Deep generative models are a class of deep learning (DL) techniques designed to *autonomously* grasp the characteristics underlying a set of samples from a distribution, i.e., training set, and to generate new samples from that distribution [21,42]. The primary distinction between the two major categories of generative models is how they represent probability distributions. Generative models can be either: (1) implicit or (2) explicit. *Implicit* models do not estimate the distribution of the training data directly; rather they learn a function that generates samples similar to the ground truth. The most notable example is Generative Adversarial Networks (GANs) [21]. *Explicit* models, on the other hand, explicitly model the underlying distribution of the training data, that can be later accessed [41]. Our models fall under this second category.

Generative Adversarial Networks [21]. GANs consist of two main components: (1) a generator $G(\mathbf{z}; \theta_g) : \mathbb{R}^n \to \mathbb{R}^n$, a neural network that takes in random noise from a prior p_z and generates samples resembling the training data

[4] Our experiments are limited in scale by GPU access. All experiments are done on a single consumer GPU. Scaling is a crucial factor to improve performance and training on larger leaks.

and (2) a discriminator $D(\mathbf{x}; \theta_d) : \mathbb{R}^n \to [0, 1]$, also a neural network, trained to distinguish between training samples and outputs from the generator.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(x)}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(z)}[\log(1 - D(G(\mathbf{z})))] \quad (1)$$

Both G and D are trained adversarially in a zero-sum game until the generator produces samples that are indistinguishable from real ones (see Equation 1). GANs can approximate sharp distributions and generate high-quality samples without defining a likelihood function.

Autoregressive Generative Models (AGMs). These explicit generative models make it possible to sample from the target distribution. They do so by specifying a probability density function over the data and decomposing it into the product of conditionals via the chain rule of probability. These conditionals can be parametrized using neural networks with parameters θ that take as input the preceding entries in the sequence (see Equation 2). This explicit definition makes it possible to train these models using *maximum likelihood estimation* unlike implicit generative models. Our models match this definition.

$$p(\mathbf{x}) \approx p(\mathbf{x}; \theta) = \prod_{i=0}^{n} p(x_i | x_0, \dots x_{i-1}; \theta) = \prod_{i=0}^{n} p(x_i | x_{<i}; \theta). \quad (2)$$

2.2 Transformers

Choosing the right neural network to model conditional probabilities in deep autoregressive models has received a lot of attention in recent research [8]. Two commonly used architectures are *recurrent neural networks* (RNNs) [40] and *transformers* [45]. Transformers are the most successful because of faster, more stable, and parallelizable training [45].

Transformers rely entirely on the attention mechanism [41] to model dependencies within the input sequence regardless of distance. The original transformer [45] consists of an encoder and a decoder, both made up of multiple layers of self-attention and feed-forward neural networks. The main difference between the encoder and the decoder lies in how they consume the input. The encoder uses all information in the input sequence to generate a latent representation for each token, while the decoder can only use information from previous tokens. Recent work has proposed using only the decoder for autoregressive language modeling, where words are generated conditioned only on previous ones. GPT models [38,9] have revolutionized NLP by relying solely on transformer decoders.

2.3 Related Work

This section focuses primarily on the use of deep generative models for *password guessing* and *password strength estimation*.

Password Guessing is a widely studied class of attacks [27,18], where the adversary either has a limited number of guesses (*online password guessing*) or is already in possession of a copy of the password hashes and needs to break them

(*offline password guessing*). In both these scenarios, the adversary seeks to crack passwords before they run out of budget, i.e., the number of tries in an online service, or computing resources available for offline guessing.

The research community has explored different approaches to guessing passwords efficiently.

Tools like Hashcat [1] or John the Ripper [31] employ heuristics, such as mangling-rules, dictionary attacks, association attacks, hybrid attacks, and more [2,5,4,3,30]. Further work in the domain has proposed and evaluated the use of Markov models [28,17], probabilistic context-free grammars (PCFG) [47], (deep) neural networks [26,14,36,33], or composition of techniques [54]. Our work focuses on the use of generative deep neural networks.

Deep Generative Models for Password Guessing. To the best of our knowledge, PassGAN [25] is the first work implementing generative models, in particular GANs, for password guessing. PassGAN uses the improved Wasserstein GAN (IWGAN) [23] to learn the underlying distribution of the RockYou password leak [52] and then evaluates the model performance on additional leaks like LinkedIn data [50]. Pasquini et al. [36] suggested an improved version of PassGAN by adding random noise to the input representation to improve training stability. Follow-up work has explored different architectures obtaining similar results. For instance, PassFlow employs normalizing flows instead of GANs [32].

Password Strength Estimation aims to define a password robustness metric against guessing [44,11]. Similarly to password guessing, this has resulted in a variety of different approaches such as Markov models [12,16], PCFGs [47], or neural networks [26]. Our work uses the *lightweight* estimator `zxcvbn` [48], as recommended by Carnavalet et al. [11].

3 Experimental Setup

In Section 2.3 we introduced the central problem we are exploring: password guessing. This section presents the datasets (Section 3.1) and novel architectures (Section 3.2) used throughout our experiments.

3.1 Datasets

We chose datasets previously utilized in password guessing work and security research[5] that enable comparison of our techniques. The diverse characteristics of these password sets enhance the robustness of our evaluation and demonstrate generalization capabilities. Table 1 summarizes the key information about each dataset. The largest leaks that we consider for training are RockYou and LinkedIn, as done by previous work [36,25,32].

We split the previous datasets into training and test sets using the same approach as PassGAN [25] and follow-up work [36]. For RockYou, we take the

[5] This work makes use of publicly available password datasets. We consider this practice to be ethical and consistent with prior security research, e.g., [25,36,26].

list of all passwords of at most 10 and 16 characters, respectively. In this leak, passwords may appear more than once. We take 80% of this list as training data. From the remaining 20%, we keep as test data all passwords that are not contained in the training split, keeping only passwords with low frequency. The most commonly used password in the test set appears only 7 times in the entire leak. The average frequency of test passwords is 1.03. In comparison, the most frequent password in RockYou –`123456`– appears 290,731 times, and the average frequency in the entire leak is 2.28. This method allows us to test our model's generation abilities on low-probability passwords that were not seen during training.

Since the LinkedIn leak does not provide information about password frequency, we take 80% as training data and the remaining 20% for evaluation. We ensure that no password appears in both sets. Additionally, we define *cross-evaluation* test sets by removing RockYou training passwords from the LinkedIn evaluation set and vice versa to evaluate generalization to unseen distributions.

Finally, we also consider the MySpace, phpBB, and Hotmail [52] leaks as evaluation sets. We perform the analogous cross-evaluation procedure to remove RockYou and LinkedIn training data from all of them.

Table 1: Main facts about the datasets used in this work.

Name	Unique passwords	Year
LinkedIn [50]	60,505,270	2012
RockYou [51,52]	14,344,391	2009
phpBB [52]	184,318	2009
MySpace [52]	37,144	2006
Hotmail [52]	8,931	Unknown

3.2 Our Models

Transformers are a versatile and broad family of deep-learning models as discussed above in Section 2.2. For password guessing, we are interested in autoregressive generative models (see Section 2.1). *PassGPT* and *PassVQT* model the probability of a character in a password, given the previous ones: $p(x_i|x_0, \cdots, x_{i-1}; \theta)$. Sampling sequentially from this distribution can generate likely passwords. Our models operate over a vocabulary, Σ, comprising 256 UTF-8 characters.

Neural networks require a vector representation of tokens as input. We define a *tokenizer* as a function that maps every character σ in the vocabulary to an integer,

$$\text{tokenizer} : \Sigma \mapsto [0, |\Sigma| - 1]. \tag{3}$$

Then, a vector representation is created for each token σ using a one-hot encoding of its image under the tokenizer function. This results in a vector of dimension

$|\Sigma|$, with all entries equal to zero and a single entry equal to 1 at position tokenizer(σ).

PassGPT , depicted in Figure 1, is an implementation of the GPT-2 architecture [39]. GPT models utilize the decoder component of transformers and are trained to predict the next token in a sequence autoregressively. To predict a specific character x_i in a password, the transformer decoder considers only previous characters $x_0, \ldots, x_{i-1}$ as input, and outputs a latent vector with dimension d ($d = 768$ in our work). This latent vector is then mapped into a real vector of dimension $|\Sigma|$ through a linear layer and further transformed into a probability distribution over the vocabulary using the *softmax* function. The output distribution over the vocabulary represents $p(x_i|x_{<i}; \theta)$. This distribution is optimized using cross-entropy loss with respect to the one-hot-encoding representation of the true character found at that position.

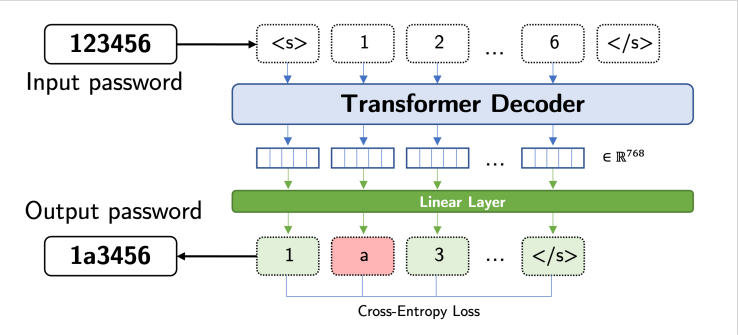


Fig. 1: *PassGPT* autoregressively predicts the input character at position n using all previous tokens. Green indicates correct prediction; red indicates incorrect.

Once the network is trained, it provides us with a parameterized distribution over our vocabulary conditioned on previous tokens, namely, $p(x_i|x_{<i}; \theta)$. For generation purposes, we can start from the start-of-password token, `<s>`, and find $p(x_1|x_0 =$ `<s>`$)$. This assigns a probability to every character in our vocabulary to be the first token in the password. If we sample from this distribution, we can fix the first character and repeat the process to find the second one by computing $p(x_2|x_0, x_1)$. The sampling process for a password finishes when the end-of-password token, `</s>`, is sampled from the distribution at any given step. Unlike training, this process is sequential.

Our implementation of PassGPT uses the HuggingFace library [53] and has the following specifications: 12 attention heads, 8 decoder layers, and GeLU activation [24]. Additionally, we train all models for 1 epoch with AdamW optimizer and a starting learning rate of 5e-5 with linear decay during training.

PassVQT enhances the transformer architecture with vector-quantization of the latent space. In the computer vision domain, this has been shown to improve

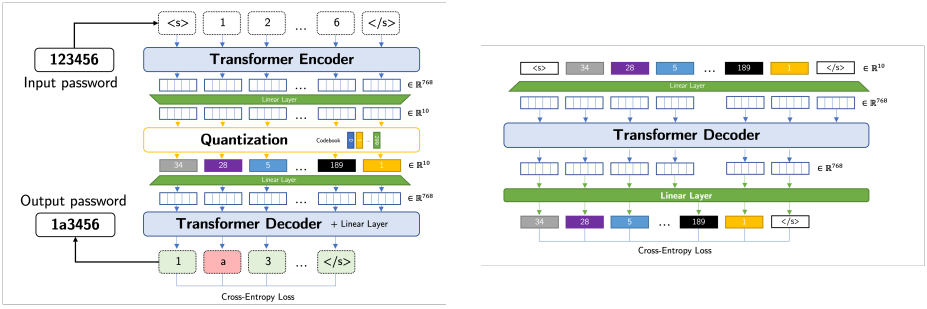


Fig. 2: Overview of PassVQT showing (left) an end-to-end model trained to compress passwords into a quantized latent space, where each code represents a fixed vector of dimension 768 and (**right**) an autoregressive GPT model that parameterizes the conditional distribution of indices. The latter is trained once the first one has converged and is required for generation. Transformer decoders in both models are independent.

sample quality [55]. PassVQT follows the architecture designed by Yu et al. [55]. While modeling the same conditional distribution as PassGPT, we aim to assess whether quantization can provide any additional benefits. In this architecture, depicted in Figure 2, a transformer encoder maps each input token to a latent representation with a dimension of 768. This latent representation is then mapped to 10 dimensions using a linear layer and quantized using k-means and a codebook with N entries. The quantized 10-dimensional vectors are mapped back to 768 dimensions through a linear layer and serve as input to a transformer autoregressive decoder. This decoder is trained to reconstruct the input password character by character, using only the quantized representations for previous tokens.

We carried out a hyperparameter search by minimizing the reconstruction loss on the RockYou leak's training split. Our findings showed that deeper encoder and decoder structures offered better results, with a codebook size of 300 providing the best performance. PassVQT employs a transformer encoder and a GPT-2 decoder with 12 attention heads and 8 layers, respectively. It was implemented on using the HuggingFace library [53] and trained end-to-end with AdamW optimizer and a starting learning rate of 5e-5 with linear decay.

Once the encoder-decoder network has converged, the model can reconstruct input passwords from a compressed quantized latent representation. If we model the distribution of latent codes, we can sample from it to produce a likely sequence of codes, which the decoder can then transform into likely passwords. For this, we train an autoregressive *codes model* over the quantized representation of the training dataset. During inference, we create new passwords by sampling sequences of codes from the *codes model* and transforming them into passwords using the original decoder. The encoder is no longer needed.

Table 2: Percentage of RockYou test set (10 characters or fewer) guessed from 10^7 generations. Models are trained on either all passwords or unique entries from RockYou.

Architecture	Trained on	% Test set guessed
PassGPT	Unique	**4.25%**
	All passwords	0.53%
PassVQT	Unique	0.14%
	All passwords	**2.86%**

4 Evaluation

Our foremost contribution focuses on password generation. This section compares PassGPT and PassVQT with state-of-the-art deep generative models and demonstrates their generalization to different datasets without the need for further training. We also examine the probabilities and entropies of passwords under PassGPT to provide insights into its capabilities and modeled distribution. Finally, we analyze the alignment of these probabilities with password-strength estimators and discuss how they can be used to improve strength estimation.

4.1 Password Generation

For a fair comparison with PassGAN [25] and its improved version (PassGAN+) [36], we train PassGPT and PassVQT using 80% of passwords of at most 10 characters in the RockYou leak. The evaluation of the generation process is determined by the percentage of passwords from a disjoint test set that the models can generate. In this case, the test set comprises the unique passwords in the remaining 20% of the RockYou leak that are not in the training set.

We consider two variations of the training set: (1) unique passwords and (2) all occurrences. PassGPT demonstrates superior generalization when trained on unique passwords, as detailed in Table 2. Conversely, PassVQT experiences difficulty generating in-distribution passwords when trained on unique entries but significantly improves upon incorporating their absolute occurrences.

We sample increasingly large pools of password guesses from PassGPT (trained on unique passwords) and PassVQT (trained on all passwords) and calculate the percentage of the RockYou test split they recover. Results in Table 3 show that PassGPT outperforms all other models. It recovers 41.9% of the test set among 10^9 guesses, whereas state-of-the-art GAN models matched 23.33%. PassVQT performance surpasses that of the original PassGAN and stays close to that of the PassGAN improved version.

Another important factor in password generation evaluation is the ability to generate novel and distinct samples. We compared the percentage of unique passwords generated by our models to those from PassGAN; results are shown in Figure 3. PassGPT retains the highest percentage of unique passwords (60%), whereas PassVQT drops to 20% of unique passwords among 10^9 guesses. Since PassVQT was trained on all occurrences of passwords, under its distribution

Table 3: Percentage of the RockYou test split (<10 characters) matched by samples from various models. PassGAN* stands for the improved PassGAN presented in [36]. Results for GANs were taken directly from original papers [25,36] and not reproduced.

Guesses	PassGAN	PassGAN*	PassVQT	PassGPT	PassGPT ∪ PassVQT
10^4	0.01%	-	0.004%	0.01%	0.01%
10^5	0.05%	-	0.05%	0.05%	0.10%
10^6	0.38%	-	0.45%	**0.50%**	0.93%
10^7	2.04%	-	2.90%	**4.25%**	6.39%
10^8	6.73%	9.51%	10.30%	**19.37%**	22.70%
10^9	15.09%	23.33%	21.46%	**41.86%**	44.66%

common passwords are more likely to be generated, reducing the number of novel passwords. PassGAN stays between them with approximately 40% unique entries.

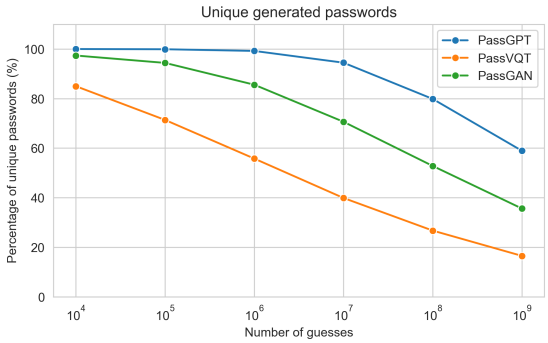


Fig. 3: Log-Linear plot of unique passwords generated by different architectures.

4.2 Generalizing to Longer Passwords and Unseen Distributions

Our models outperform state-of-the-art deep generative models in a common setup. To further evaluate the effectiveness of our models, we extend the modeling to longer passwords, which are more representative of real-world distributions. We train PassGPT and PassVQT on passwords with up to 16 characters (including longer passwords primarily increases entries that are difficult to guess). We again train the models using both unique and all occurrences of the data. PassGPT, as before, performs best when trained on unique samples. Surprisingly, PassVQT now obtains better performance when trained on unique passwords. After training on this new distribution, models retain similar accuracy. From 10^8 guesses, PassGPT and PassVQT recover 15.5% and 8.57% of the test set, respectively, compared to 19.37% and 10.30% in the 10-character setting. From now on, we will focus on 16- character models for a richer analysis.

To assess the models' generalization to unseen password distributions, we test them on leaks different from the RockYou leak. Although users tend to reuse passwords, they are likely to vary based on platform and year of creation[15,6]. We first analyze the LinkedIn leak, which is the largest of our samples and was obtained 3 years after the RockYou leak. To determine how well RockYou models generalize, we benchmark them against a PassGPT model trained solely on 80% of LinkedIn data. We take the remaining 20% as the test set after excluding any passwords present in the RockYou training set. This results in a test set of over 11M unique passwords unseen by any of the models during training. We evaluate the models' performance by determining the percentage of test passwords generated by each architecture. RockYou models achieve comparable results to the LinkedIn-trained PassGPT, as shown in Table 4, demonstrating the ability of autoregressive models to parameterize rich distributions that generalize beyond the training leak without the need for retraining.

Table 4: Percentage of passwords from the LinkedIn test split guessed. Columns indicate training distribution. The test set does not contain passwords in the RockYou training set.

Guesses	PassGPT		PassVQT
	RockYou	**LinkedIn**	**RockYou**
10^4	0.001%	0.001%	0.001%
10^5	0.012%	0.010%	0.012%
10^6	0.11%	0.10%	0.13%
10^7	1.03%	0.94%	1.10%
10^8	6.03%	6.80%	5.41%

Finally, we evaluate the RockYou and LinkedIn models to determine which training leak leads to a better generalization. The models are tested on three additional datasets: phpBB, MySpace, and Hotmail (refer to Section 3.1), after removing passwords present in either the RockYou or LinkedIn training sets. The results are shown in Table 5. RockYou models exhibit superior performance, with password recovery rates of 9.45%, 11.39%, and 7.22% from 10^8 guesses.

4.3 Guided Generation

We propose a novel approach to password generation: *guided password generation*. Unlike previous deep generative methods that generate passwords as a whole, PassGPT models each token separately, granting full control over each character. This allows the generation process to meet specific constraints. Some examples of these constraints are: password length, fixed characters (e.g., "a" at first position) and templates (e.g., four lowercase letters and two numbers). This can be achieved by restricting the sampling distribution $p(x_i|x_0, \cdots, x_{i-1})$ to consider only the probability mass assigned to a subset of interest $\Sigma' \subset \Sigma$; for instance, limiting Σ'

Table 5: Percentage of phpBB, MySpace, and Hotmail leaks generated by PassGPT trained on RockYou, compared with PassGPT trained on LinkedIn. Evaluation is performed on the entire leak after entries contained in the RockYou training set are removed.

Guesses	PassGPT trained on RockYou			PassGPT trained on LinkedIn		
	phpBB	**MySpace**	**Hotmail**	**phpBB**	**MySpace**	**Hotmail**
10^4	0.002%	0%	0%	0%	0%	0%
10^5	0.02%	0%	0.02%	0.008%	0%	0%
10^6	0.20%	0.22%	0.18%	0.10%	0.10%	0.05%
10^7	1.80%	2.06%	1.24%	0.77%	0.94%	0.61%
10^8	9.45%	11.39%	7.13%	6.02%	6.57%	4.67%

to lowercase letters for the first four tokens. The resulting password generation is guided by these constraints while still being likely under the modeled password distribution. Table 6 shows various templates and their corresponding generations produced by PassGPT.

Table 6: *Guided generation* examples from PassGPT. Templates formatted using `l` for lowercase, `u` for uppercase, `d` for digit, `p` for punctuation, and `*` for any character.

`llllll`	`llldd`	`ullppdd`	`uuuu**dd`
orange	manb13	Nms__12	PARLA198
iluvma	sall89	Zac&&09	CELAN777
gikiyd	lowm12	Chl@(18	QWER1234

4.4 Probabilities and Entropies Estimates by PassGPT

One of the main advantages of autoregressive models is having access to an explicit representation of the modeled distribution. We exploit this property to provide further intuitions behind the PassGPT generation process.[6] The probability of a password is estimated as the product of the conditional probability for each sampled character, which is more conveniently represented as the log probability (Equation 4). Furthermore, the entropy measures the uncertainty in the model for each token and is calculated according to Equation 5.

$$\log_{10} p(\mathbf{x}; \theta) = \sum_{i=1}^{n} \log_{10} p(x_i | x_1, \dots x_{i-1}; \theta). \tag{4}$$

$$H(X_i) = \sum_{x_i \in \Sigma} p(x_i | x_{<i}; \theta) \cdot \log_2 p(x_i | x_{<i}; \theta). \tag{5}$$

[6] For PassVQT, this is not possible, as the modeled distribution is in the codebook space, and different codes can lead to the same password generation.

We computed the log probability and entropy for every position in all unique passwords (<16 characters) in the RockYou dataset using PassGPT. Examples of passwords with different probabilities under the model can be found in Appendix A. Figure 4a depicts the distribution of the entropy for characters found at specific positions in passwords of length 16. The entropy of the first character is slightly above five; it is constant since $p(x_1|x_0 = \texttt{<s>})$ is equal across passwords. The median entropy decreases as we move towards the last positions because the model reduces uncertainty as more characters are observed. Figure 4b illustrates how the log-probability of passwords decreases with length, with the median log-probability dropping by approximately 1 unit for each additional character. This corresponds to an average probability of 0.1 for each new character.

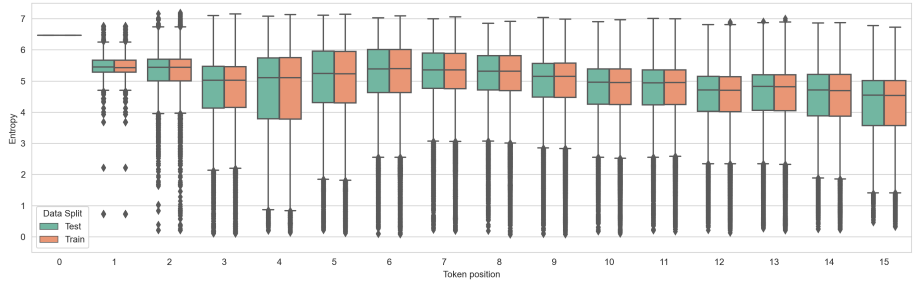
We can analyze password probabilities under the model compared to brute-force search. Our vocabulary Σ contains 256 characters. Therefore, the log-probability of discovering a password of length 3 through brute force can be approximated as $\log_{10}(1/256^3) \approx -7.5$. This value is close to the median log-probability under PassGPT. However, the utility of generative methods becomes evident when we deal with longer passwords that are computationally infeasible to uncover through exhaustive search. For instance, the log-probability of successfully recovering a 16-character password using brute-force attacks is -38.5. In contrast, the median log-probability under PassGPT hovers around -18. This indicates that finding a 16 character password using PassGPT is approximately 10^{20} times more likely than relying on random guessing.

4.5 Discussion: PassGPT vs PassVQT

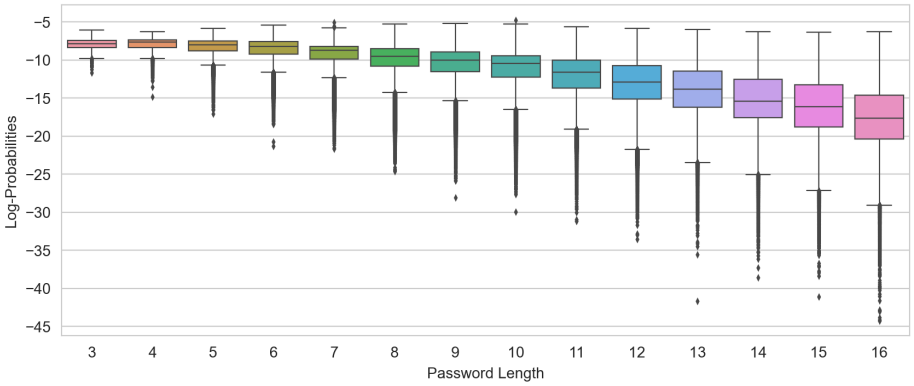
We wrap up this section with a brief discussion about the main differences between PassGPT and PassVQT, and when to use each of them. Details can be found in Appendix B. Overall, these models can surpass state-of-the-art deep generative models and generalize to unseen distributions. Focusing on models trained on 16 characters, we can highlight several differences:

1. PassVQT generates longer passwords than PassGPT.
2. PassGPT guesses weaker passwords, while PassVQT matches stronger passwords.
3. PassGPT generates more unique passwords than PassVQT: 84% vs 76%.
4. PassGPT can generate passwords faster than PassVQT: 12h vs 24h to generate 10^8 samples on 1 NVIDIA RTX3090.

All things considered, PassGPT seems better at modeling the actual leaked distribution and generating in-distribution samples. On the other hand, PassVQT is "more imaginative" and creates stronger passwords with a similar distribution to that of the leaked file. Nevertheless, the sampling process of both models can be tweaked to pursue specific goals. For instance, if we want to reduce out-of-distribution samples, we can perform top-k sampling for each character, considering only the most likely tokens under the model to avoid long-tail passwords. Similarly, to incentivize the generation of stronger and less likely passwords, we can increase the temperature of the *softmax* function, or avoid sampling from the top-k most likely tokens.



(a) Entropy distribution per token for passwords of length 16 under PassGPT.



(b) Log-probabilities with respect to length for all passwords in RockYou.

Fig. 4: Entropy and log-probability of passwords in the RockYou leak under PassGPT

4.6 Password Strength Estimation

In the previous section, we carried out a comprehensive analysis of the fundamental characteristics of the probabilities and entropies of passwords in PassGPT. In this section, we delve deeper into the relationship between probability, entropy, and password strength to gain a better understanding of the modeled distribution. For each unique password in the RockYou and LinkedIn leaks, we calculate its log-probability and entropy under PassGPT and its strength as determined by *zxcvbn* [48]. This method assigns a score ranging from 0 (very weak) to 4 (very strong). The distribution of log-probabilities and entropies for each strength score is illustrated in Figure 5. The results show that PassGPT assigns lower probability and higher entropy to stronger passwords, thereby demonstrating that weak passwords are more likely in the modeled distribution.

Finally, we conduct a manual examination of outliers in the distributions to better understand when PassGPT does not align with *zxcvbn*. Our analysis revealed three distinct phenomena where the model assigns low probabilities to passwords that are considered weak by *zxcvbn*. These examples are hard to model for PassGPT, but easy to detect using dictionary attacks.

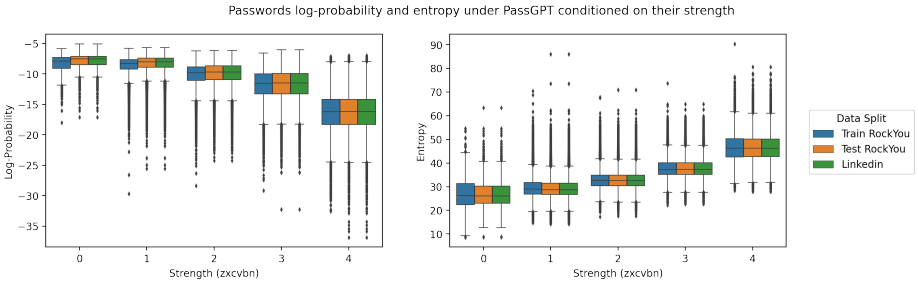


Fig. 5: Log-probability and entropy under the model according to password strength (`zxcvbn`)

1. Pattern repetition. These passwords are composed of a sequence that is repeated several times. Examples: ":X:X:X:X:X:X", "qwertqwertqwert".
2. Replacement of characters in common words by similar symbols. A dictionary attack is successful to find these passwords. Examples: "k1m83rly" (from "kimberly") or "r00sevelt" (from "roosevelt").
3. Reversed words. These can also be easily detected by *zxcvbn* but are unlikely under the model distribution. Example: "llabtooF" (from "Football").

On the other hand, there are very strong passwords, according to *zxcvbn*, that obtain high probabilities under the model. We can also identify predominant phenomena:

1. Passwords containing non-English words. *zxcvbn* tries to decompose them as English words unsuccessfully. For example, the password "teamomiamorcito" is formed by the Spanish words "te amo mi amorcito" ("I love you my love"). However, *zxcvbn* parses it as "team", "omi", "amorcito".
2. Love-related passwords. "iloveyou" is one of the most common passwords in RockYou. When analyzing passwords with strength 4 that obtained high probabilities, we found copious variations of it. Examples: "ilovematt4eva", "ilovetoby4eva", "ilovetyler4ever", "ilovehotmail", "iloveyousomuch". The suffixes "4ever" and "4eva" are very common among these passwords.

It is crucial for strength estimators to minimize the number of false negatives, i.e., classifying passwords that can be guessed by any existing technique as strong. Our analysis revealed instances of very strong passwords with high probabilities under PassGPT, indicating that they are likely to be discovered by such a model. We believe that incorporating the log-likelihood from generative models into existing password strength estimators could provide valuable supplementary information and improve the accuracy of these systems for high-stake scenarios.

5 Conclusions

In this work, we investigated the use of large language models to model password distributions without explicit supervision. We introduced two autoregressive architectures that model the conditional distribution of characters based on

previous ones: PassGPT and PassVQT. PassGPT might be preferable because it provides access to an explicit probability distribution, is simpler, and provides faster generation. However, PassVQT might still be helpful for scenarios where we want to express more variability and generate more complicated passwords that are still close to the training distribution.

Advantages of autoregressive models over state-of-the-art GAN generators include *guided password generation* and access to an explicit probability distribution. We have analyzed how the log-probabilities of passwords under PassGPT align with their strength, and how this metric could be used to mitigate limitations in strength estimators.

Overall, this work seeds many promising research directions in the field of password modeling using large language models that are to be explored by future research.

References

1. Hashcat: Advanced password recovery. `https://hashcat.net/hashcat/`
2. Hashcat: Advanced password recovery - Attacks Wiki. `https://hashcat.net/wiki/`
3. Hashcat: Advanced password recovery - Mask attack. `https://hashcat.net/wiki/doku.php?id=mask_attack`
4. Hashcat: Advanced password recovery - Rule-based attack. `https://hashcat.net/wiki/doku.php?id=rule_based_attack`
5. Hashcat: Advanced password recovery - Slow candidates mode. `https://github.com/hashcat/hashcat/blob/master/docs/slow-candidates-mode.md`
6. Bailey, D.V., Dürmuth, M., Paar, C.: Statistics on password re-use and adaptive strength for financial accounts. In: Security and Cryptography for Networks: 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings 9. pp. 218–235. Springer (2014)
7. Blocki, J., Harsha, B., Zhou, S.: On the economics of offline password cracking. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 853–871. IEEE (2018)
8. Bond-Taylor, S., Leach, A., Long, Y., Willcocks, C.G.: Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. IEEE transactions on pattern analysis and machine intelligence (2021)
9. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
10. de Carné de Carnavalet, X., Mannan, M.: From very weak to very strong: Analyzing password-strength meters. In: Network and Distributed System Security Symposium (NDSS 2014). Internet Society (2014)
11. Carnavalet, X.D.C.D., Mannan, M.: A large-scale evaluation of high-impact password strength meters. ACM Transactions on Information and System Security (TISSEC) **18**(1), 1–32 (2015)
12. Castelluccia, C., Dürmuth, M., Perito, D.: Adaptive password-strength meters from markov models. In: NDSS (2012)
13. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S., et al.: Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311 (2022)

14. Ciaramella, A., D'Arco, P., De Santis, A., Galdi, C., Tagliaferri, R.: Neural network techniques for proactive password checking. IEEE Transactions on Dependable and Secure Computing **3**(4), 327–339 (2006)
15. Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X.: The tangled web of password reuse. In: NDSS. vol. 14, pp. 23–26 (2014)
16. Dell'Amico, M., Michiardi, P., Roudier, Y.: Password strength: An empirical analysis. In: 2010 Proceedings IEEE INFOCOM. pp. 1–9. IEEE (2010)
17. Dürmuth, M., Angelstorf, F., Castelluccia, C., Perito, D., Chaabane, A.: 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. pp. 119–132. Springer (2015)
18. Feldmeier, D.C., Karn, P.R.: Unix password security-ten years later. In: Advances in Cryptology—CRYPTO'89 Proceedings. pp. 44–63. Springer (2001)
19. Golla, M., Beuscher, B., Dürmuth, M.: On the security of cracking-resistant password vaults. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 1230–1241 (2016)
20. Golla, M., Dürmuth, M.: On the accuracy of password strength meters. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1567–1582 (2018)
21. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems. vol. 27. Curran Associates, Inc. (2014), https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
22. Greenbag, A.: Hackers are passing around a megaleak of 2.2 billion records. https://www.wired.com/story/collection-leak-usernames-passwords-billions/ (2019)
23. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. Advances in neural information processing systems **30** (2017)
24. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
25. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: A deep learning approach for password guessing. In: International conference on applied cryptography and network security. pp. 217–237. Springer (2019)
26. Melicher, W., Ur, B., Segreti, S.M., Komanduri, S., Bauer, L., Christin, N., Cranor, L.F.: Fast, lean, and accurate: Modeling password guessability using neural networks. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 175–191 (2016)
27. Morris, R., Thompson, K.: Password security: A case history. Communications of the ACM **22**(11), 594–597 (1979)
28. 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 (2005)
29. OpenAI: Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt/ (2022)
30. Openwall: John the ripper markov generator. https://openwall.info/wiki/john/markov
31. Openwall: John the ripper password cracker. https://www.openwall.com/john/
32. Pagnotta, G., Hitaj, D., De Gaspari, F., Mancini, L.V.: PassFlow: Guessing passwords with generative flows. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 251–262. IEEE (2022)

33. Pal, B., Daniel, T., Chatterjee, R., Ristenpart, T.: Beyond credential stuffing: Password similarity models using neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 417–434. IEEE (2019)
34. Pasquini, D., Ateniese, G., Bernaschi, M.: Interpretable probabilistic password strength meters via deep learning. In: European Symposium on Research in Computer Security. pp. 502–522. Springer (2020)
35. Pasquini, D., Cianfriglia, M., Ateniese, G., Bernaschi, M.: Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 821–838 (2021)
36. Pasquini, D., Gangwal, A., Ateniese, G., Bernaschi, M., Conti, M.: Improving password guessing via representation learning. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1382–1399. IEEE (2021)
37. Paterson, K.G., Stebila, D.: One-time-password-authenticated key exchange. In: Information Security and Privacy: 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings 15. pp. 264–281. Springer (2010)
38. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
39. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
40. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985)
41. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Advances in neural information processing systems **27** (2014)
42. Tomczak, J.M.: Deep generative modeling. Springer (2022)
43. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
44. Ur, B., Kelley, P.G., Komanduri, S., Lee, J., Maass, M., Mazurek, M.L., Passaro, T., Shay, R., Vidas, T., Bauer, L., et al.: How does your password measure up? the effect of strength meters on password creation. In: USENIX Security Symposium. pp. 65–80 (2012)
45. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
46. Wayman, J.L., Jain, A.K., Maltoni, D., Maio, D.: Biometric systems: Technology, design and performance evaluation. Springer Science & Business Media (2005)
47. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 30th IEEE Symposium on Security and Privacy. pp. 391–405. IEEE (2009)
48. Wheeler, D.L.: zxcvbn: Low-budget password strength estimation. In: USENIX security symposium. pp. 157–173 (2016)
49. Whitney, L.: Billions of passwords leaked online from past data breaches. `https://www.techrepublic.com/article/billions-of-passwords-leaked-online-from-past-data-breaches/` (2021)
50. Wikipedia: 2012 linkedin hack. `https://en.wikipedia.org/wiki/2012_LinkedIn_hack` (2023), last accessed 21 Jan 2023
51. Wikipedia: Rockyou. `https://en.wikipedia.org/wiki/RockYou#Data_breach` (2023), last accessed 21 Jan 2023
52. WikiSkull: Password datasets. `https://wiki.skullsecurity.org/index.php/Passwords` (2023), last accessed 21 Jan 2023

53. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020), https://www.aclweb.org/anthology/2020.emnlp-demos.6
54. Xu, M., Wang, C., Yu, J., Zhang, J., Zhang, K., Han, W.: Chunk-level password guessing: Towards modeling refined password composition representations. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 5–20 (2021)
55. Yu, J., Li, X., Koh, J.Y., Zhang, H., Pang, R., Qin, J., Ku, A., Xu, Y., Baldridge, J., Wu, Y.: Vector-quantized image modeling with improved vqgan. arXiv preprint arXiv:2110.04627 (2021)

A Passwords at the quantiles of the distribution

In our analysis in Section 4.4, we examined password probabilities under PassGPT in relation to password strength. PassGPT assigns lower probabilities to stronger passwords and higher probabilities to weaker ones. In Figure 6, we illustrate some passwords along with their log-probabilities under PassGPT for visual exploration.

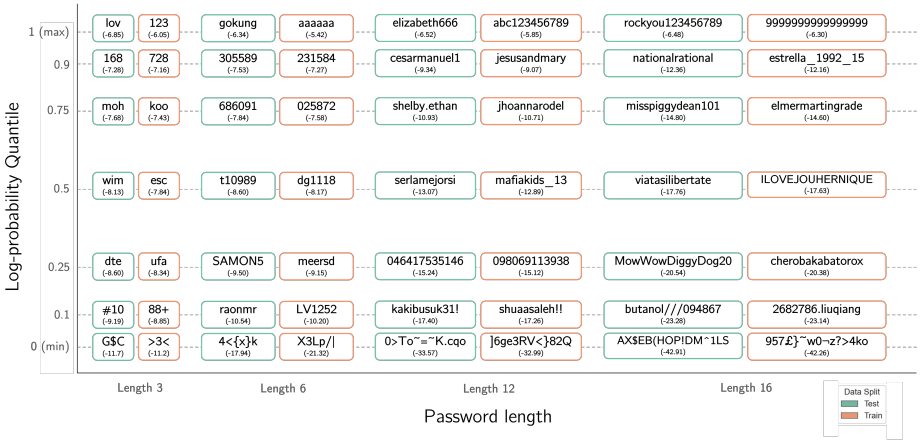


Fig. 6: Passwords in RockYou located at the quantiles of the probabilities distribution for different password lengths. The lower the quantiles, the more unlikely the password is under the model.

B PassGPT vs PassVQT

This section includes a detailed comparison between PassGPT and PassVQT generations. Table 7 illustrates how many passwords are guessed by each method

conditioned on their strength. Finally, Figure 7 depicts a histogram of the length of passwords generated by each architecture. PassVQT overall tends to generate longer and more difficult passwords, but PassGPT better fits the distribution of easy passwords, improving its overall performance.

Table 7: Detailed statistics on the matched passwords by different models from RockYou test set (<16 characters) with respect to their strength. Passwords are only considered once.

Strength	**Total**	**Guessed by**			**Not guessed**
		PassGPT ∩ PassVQT	PassGPT	PassVQT	
0	2,035	41 (2.0%)	481 (23.7%)	28 (1.4%)	1485 (73%)
1	752,137	150,945 (20.1%)	111,704 (14.9%)	66,004 (8.8%)	423,484 (56.3%)
2	926,826	45,337 (4.9%)	50,704 (5.5%)	53,782 (5.8%)	777,003 (83.8%)
3	558,029	4,501 (0.8%)	7,831 (1.4%)	8,832 (1.6%)	536,865 (96.2%)
4	158,635	88 (0.06%)	194 (0.1%)	152 (0.09%)	158,201 (99.7%)
	2,397,662	200,912 (8.38%)	170,914 (7.12%)	128,798 (5.37%)	1,897,038 (79.12%)

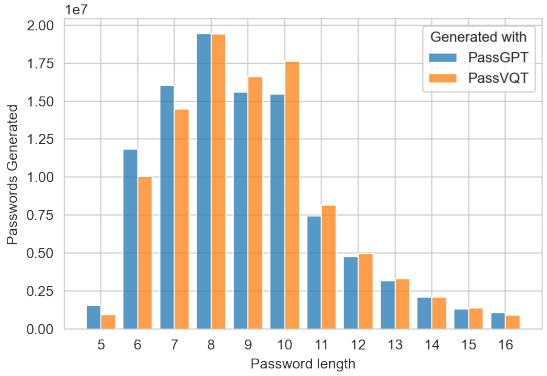


Fig. 7: Histogram of generated password length by each model on a subset of 10^8 samples.