USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Password Guessing Using Large Language Models

Yunkai Zou, Maoxiang An, and Ding Wang, *Nankai University*

https://www.usenix.org/conference/usenixsecurity25/presentation/zou-yunkai

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Password Guessing Using Large Language Models

Yunkai Zou, Maoxiang An, Ding Wang
College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China; wangding@nankai.edu.cn
Academy for Advanced Interdisciplinary Studies, Nankai University, Tianjin 300350, China
NDST, DISSec, TBI Center, Nankai University, Tianjin 300350, China

Abstract

Passwords are ubiquitously used for authentication/encryption, and password guessing attacks are the most effective technique for evaluating password strength. While large language models (LLMs) like ChatGPT-4o have demonstrated remarkable capabilities in text comprehension and reasoning across various general natural language processing tasks, they face limitations due to their static knowledge (e.g., fixed training data that lacks domain adaptability), especially in specialized tasks such as generating accurate password guesses.

This work provides *a brand new technical route* for password guessing, by proposing an LLM-based guessing framework, namely PASSLLM, that leverages low-rank adaptation techniques. PASSLLM systematically addresses four major password guessing scenarios, each of which is based on varied kinds of information available to the attacker. To reduce the high computation costs in password generation with LLMs, we propose two generation algorithms tailored for trawling and targeted guessing, respectively, enabling efficient password generation at scale (e.g., 1,000 guesses per user). Further, we apply model distillation to improve the generation speed by 11.5 times in trawling guessing scenarios without significantly reducing the success rate. Particularly, our generation algorithms are applicable to a wide range of decoder-only-based LLMs (e.g., Mistral, Llama-2/3, and Qwen-2).

Extensive experiments on 11 real-world password datasets demonstrate the effectiveness of our framework: (1) PASSLLM for trawling guessing scenarios, whose guessing success rates are generally 2.87%-17.07% higher than its foremost counterpart; (2) PASSLLM-I for targeted guessing based on personally identifiable information (PII), which guesses 12.54%-31.63% of common users within 100 guesses, outperforming its foremost counterpart by 15.10%-45.98%; (3) PASSLLM-II for targeted guessing based on users' password reuse behaviors, which outperforms its foremost counterpart by 6.31%-13.87%; and (4) PASSLLM-III for targeted guessing based on users' PII and sister password(s), which outperforms its foremost counterpart by 13.44%-36.14%. We believe this work makes a substantial step toward introducing LLMs into the password guessing domain.

1 Introduction

Passwords are short, human-memorable keys, and are the major cryptographic component that users can perceive and directly access. They have broad applications in critical areas such as authentication [23], encryption [53], digital signatures [14], and key exchange [21]. Despite criticism regarding security and usability issues, passwords remain irreplaceable in the foreseeable future due to their simplicity, low cost, ease of change, and superior deployability compared to alternative authentication technologies [8, 9, 16, 54].

However, the widespread use of passwords poses significant security risks, particularly when weak passwords are involved. The strength of a password is relative, with those easily guessed by attackers considered weak [30, 45, 47]. Given the increasing number of password breaches (e.g., 184 million logins leaked from major tech platforms [13]) and the large-scale leakage of personally identifiable information (PII, e.g., the LoanDepot breach with 16.9 million customer information [48] and the Indonesia's National Civil Service Agency breach with 4.7 million records [3]), it is critical to investigate password security from the perspective of an attacker, especially in the context of targeted guessing attacks (that exploit victims' PII and/or sister passwords).

Password guessing research can generally be divided into three stages based on the methodologies employed. The first stage, before 2008, was characterized by heuristic attacks, where various character transformation rules were designed through "creative ideas". During this period, password guessing was more of an art. In the second stage, from 2009 to 2015, password guessing entered into the realm of science, which is mainly driven by various statistical models, including probabilistic context-free grammars (PCFG [46]) and Markov chains [25, 27]. The third stage, since 2016, coincided with the prosperity of deep learning, where researchers began exploring various deep learning techniques (such as password models based on recurrent neural networks [26, 30], and on generative adversarial networks [18, 32]) to design sophisticated password models, aiming to use neural networks to overcome the challenges of data sparsity and overfitting inherent in traditional statistical methods.

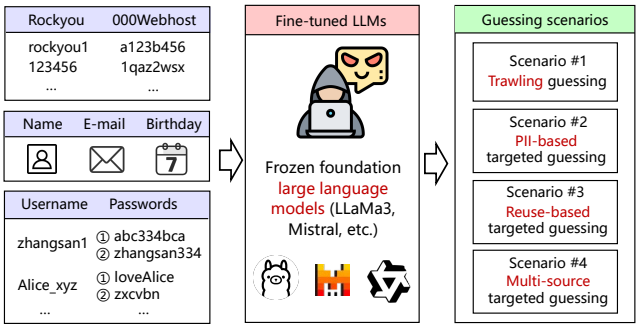


Figure 1: An illustrative example of the PASSLLM framework applied to four guessing attacks: trawling, PII-based, reuse-based, and multi-source (PII and existing passwords) targeted guessing scenarios. The attacker fine-tunes existing large language models using password datasets, PII datasets, and multi-password datasets, and then conducts the guessing attacks.

Recently, the rapid development of large language models (LLMs, e.g., Llama [37], Mistral [20], and GPT-4 [28]) has demonstrated outstanding capabilities in text generation and understanding, along with significant potential across various downstream tasks. Password guessing attacks can be fundamentally viewed as attempts to approximate the probability distribution of passwords (i.e., textual sequences), making them inherently suited for transformer-based LLMs that employ autoregressive generation methods.

1.1 Design challenges

There are several potential technical approaches to generating passwords using LLMs, such as constructing prompts to directly call APIs of pre-trained models or fine-tuning LLMs with password data. While these methods may seem straightforward, they present significant challenges in practice.

At IEEE S&P 2025, Yang and Wang [52] mention that LLMs may not be suitable for password guessing due to fundamental differences between passwords and natural language. Specifically, passwords are typically short, and consist of a limited vocabulary (e.g., >99% of passwords are composed of 95 printable characters; see Table 1), but some recent LLMs (e.g., Mistral-7B-v0.1[1], Qwen2.5[2]) employ vocabularies with over 10,000 tokens. Moreover, LLMs inherently offer slow inference speeds and will face challenges in generating guesses. This highlights the first major challenge of using LLMs for password guessing: Ensure that the LLM can efficiently and reliably generate sufficient and effective/diverse guesses.

More specifically, password guessing typically involves generating a relatively large number (>1,000) of short character sequences (e.g., 6-16 characters). In contrast, LLMs are primarily used to generate human natural language (e.g., in conversational contexts), where the generated sequences tend to be much fewer and longer compared to those used in password guessing tasks. Currently, there is *no* well-optimized inference algorithm tailored to the password guessing task. Furthermore, due to the vast number of parameters in most state-of-the-art open-source LLMs, using their default inference algorithms directly requires significantly more computational resources than traditional password models like FLA [47] and UNCM [30]. Without a carefully designed generation algorithm, LLMs' VRAM requirements may exceed the capacity of consumer-grade GPUs (e.g., 24GB VRAM of NVIDIA 4090), making large-scale password generation costly.

While directly calling an LLM (e.g., GPT-4 [28]) may seem feasible, the model tends to generate passwords that appear plausible but do not align with the password creation behaviors of real-world users (see typical examples in Table 14 of the full version of our paper `https://bit.ly/43Woqk5`). Actually, whether fine-tuned or accessed via an API, the default inference algorithm frequently fails to generate sufficient guesses or results in duplicate outputs when called multiple times (see Table 13 in the full version). Additionally, ethical concerns have impeded the use of existing LLMs for large-scale password generation. Some models, such as GPT-4, refuse to generate extensive password lists, citing privacy and security concerns. These limitations greatly impair the direct use of LLMs in password guessing applications.

Second, LLMs are designed and trained primarily for natural language processing, where minor grammatical errors or inconsistencies are generally tolerable, as they do not significantly affect the overall meaning. However, password guessing differs greatly from traditional NLP tasks. Passwords do not follow grammatical conventions (e.g., `iloveu4ever`) and require *exact* matches. For example, a guess of `p@ssw0rd123` will not correctly match `p@ssword123`, even though the two are very similar. This need for precision poses a significant challenge for LLMs in the context of password guessing.

To address these challenges, we employ efficient parameter fine-tuning (e.g., Low-Rank Adaptation, LoRA [19]) on LLMs to build the PASSLLM framework, which stands as the most versatile password guessing framework to date for a wide range of attack scenarios (see Fig. 1). More specifically, by designing various fine-tuning paradigms, PASSLLM is highly effective for trawling guessing and three targeted guessing scenarios: those based on the victim's PII, sister password(s) leaked from her other accounts, and a combination of both. Particularly, we propose two inference algorithms tailored to password guessing, which enables PASSLLM to generate a sufficient number of guesses efficiently and reliably. Experimental results show that our PASSLLM(-I∼III) outperform the state-of-the-art password models by 2.87%-17.07%, 15.10%-45.98%, 6.31%-13.87% and 13.44%-36.14% in trawling, PII-based, reuse-based, and multi-source combined (PII and sister passwords) attacks.

Further, we apply model distillation [17] to reduce the size of PASSLLM from 7B to 0.5B parameters, achieving an 11.5-fold speed improvement in trawling password generation without significantly reducing the success rate.

[1] `https://huggingface.co/mistralai/Mistral-7B-v0.1`
[2] `https://huggingface.co/collections/Qwen`

1.2 Contributions

We summarize our main contributions as follows:

- **A new technical route.** We introduce the LoRA fine-tuning technique into LLM-based password guessing, leading to the successful development of PassLLM, so far the first billion-sized parameter framework tailored for password guessing. Particularly, our proposed fine-tuning paradigm is *generic*, enabling decoder-only-based LLMs (e.g., Llama, Mistral, and Qwen), for the first time, be successfully applied to password guessing.
- **Two tailored generation algorithms.** To address the challenges posed by current LLMs in efficient and reliable password generation, we propose two generation algorithms tailored for trawling and targeted attacks, respectively. These algorithms enable large-scale, explicit password generation with regular computational resources (e.g., a consumer-grade RTX 4090 GPU).
- **Model distillation for efficient guessing.** For the first time, we introduce the model distillation technique [17] into LLM-based password guessing. By distilling a 7B-parameter model (i.e., Mistral) into a compact 0.5B-parameter one (i.e., Qwen2.5), we achieve an 11.5-*fold* speedup in trawling password generation while maintaining the guessing success rates as the original model.
- **Some insights**. We obtain a number of insights, some expected and some surprising. As expected, for LLMs with the same architecture, a larger size of parameters leads to better fine-tuning performance in password guessing. Unexpectedly, the content of the prompt has minimal impact on the guessing task results, yet it significantly outperforms the scenario where no prompt is used.

2 Background and related work

In this section, we introduce the background and developments related to password guessing, and provide an overview of the relevant LLM techniques in password security.

2.1 Trawling password models

Password guessing attacks can be categorized into trawling guessing and targeted guessing, depending on whether attackers exploit the victim's personal information.

Trawling attacks aim to guess as many passwords as possible without targeting specific individuals, constrained only by the number of guesses or available resources. In 2009, Weir et al. [46] introduced the first fully automated trawling password-guessing model based on Probabilistic Context-Free Grammar (PCFG), assuming independence between segments of a password (e.g., letters, digits, special characters). In contrast, Narayanan et al. [27] proposed the Markov model, which treats password characters as a correlated sequence, focusing on the probability of each character following a substring. Subsequent improvements to PCFG and Markov models include the incorporation of semantic patterns [40], the application of normalization and smoothing techniques [25], and the introduction of finer-grained password segmentation [50].

In 2016, Melicher et al. [26] advanced the field by applying deep learning with their FLA (Fast, Lean, and Accurate) model, which outperformed statistical models (e.g., PCFG [46] and Markov [25]), especially for large-scale guesses (e.g., $\geq 10^{10}$). In 2019, Briland et al. [18] demonstrated the potential of GANs with PassGAN, later enhanced by Pasquini et al. [32] to address training challenges like mode collapse. Additionally, Pasquini et al. introduced ADaMs [31], an adaptive rule-based model that significantly improved the efficiency of rule-based password attacks without supervision.

In 2023, Wang et al. introduced RFGuess [45], a classical machine learning-based framework that models password guessing as a multi-class classification task. Particularly, they demonstrated its effectiveness across three major guessing scenarios: trawling, PII-based, and reuse-based guessing.

In 2024, Pasquini et al. [30] introduced UNCM (Universal Neural-Cracking-Machines), a general-purpose password model that adapts to optimal guessing strategies based on context. After training, UNCM leverages auxiliary data (e.g., email addresses) to construct targeted inference models, evaluating the password strength of given groups without accessing users' plaintext passwords. Most recently, at IEEE S&P 2025, Yang and Wang [52] proposed RankGuess, a password guessing framework based on adversarial ranking, which frames the task as a Markov Decision Process and outperforms existing models in trawling, mask, and PII-based guessing scenarios.

LLMs in password security. In 2023, Xu et al. [51] proposed PassBERT, a framework based on a bidirectional Transformer. The authors first pre-trained a BERT model with the Masked Language Modeling (MLM) objective using large-scale password datasets. Then, they designed attack-specific fine-tuning approaches to tailor the pre-trained password model to three attack scenarios. Note that the maximum parameter size of PassBERT is 9,952,904, which does not reach the billion-parameter scale, a parameter threshold that has become a de facto standard for current LLMs. In the same year, Rando et al. [34] introduced an offline password model based on the GPT-2 architecture. The authors demonstrated that their proposed model, PassGPT, outperforms existing GAN-based approaches. However, the scope of their experiments was limited due to constrained GPU resources, and the model's parameter size is relatively small. Additionally, PassGPT was trained from scratch on password leaks rather than using pre-trained models, limiting its ability to leverage the nuanced language understanding inherent in pre-trained models.

In 2024, Atzori et al. [6] investigated the capabilities of LLMs in generating passwords and evaluating password strength using publicly reconstructed personal information from social networks. Their findings revealed that while the passwords generated by LLMs typically relate to the given

personal information, *they tend to be syntactically complex*, which limits their effectiveness as guessing dictionaries.

At DSN'24, Su et al. presented PagPassGPT [35], a GPT-2-based model for pattern-guided password guessing, addressing challenges in password quality and duplication. The authors introduce D&C-GEN, a divide-and-conquer algorithm to minimize duplicate passwords. Experiments show PagPassGPT outperforms existing models in success rates and reduces repeat rates in trawling and cross-site attack scenarios.

Summary. To the best of our knowledge, apart from the works by Rando et al. [34], Su et al. [35], who developed the trawling password models using the GPT-2 architecture, and Xu et al.'s BERT-based framework [51], there has been no relevant research using pre-trained generative LLMs (with billions of parameters) to guess passwords. While these studies claim to have utilized large language models, *GPT-2 is not a typical example of an LLM with emergent abilities* (i.e., the ability of large models to exhibit emergent behaviors as the model scales, allowing them to perform tasks beyond what they were explicitly trained to do). It remains unclear whether more representative LLMs, such as Llama [37], Mistral [20], and Qwen2.5 [4], could enhance the guessing success rates of leading password models (e.g., RankGuess [52], FLA [26], and Pass2Path [29]). Further, if such improvements are achievable, it is still unclear how these models can be effectively applied to a variety of major password guessing scenarios.

2.2 PII-based password models

Research on targeted password guessing is still in its early stages, with a primary focus on leveraging demographic information (e.g., name, birthday, phone, e-mail, and username) to improve guessing success rates. In 2015, Wang et al. [41] first introduced a Markov-based [25] targeted password model. The core idea is that if a certain proportion of users in a population use specific PII to create their passwords, attackers will incorporate the same proportion of PII to build their guessing dictionaries. In 2016, Li et al. [24] introduced the first targeted password model based on probabilistic context-free grammar (PCFG) [46], called Personal-PCFG. This model extends the original PCFG, which classifies passwords as Letters, Digits, and Special characters (e.g., `john@1982`→$L_4S_1D_4$), by introducing PII tags based on *length* partitioning (e.g., `john@1982`→$N_4S_1B_4$, where N_4 stands for name string of length 4, and B_4 stands for birthday string of length 4).

Later that year, Wang et al. [43] identified an issue with length-based partitioning, noting that it lacks sensitivity to the subtype of the segment. For example, both `john@1982` and `wang@1982` will be parsed into $N_4S_1B_4$, even though `wang` is a family name and `john` is a given name. To address this issue, the authors introduced *type*-based PII tags into PCFG and proposed the TarGuess-I model. For instance, N_i represents names, with N_1 for full names, N_2 for name abbreviations, and so on. In this way, the guessing success rates of TarGuess-I significantly outperform Personal-PCFG. In 2023, Wang et al. [45] proposed the RFGuess-PII password model. Like TarGuess-I [43], this model incorporates PII as type-based labels within a corresponding trawling model.

Summary. In practice, PII-based targeted password models are typically built on trawling password models. These models (e.g., TarGuess-I [43] and RFGuess-PII [45]) treat PII strings within passwords as fundamental training elements and adapt the generation strategy during the guessing phase. As a result, their performance heavily relies on the identification and matching of PII within passwords. However, the type-based PII matching approach struggles to fully capture the diverse ways in which users incorporate PII into password creation. For instance, both TarGuess-I [43] and RFGuess-PII [45] define seven types of name tags, including full names, surnames, and name abbreviations, among others (see Table 8 in the full version). Yet, variations beyond these seven forms are not covered. The same issue appears in other types of PII.

2.3 Reuse-based password models

Besides PII, attackers can also exploit users' sister passwords leaked from other accounts. These targeted attacks exploit users' tendency to reuse passwords, posing a more significant threat than those based on PII.

At NDSS'14, Das et al. [10] introduced the first rule-based algorithm for guessing reused passwords. Despite relying on heuristic rules, it significantly outperforms trawling algorithms like PCFG [46]. Later, Wang et al. [43] introduced TarGuess-II, a PCFG-based model for password reuse, which considers both structural-level modifications (e.g., L_3D_3→$L_3D_3S_2$) and character-level modifications (e.g., L_3:`abc`→L_4:`abcd`). By leveraging the concept of graph isomorphism, it maps the modification behavior of existing and new passwords to structural transformations in traditional PCFG. On this basis, the authors incorporated the PII tags defined in TarGuess-I into TarGuess-II, and proposed the TarGuess-III model, which leverages both users' PII and sister passwords to further improve the guessing success rate.

In 2019, Pal et al. [29] viewed users' password modification behavior as a sequence of atomic editing operations, such as adding, deleting, or replacing characters at individual positions. Based on this, the authors enabled the model's output to the *sequence of edits* from the sister password to the new one, ultimately constructing the Pass2Path model. Besides, the authors heuristically merged guess lists generated from the victim's two sister passwords to preliminarily explore multi-password reuse attacks.

In 2023, Wang et al. [44] noted that Pass2Path [29] does not accurately capture the impact of individual modification operations on the current password. To more precisely model users' password reuse behaviors, they adopted a multi-step decision-making mechanism and proposed a generative password model called Pass2Edit.

In the same year, Xu et al. [51] adjusted the editing operations defined in Pass2Path [29] and introduced PassBERT, a

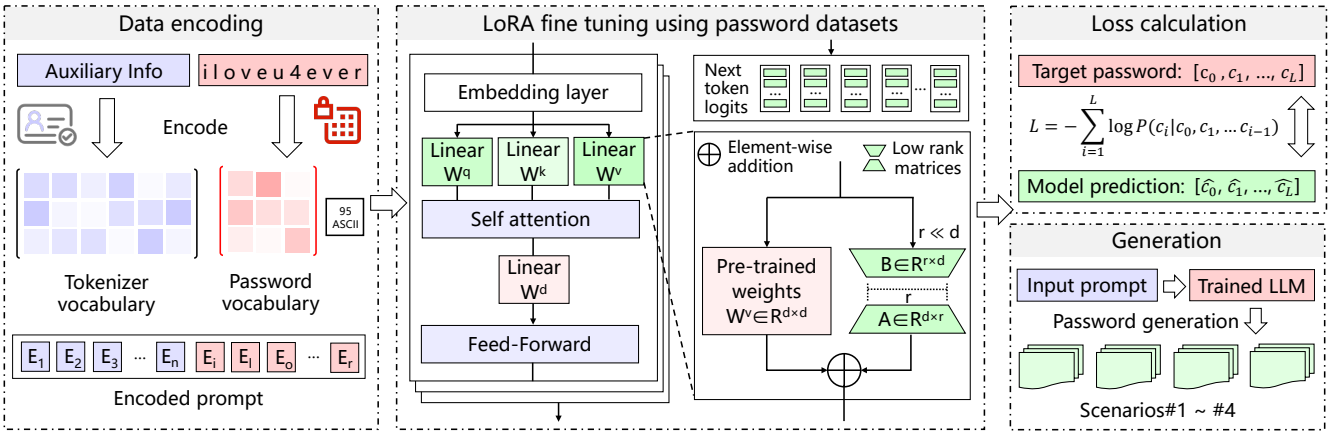


Figure 2: The PassLLM framework mainly consists of three parts. The first part is encoding, where we use different encoding vocabularies for auxiliary information (PII and/or sister passwords if available) and passwords. Then, during training, we use the LoRA fine-tuning technique. To ensure privacy security, loss computation is limited to the predicted password characters. We use our proposed dynamic beam search algorithm for efficient password generation.

password reuse model based on a bidirectional Transformer [39]. In 2024, Xiu and Wang [49] introduced PointerGuess, a Seq2Seq-based targeted password model that incorporates a pointer mechanism. Also, Xiu and Wang [49] utilized two encoders to conduct preliminary explorations of multiple password reuse scenarios. Note that this approach lacks flexibility and cannot effectively address the reality of different users leaking varying numbers of sister passwords.

Summary. Note that these studies primarily focus on a single password leaked from the victim user. Apart from a brief mention in [29] and an initial exploration in [49], there has been limited research on targeted guessing based on users' multiple leaked passwords. However, statistical analyses by Pal et al. [29] reveal that 16%-25% of users have leaked at least three passwords. Moreover, a recent report [2] highlights that, on average, each user (identified by email) has at least 2.5 leaked passwords. Therefore, evaluating the threat users encounter when multiple sister passwords are compromised is essential. This necessitates the development of new password models that can capture the intrinsic relationships among users' multiple sister passwords.

3 PassLLM: An LLM-based password guessing framework

In this section, we first present how to fine-tune large language models for the password guessing task, and then propose two efficient password generation algorithms.

3.1 Password guessing modeling

Targeted guessing attackers exploits additional auxiliary information (such as personally identifiable information and/or sister passwords) to generate a *corresponding* guessing dictionary for each target victim. From this perspective, trawling guessing can be seen as targeted guessing without any additional information, where the *same* guessing dictionary is generated for each target. Therefore, we model both the trawling guessing and targeted guessing in a unified manner.

Given a pre-trained language model $P_\Phi(y|x)$, parameterized by Φ. the password guessing task involves generating a sequence of characters for a password pw_i, based on available auxiliary information INFO_i, which may include personally identifiable information and/or users' sister passwords. The training dataset consists of sequences $\mathcal{Z} = \{\text{seq}_i\}_{i=1}^{N}$, where each sequence $\text{seq}_i = [\text{INFO}_i, \text{pw}_i]$ is composed of a user's related information followed by the corresponding password. Specifically, $\text{seq}_i = [\text{INFO}_i, \text{pw}_i] = [c_1, c_2, \ldots, c_L]$, where c_1 represents the first character in the sequence (which could be part of the auxiliary information or the start of the password), and c_L is the last character.

Fine-tuning existing large language models offers significant advantages over training from scratch, reducing both computational costs and time, while leveraging the inherent language understanding to enhance reasoning and generation performance. Accordingly, this paper adopts fine-tuning for the task of password guessing.

During fine-tuning, the language model is initialized with pre-trained weights Φ_0 and updated to $\Phi_0 + \Delta\Phi$ by following the gradient to maximize the autoregressive objective:

$$\max_{\Phi} \sum_{\text{seq} \in \mathcal{Z}} \sum_{t=1}^{L} \log P_\Phi(c_t | c_{<t}), \tag{1}$$

where c_t represents the characters in the combined sequence seq, and L is the total length of the sequence.

A drawback of full fine-tuning is that for downstream guessing tasks, we learn a different set of parameters $\Delta\Phi$, whose dimension $|\Delta\Phi|$ equals $|\Phi_0|$. For LLMs with billions of parameters, this becomes computationally expensive. To address this issue, we adopt a more parameter-efficient approach known as Low-Rank Adaptation (LoRA) [19]. In this light, the task-specific parameter update $\Delta\Phi = \Delta\Phi(\Theta)$ is represented by a smaller set of parameters Θ, with $|\Theta| \ll |\Phi_0|$. The task of fine-tuning is formulated as an optimization over Θ:

$$\max_{\Theta} \sum_{\text{seq} \in \mathcal{Z}} \sum_{t=1}^{L} \log P_{\Phi_0 + \Delta\Phi(\Theta)}(c_t | c_{<t}) \tag{2}$$

Currently, most existing LLMs are built on the Transformer architecture [39]. A key component of the Transformer is the attention mechanism, which enables the model to focus on different parts of the input sequence when making predictions. We employ LoRA [19] to modify the attention mechanism of the model by applying low-rank updates to the projection matrices of query, key, and value in the self-attention layers. Specifically, instead of updating the full query projection matrix W_q, we decompose it into:

$$W_q = W_q^0 + \Delta W_q, \tag{3}$$

where W_q^0 is the pre-trained query projection matrix (i.e., does not receive gradient updates), and ΔW_q is the task-specific (i.e., password guessing) update, parameterized as:

$$\Delta W_q = A_q B_q, \tag{4}$$

with $A_q \in \mathbb{R}^{d \times r}$ and $B_q \in \mathbb{R}^{r \times k}$, where $r \ll \min(d,k)$. The matrices A_q and B_q are the low-rank components introduced during fine-tuning, significantly reducing the number of trainable parameters and making fine-tuning computationally efficient.

The same low-rank adaptation is applied to the projection matrices of the key and value, allowing the model to specialize in the password generation task without updating all parameters. The loss function used during training is the cross-entropy loss, defined as:

$$\mathcal{L} = -\sum_{i=k}^{L} \log P(c_i | c_0, c_1, \ldots, c_k, c_{k+1}, \ldots, c_{k+i-1}) \tag{5}$$

where c_k represents the first password character. This objective encourages the model to generate password sequences that are contextually dependent on preceding password characters and any available auxiliary information (PII and/or sister passwords). Notably, while typical training processes calculate the loss across the entire input sequence, our approach deviates to address specific security concerns.

In our task, the prefix section typically includes sensitive information, such as PII and/or sister passwords. Optimizing the loss over the entire sequence could expose the model to potential vulnerabilities, enabling malicious attackers to craft adversarial prompts that extract portions of users' PII or previously leaked passwords. To mitigate this risk, we restrict the loss calculation to the password prediction portion, ensuring that sensitive input content is excluded from direct optimization during training.

During model training, it's necessary to encode the training data into numerical representations. To achieve this, we employ *separate vocabularies* for encoding passwords and prefixes (i.e., prompt and available auxiliary information). For prefixes, we use the default vocabulary of the large language model. While for passwords, we employ a character-level encoding. This separation aligns with the inherent characteristics of the task: PASSLLM is designed *exclusively* for password generation, which is inherently a character-level language modeling task. Given that over 99% of real-world passwords are composed of 95 printable ASCII characters (see Table 1), limiting the password vocabulary to this character set is a natural and task-aligned design choice.

More specifically, we extract these 95 characters from the model's original vocabulary and add an EOS token (end of the sequence being generated), resulting in a 96-symbol password vocabulary used for character-to-index encoding.

Algorithm 1: The two-stage password generation algorithm for trawling guessing

Input: Prompt p_0, maximum prefix count N, probability threshold τ, large language model M, Vocabulary V
Output: Finished password collection C_{finished}

Stage 1: Prefix generation
PQ ← PriorityQueue(); /* Initialize priority queue */
$C_{\text{finished}} \leftarrow \emptyset$; /*Store finished passwords in two stages*/
PQ.push("", 1);
while $|PQ| < N$ **do**
  $(s, \Pr(s)) \leftarrow$ PQ.pop(); /* Pop the highest probability prefix */
  if $s =$ "" **then**
    $C_{\text{next}}, \text{Cache}_{\text{prompt}} \leftarrow \text{Predict}(M, p_0, V)$ /*C_{next} contains next tokens and their probabilities*/
  else
    $C_{\text{next}}, \_ \leftarrow \text{Predict}(M, s, \text{Cache}_{\text{prompt}}, V)$
  $C_{\text{candidates}} \leftarrow \text{ConcatAndUpdate}(s, C_{\text{next}})$ /*Concatenate the prefix and next tokens, update cumulative probabilities*/
  for *each* $c \in C_{candidates}$ **do**
    if *c contains EOS and Pr(c)*>τ **then**
      $C_{\text{finished}} \leftarrow C_{\text{finished}} \cup \{c\}$
    else
      PQ.push($c, Pr(c)$)
Stage 2: Password generation
for *each* $(prefix, Pr(prefix)) \in PQ$ **do**
  Q ← Queue(); /* Initialize queue for password generation */
  Q.enqueue(prefix, Pr(prefix)); /* Start from prefix with its probability */
  seed ← p_0 || prefix /* Concatenate the prefix with the prompt */
  while *Q is not empty* **do**
    $(s, \Pr(s)) \leftarrow$ Q.dequeue()
    if $s = prefix$ **then**
      $C_{\text{next}}, \text{Cache}_{\text{seed}} \leftarrow \text{Predict}(M, \text{seed}, V)$
    else
      $C_{\text{next}}, \_ \leftarrow \text{Predict}(M, s, \text{Cache}_{\text{seed}}, V)$
    $C_{\text{candidates}} \leftarrow \text{ConcatAndUpdate}(s, C_{\text{next}})$
    for *each* $c \in C_{candidates}$ **do**
      if $Pr(c) \geq \tau$ **then**
        if *c contains EOS* **then**
          $C_{\text{finished}} \leftarrow C_{\text{finished}} \cup \{c\}$;
        else
          Q.enqueue($c, \Pr(c)$)
return C_{finished}

3.2 Password generation algorithms

In practice, password generation encompasses two distinct scenarios with fundamentally different characteristics. Specifically, trawling guessing aims to crack passwords across a wide user population (without user-specific information) by generating a large-scale (e.g., 10^8), generic password dictio-

nary. In contrast, targeted guessing exploits the victim's PII and/or sister passwords to produce a small set (e.g., 10^3) of personalized/targeted guesses. Given the significant differences in guessing scale and information exploited, we design two dedicated and efficient password generation algorithms tailored to trawling and targeted guessing, respectively.

3.2.1 Trawling password generation algorithms

For trawling guessing scenarios, we employ a Breadth-First Search (BFS) algorithm with probability threshold pruning (see Appendix E in the full version). This approach is widely adopted in existing trawling password models [26, 52]. Given the vast parameter size of PASSLLM (e.g., 7B), we propose a two-stage password generation algorithm to further improve generation efficiency by facilitating effective parallelization of large-scale password generation (see Algorithm 1).

In the first stage, a *priority* queue is employed to generate a set of non-overlapping prefixes (e.g., `1234`, `zxcv`, `ilo`) based on the initial prompt p_0. Specifically, we begin by generating 95 single-character prefixes (here, our vocabulary V consists of 95 printable ASCII characters and an end-of-sequence token). These prefixes are inserted into a max-heap priority queue. At each iteration, the prefix with the highest cumulative probability (e.g., `a`) is dequeued and used to predict the next-token probability distribution. The prefix is then expanded by appending each of the 95 printable ASCII characters to form new prefixes (e.g., `aa`, `ab`, ..., `a#`), whose cumulative probabilities are updated accordingly. For example, after two iterations, the queue contains 189 (=95−1+95) prefixes. This process repeats until the queue exceeds a predefined size.

It is important to note that the generated prefixes are of *variable lengths* and form a prefix-free structure in which no sequence is a prefix of another (e.g., prefixes like `abc` and `abcd` do not coexist in the queue). Besides, these prefixes are not necessarily the top-ranked by probability; rather, they cover all possible password generation paths. The use of a priority queue is motivated by the desire to balance the number of generated passwords across different prefixes in the second stage (mitigating the situation where high-probability prefixes lead to extremely large sub-dictionaries while others yield very small ones). During the prefix generation process, if the cumulative probability of a prefix concatenated with the EOS (end of the sequence being generated) token exceeds a threshold τ, it is added to the final set of generated passwords.

In the second stage, each prefix is treated as an independent seed. It is concatenated with the same initial prompt p_0 to form a new input sequence, which is then used to generate sub-dictionaries via the BFS algorithm with probability threshold pruning (see Stage 2 in Algorithm 1). Once all sub-dictionaries are generated, they are merged and ranked by their probabilities to produce the final password dictionary. This approach naturally decomposes the password generation task into multiple independent subtasks, enabling efficient large-scale parallelization of dictionary generation.

Algorithm 2: Dynamic beam search for targeted guessing

Input: Language model M; INFO (i.e., PII and/or sister passwords); Beam width list $K[1{:}m]$; Batch size B; EOS threshold ε; Vocabulary V
Output: Finished password collection C_{finished}

$C_{\text{finished}} \leftarrow \emptyset$ /* Store finished candidates with EOS */
$C_{\text{beam}} \leftarrow \emptyset$ /* Store prefixes and their cumulative probabilities in current beam */
$\text{Cache}_{\text{pw\_prev}} \leftarrow \emptyset$ /* Store password KV cache from last depth */
$C_{\text{next}}, \text{Cache}_{\text{INFO}} \leftarrow \text{Predict}(M, \text{INFO}, V)$ /* C_{next} contains next tokens and their probabilities */
for $i = 1$ **to** m **do**
  $C_{\text{candidates}} \leftarrow \text{ConcatAndUpdate}(C_{\text{beam}}, C_{\text{next}})$ /*Concatenate prefixes and next tokens, update cumulative probabilities*/
  $C_{\text{finished}} \leftarrow C_{\text{finished}} \cup \{s \in C_{\text{candidates}} | s[-1]{=}\text{EOS}, \Pr(\text{EOS}|\cdot) > \varepsilon\}$ /* Filter finished sequences based on the EOS probability */
  $C_{\text{unfinished}} \leftarrow \{s \in C_{\text{candidates}} \mid s[-1] \neq \text{EOS}\}$
  $C_{\text{beam}} \leftarrow \text{SelectTopK}(C_{\text{unfinished}}, K[i])$
  $\{B_1, \ldots, B_n\} \leftarrow \text{PartitionIntoBatches}(C_{\text{beam}}, B)$ /* Partition beam into n small batches */
  $C_{\text{next}} \leftarrow \emptyset$
  $\text{Cache}_{\text{pw}} \leftarrow \emptyset$ /* Store current depth password KV cache */
  for $j = 1$ **to** n **do**
    $\text{B\_cache}_{\text{pw}} \leftarrow \text{SelectPasswordCache}(B_j, \text{Cache}_{\text{pw\_prev}})$
    /* Select B_j's password KV cache from i-1 depth */
    $\text{B\_cache} \leftarrow \text{Concat}(\text{Repeat}(\text{Cache}_{\text{INFO}}, |B_j|), \text{B\_cache}_{\text{pw}})$
    /*Concatenate INFO KV cache with password KV cache*/
    $\text{B\_}C_{\text{next}}, \text{B\_cache} \leftarrow \text{Predict}(M, B_j, \text{B\_cache}, V)$
    $\text{B\_cache}_{\text{pw}} \leftarrow \text{ExtractPasswordCache}(\text{B\_cache})$ /* Extract password KV cache*/
    $\text{Cache}_{\text{pw}} \leftarrow \text{Cache}_{\text{pw}} \cup \text{B\_cache}_{\text{pw}}$ /* Save the newly generated password KV cache */
    $C_{\text{next}} \leftarrow C_{\text{next}} \cup \text{B\_}C_{\text{next}}$
  $\text{Cache}_{\text{pw\_prev}} \leftarrow \text{Cache}_{\text{pw}}$ /* Update password KV cache */
return C_{finished}

3.2.2 Targeted password generation algorithms

In targeted guessing scenarios, the guess number allowed is typically limited (e.g., NIST-recommended rate limiting $\leq$100 failed attempts per account [15]), making exhaustive search strategies such as BFS (even with probability threshold pruning) computationally inefficient (in the context of LLM-based password generation). A natural alternative is to employ the beam search algorithm, which prioritizes the generation of the top-k high-probability password candidates by pruning less promising paths [29, 44, 49]. However, the naive implementation of beam search incurs substantial VRAM consumption when scaled to large beam widths (e.g., 1,000). To mitigate this, we propose a dynamic beam search strategy (see Fig. 3 and Algorithm 2) that allocates beam width during generation, achieving a better balance between efficiency and performance. The algorithm takes the following inputs:

- m: the maximum depth of generation
- $K[i]$: the beam width at i-th depths
- V: the vocabulary used for password generation
- B: the batch size during the model prediction
- ε: the threshold for the end-of-sequence symbol

At each depth i, the algorithm selects the top $K[i]$ unfinished password prefixes from depth $i-1$ based on their cumulative

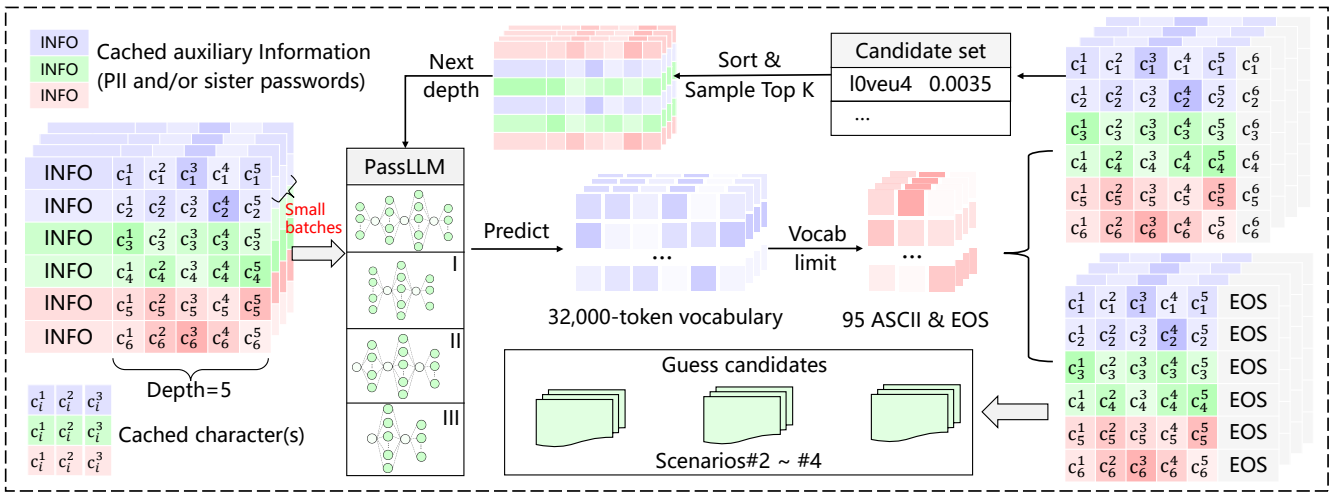


Figure 3: Dynamic beam search algorithm of PassLLM. At each prediction step, the original 32,000-token vocabulary is limited to 95 printable characters and an EOS symbol. Sequences containing EOS are then placed into the guess set, and from the remaining sequences, the Top-k candidates are selected and sorted based on their probabilities. This process repeats for subsequent prediction steps until the predefined maximum depth is reached.

log-probabilities. Due to memory constraints, these candidates are divided into $n = \max(\lceil K[i]/B \rceil, 1)$ smaller batches, where B denotes the maximum number of sequences that can be processed in parallel on a given GPU (e.g., NVIDIA 4090).

Each small batch undergoes a single forward pass (inner-forward), producing unfinished candidate sequences for the next depth. Unlike traditional beam search, we do not treat the EOS token as a normal candidate in beam ranking. Instead, we introduce a termination threshold mechanism: a sequence is forcibly ended and added to the output set only if the model's predicted $\Pr(\text{EOS}|\cdot) > \varepsilon$ for that sequence. This design addresses a specific observation in our targeted guessing scenarios: certain prefix sequences closely related to auxiliary information (e.g., `Smit` as a partial form of `Smith`) tend to have high prefix probabilities. As a result, even when $\Pr(\text{EOS}|\texttt{Smit})$ is low, the overall probability $\Pr(\texttt{Smit[EOS]})$ can still be relatively high (e.g., compared to $\Pr(\texttt{SmithWill1968[EOS]})$), potentially resulting in the inclusion of such incomplete/unrealistic passwords in the Top-1,000 guess list, which do not reflect users' actual password creation behaviors. Our threshold strategy mitigates this risk.

After all n batches are processed at depth i, the algorithm selects the top $K[i+1]$ candidates for the next depth, until reaching the maximum depth m. All completed sequences (i.e., terminated with EOS) are finally sorted and returned in descending order of their cumulative log-probabilities.

KV cache. To reduce redundant computation in the autoregressive decoding process, we adopt key-value (KV) caching to store intermediate attention states. However, using the default beam search (e.g., as implemented in Hugging Face Transformers) incurs substantial memory overhead, as each beam *separately stores full KV caches for both the PII and the evolving password prefix*. This leads to severe GPU memory exhaustion, especially under large beam widths. To mitigate this, we perform a one-time forward pass on the INFO to obtain its corresponding KV cache, denoted as $\text{Cache}_{\text{INFO}}$. This KV cache is shared across all prefixes during generation.

As a result, for each target, we only need to maintain batch_size instances of $\text{Cache}_{\text{INFO}}$, instead of the beam_width instances required by conventional beam search. Particularly, batch_size can be flexibly adjusted based on available VRAM without affecting the guessing success rates under a fixed beam_width. As for the password prefix, we still maintain separate beam_width KV caches. Since a single KV cache for INFO is typically much larger than that for the password prefix, our design substantially reduces VRAM consumption compared to conventional beam search, which stores beam_width instances for both components.

4 Experiments

We first elaborate on the experimental setup, and then fairly/comparatively evaluate our PassLLM(-I∼III) with its foremost counterparts (i.e., RankGuess [52], RFGuess [45], PassGPT [34], PagPassGPT [35], PCFG [46], Markov [25], FLA [47], TarGuess [43], PointerGuess [49], Pass2Edit [44], Pass2Path [29] and PassBERT [51]).

4.1 Our datasets

Datasets. Our evaluation is based on 11 large-scale password datasets with over 3.3 billion plaintext passwords, including six from English sites, four from Chinese sites and one mixed (see Tables 1 and 2). Among them, COMB (Compilation of Many Breaches) is a consolidated archive of over 3.28 billion plain text email-password pairs tied to 2.18 billion unique emails. It draws on credentials exposed in some 252 previous breaches (including Netflix, LinkedIn, Yahoo, Bitcoin and Hotmail) and, despite containing no new leaks, represents one of the largest single compilations ever released. Notably, it includes over 1.5 million records from government domains

Table 1: Basic information about our eight password datasets (PW=password).

Dataset	Language	Service type	Leaked time	Original size	Non-PWs/Non-ASCII	PW length>30	Removed	After cleaning	Remaining
Post Millennial	English	News outlet	May 2024	38,902	0	31	0.08%	38,871	99.92%
000Webhost	English	Web hosting	Oct. 2015	15,299,907	69,606	1,131,721	7.76%	14,098,263	92.24%
Rockyou	English	Social forum	Dec. 2009	32,603,387	149,971	2,068	0.47%	32,451,348	99.53%
Taobao	Chinese	E-commerce	Feb. 2016	15,072,418	0	86	0.00%	15,072,332	100.00%
126	Chinese	Email	Dec. 2011	6,392,568	0	599	0.00%	6,391,969	100.00%
CSDN	Chinese	Programmer forum	Dec. 2011	6,426,872	0	125	0.00%	6,428,285	100.00%
Dodonew	Chinese	E-commerce	Dec. 2011	16,282,276	4,975	12,070	0.10%	16,265,231	99.90%
COMB	Mixed	Mixed	Feb. 2021	3,279,064,312	14,948,181	10,576,452	0.78%	3,253,539,679	99.22%

Table 2: Basic information of our PII datasets.

Dataset	Language	Items num	Types of PII useful for this work
12306-PII	Chinese	129,303	Email, User name, Name, Birthday, Phone
Dodonew-PII	Chinese	161,517	Email, User name, Name, Birthday, Phone
CSDN-PII	Chinese	77,216	Email, User name, Name, Birthday, Phone
000Webhost-PII	English	79,580	Email, User name, Name, Birthday
Rootkit-PII	English	69,418	Email, User name, Name, Birthday
Clixsense-PII	English	2,222,045	Email, User name, Name, Birthday

(e.g., 625,505 U.S. `.gov`, 205,099 U.K. `.gov.uk`) and roughly 450 million Yahoo and 200 million Gmail accounts [1].

12306, Rootkit, and ClixSense are associated with various kinds of PIIs (e.g., name, email, username, phone, and birthday). For targeted password guessing, we obtain another two PII datasets by matching the email (see Table 2). Besides some early disclosed datasets (i.e., CSDN, Dodonew, Rockyou, and 000Webhost) which have been widely used in research [25, 26, 42, 43], we incorporate one very recently leaked dataset (i.e., the Post Millennial [5]) that may exhibit up-to-date users' password behaviors. These datasets are compromised by hackers or leaked by insiders, and have been publicly available on the Internet for some time. Following the data cleaning method in [25,42,52], we remove non-password strings in the original dataset, including headers, descriptions, footnotes, and non-ASCII strings. We also remove passwords with length>30, because these overly long strings do not seem to be generated by users, but more likely by password managers or simply junk information.

4.2 Key factors evaluation

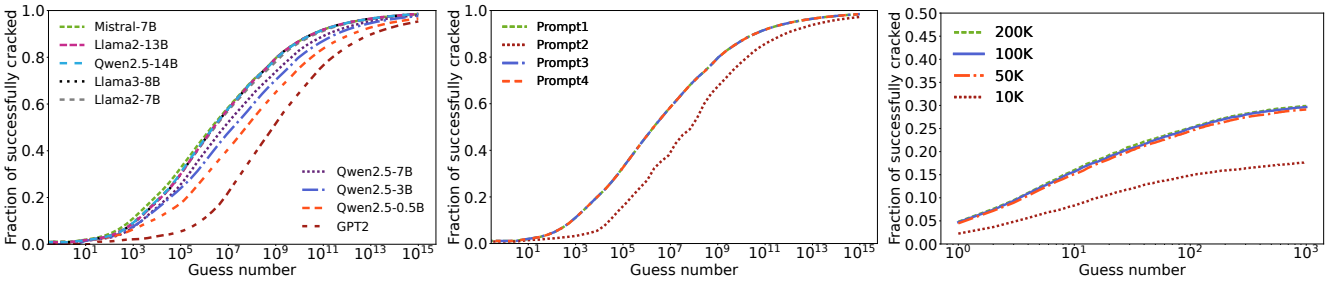
Choice of fundamental model. Currently, there is a wide variety of general-purpose large language models based on different architectures, including encoder-decoder models (e.g., T5 [33]), decoder-only models (e.g., Mistral [20] and Llama [37]), and encoder-only models (e.g., BERT [12]). Given that password guessing is fundamentally a character-level language modeling problem, where passwords are typically generated autoregressively, encoder-only architectures are unsuitable for this task. Between encoder-decoder and decoder-only architectures, we choose the decoder-only fundamental model for fine-tuning, as it avoids the additional computational overhead associated with unnecessary encoders.

More specifically, we randomly select one million Rockyou passwords and fine-tune a number of large language models (using 100k randomly chosen from the remaining Rockyou passwords as the test set), including Llama2-7B[3], Llama3-8B[4], Llama2-13B[5], GPT-2[6], Qwen2.5-0.5B[7], Qwen2.5-3B[8], Qwen2.5-7B[9], Qwen2.5-14B[10], and Mintral-7B[11]. Particularly, the LoRA trainable parameters for all large language models are detailed in Fig. 13 (the "PassLLM setting").

Fig. 4(a) shows that, when the model parameters are comparable, LLMs primarily trained on English corpora (e.g., Llama and Mistral) show higher guessing success rates than Qwen models trained mainly on Chinese corpora. In particular, the performance of Llama2-7B/Mistral-7B is comparable to that of Qwen-14B. Among LLMs primarily trained on English corpora, the differences are minimal, and increasing the model's parameter size (e.g., from Llama2-7B to Llama2-13B) does not significantly improve the guessing success rate. Interestingly, while Llama3-8B outperforms Mistral-7B/Llama2-7B on various benchmark tasks[12], their performance on password guessing is comparable. This highlights the unique nature of password text, indicating that password guessing requires specific fine-tuning. Overall, Mistral-7B [20] performs the best. Therefore, without loss of generality, we take Mistral-7B [20] as a representative case study to demonstrate how to fine-tune LLMs for password guessing.

Impact of prompt. Before conducting large-scale experiments, we evaluate the impact of four distinct prompts on the overall guessing effectiveness. The first prompt explicitly positions the LLM as a trawling password model, guiding it to generate passwords. The second prompt removes all instructions, using only the training passwords to evaluate the model's baseline performance without guidance. The third prompt shifts the model's role to a password defense advisor, aimed at preventing password guessing, and exploring the impact of an adversarial or defensive prompt. Finally, the fourth prompt, set within the unrelated domain of chemistry, serves as a control to evaluate the effect of irrelevant instructions on password guessing. This comparison helps evaluate how different prompts influence password prediction performance.

[3] https://huggingface.co/meta-Llama/Llama-2-7b
[4] https://huggingface.co/meta-Llama/Meta-Llama-3-8B
[5] https://huggingface.co/meta-Llama/Llama-2-13b
[6] https://huggingface.co/openai-community/gpt2
[7] https://huggingface.co/Qwen/Qwen2.5-0.5B
[8] https://huggingface.co/Qwen/Qwen2.5-3B
[9] https://huggingface.co/Qwen/Qwen2.5-7B
[10] https://huggingface.co/Qwen/Qwen2.5-14B
[11] https://huggingface.co/mistralai/Mistral-7B-v0.1
[12] https://ai.meta.com/blog/meta-Llama-3/



(a) The impact of base models (b) The impact of prompts (c) The impact of training set size

Figure 4: Evaluation of three key factors (i.e., base model, prompts, and training set size) affecting the model's overall performance. Fig. 4(a) demonstrates that using Mistral-7B as the base model results in a slightly higher guessing success rate compared to other alternative large language models. Fig. 4(b) illustrates that prompts with different roles (see Table 8) have minimal impact on the model's guessing success rate, but they all significantly outperform the no-prompt condition. Fig. 4(c) shows that the model's guessing success rate in the PII-based targeted scenario stabilizes once the fine-tuning dataset size reaches 50,000.

Table 3: Trawling guessing scenarios

Scenario #	Training set	Training set size	Test set	Test set size
1	80% Rockyou	1,000,000	20% Rockyou	100k
2			000Webhost	
3	80% Dodonew	12,955,973	20% Dodonew	
4			Taobao	

Table 8 presents the detailed prompts.

Surprisingly, assigning different roles to the LLMs has minimal impact on the final guessing success rate (see Fig. 4(b)). Using a prompt, even one that is completely unrelated or contrary, significantly outperforms not using any prompt. This phenomenon may stem from the model's ability to utilize any context, regardless of its relevance, to guide its predictions. It suggests that the presence of structure, even in the form of an unrelated or adversarial prompt, helps the model focus on the task at hand. Given that the difference in performance is minimal, we opt for a simple and straightforward prompt format for our study: the first prompt (In targeted guessing scenarios, our prompt is adjusted to "As a targeted password guessing model, your task is to utilize the provided information to guess the corresponding password. \n Auxiliary information sequence+password").

4.3 Trawling attack scenarios

Experimental setup. The most recent advancement in password guessing research is the RankGuess model, proposed by Yang and Wang in their IEEE S&P 2025 paper [52]. In this work, we follow their experimental setup by randomly selecting one million samples from 80% of the Rockyou dataset for training, and using a randomly chosen subset of 100,000 passwords from the remaining 20% as the test set. This is common practice since previous research [11,44] demonstrates that the password model's guessing success rate converges once the test set size reaches 10,000 entries.

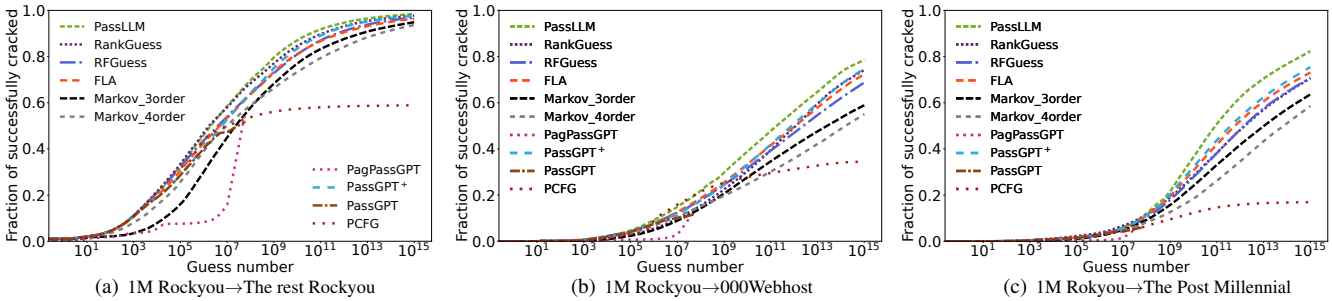
To evaluate the fitting and generalization abilities of PassLLM, we conduct both one-site and cross-site password guessing experiments (i.e., 1M Rockyou→20% Rockyou and 1M Rockyou→000Webhost, where M=million). Considering that some password models (e.g., FLA [26]) may perform better with large-scale training sets (e.g., over ten million samples), we also compare the performance of various models using the 80% Dodonew dataset (a Chinese dataset, to demonstrate the generalizability of our model). The trawling guessing scenarios are summarized in Table 3. The descriptions and parameter settings for the seven leading comparative models (i.e., RankGuess [52], RFGuess [45], PassGPT [34], PagPassGPT [35], PCFG [46], 3/4-order Markov [25], and FLA [47]) can be seen in Appendix A.

Experimental results. Due to the computational intensity of enumerating all possible guesses in trawling guessing scenarios, we use the Monte Carlo algorithm [11] to approximate the number of attempts required for an attacker to successfully crack a password. A key requirement for applying this method to PassLLM is that the generated password samples must be conditioned on a fixed prompt (see Appendix D for details). This ensures that the Monte Carlo estimate of the guess number for a given password corresponds to its expected rank under PassLLM. (see Fig. 10(h)).

For PassGPT [34] and PagPassGPT [35], we run the source code with the best parameters recommended in their papers (see Appendix A.1 for detailed settings) and set the total dictionary size to 10^8. Both models ultimately generate approximately 6×10^7 unique passwords.

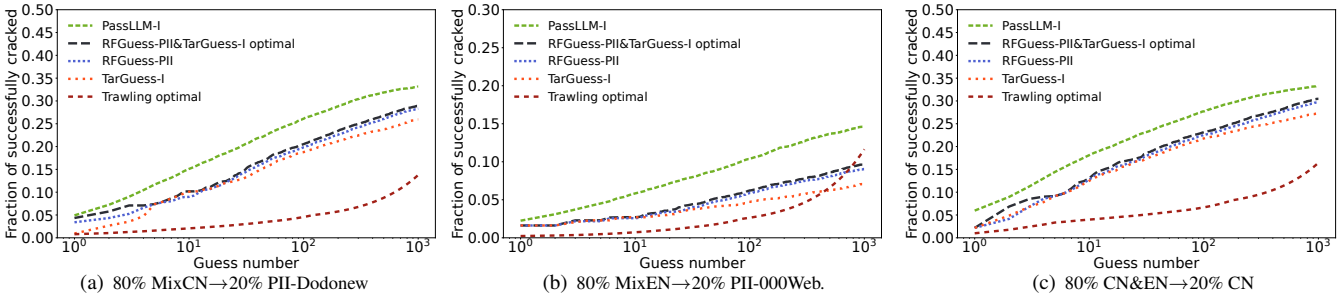
We notice that PassGPT [34] is essentially a probabilistic model; however, in the publicly released code[13], the authors directly invoke the `generate` function in `GPT2LMHeadModel`, which returns random sampled outputs without associated probabilities. These sampled passwords are then used as the final generated dictionary in the original paper. As a result, the dictionary contains a high proportion of duplicate passwords (around 40% under a 10^9 guessing budget), and lacks probability scores, making it infeasible to effectively rank the generated guesses. To address this, we apply the Monte Carlo sampling method used for our PassLLM (see Appendix D) to PassGPT [34], enabling it to assign probabilities to the generated passwords. This modification significantly improves

[13] https://github.com/javirandor/passgpt/blob/main/src/generate_passwords.py



(a) 1M Rockyou→The rest Rockyou (b) 1M Rockyou→000Webhost (c) 1M Rokyou→The Post Millennial

Figure 5: Comparison of the performance of our PASSLLM with leading trawling password models (i.e., RankGuess [52], PassGPT [34], PagPassGPT [35], RFGuess [45], PCFG [46], 3/4-order Markov [25], and FLA [26]) across one-site and cross-site guessing scenarios. Here, → indicates that the attacker uses the data on the left to train the model and the data on the right as the corresponding test set. Overall, our PASSLLM either matches or outperforms all its counterparts.



(a) 80% MixCN→20% PII-Dodonew (b) 80% MixEN→20% PII-000Web. (c) 80% CN&EN→20% CN

Figure 6: Experimental results of our PASSLLM-I against TarGuess-I [43] and RFGuess-PII [45] in PII-based targeted guessing scenarios (see Fig. 10 for the other four scenarios). The guessing success rates of our PASSLLM-I are always the best within 10^3 guesses. CN=Chinese; EN=English; 000Web.=000Webhost.

the model's guessing success rate compared to the original version, as shown in the PassGPT$^+$ result curve in Fig. 5. In contrast, the PagPassGPT model [35] introduces a pattern-guided guessing mechanism, which breaks the underlying probabilistic nature of the model. Therefore, Monte Carlo simulation is not applicable in this case.

The results (see Fig. 5) show that within 10^{14} guesses, our PASSLLM achieves a guessing success rate that matches or outperforms all existing password models. In particular, PASSLLM demonstrates strong generalization ability in cross-site guessing scenarios. For example, on the 2024 leaked Post Millennial dataset (which is unlikely to appear in the LLM's pretraining data due to its recency), PASSLLM significantly outperforms the state-of-the-art RankGuess model [52] by 3.36%-32.21% at 10^9 guesses. We further verified that the overlap between the Post Millennial and our training set (i.e., 1M Rockyou) is minimal, with a match rate of only 2.42%.

Improvement principles. Apart from PCFG [46], all the password models compared, including our PASSLLM, are based on the Markov principle from a high-level perspective. Specifically, they predict the probability distribution of the next character using preceding character(s) as context. The performance difference between PASSLLM and other password models can be attributed to two main factors. First, the sufficiently large parameter size of PASSLLM (i.e., 7B) allows it to more accurately capture the character probability distribution of a given password dataset. Second, LLMs (e.g., the Mistral base model in PASSLLM) are pretrained on vast amounts of natural language text. While there are significant differences between natural language and password characters, both are fundamentally textual, which gives LLMs superior contextual learning ability for password characters compared to traditional password models. Further, we analyze the overlap among the passwords cracked by the top three performing models (i.e., our PASSLLM, RankGuess [52], and PassGPT [34]). Detailed results are provided in Appendix B.

4.4 PII-based attack scenarios

Considering that the datasets containing PII are typically small (less than one million), we first evaluate the impact of training set size on the performance of PASSLLM-I. Specifically, we combine three Chinese datasets (i.e., PII-12306, PII-Dodonew, and PII-CSDN) and three English datasets (i.e., PII-Rootkit, PII-000Webhost, and PII-ClixSense) to create two larger mixed datasets. Each mixed dataset was then split into an 80% training set and a 20% test set. From the 80% training set, we randomly select samples of 10,000, 50,000, 100,000, and 200,000 entries as training subsets. Meanwhile, a fixed subset of 10,000 entries was randomly chosen from the remaining 20% to serve as the test set.

Figure 4(c) shows a comparison of PASSLLM-I's guessing success rates under different training set sizes. Results show that the guessing success rate increases with the number of fine-tuning samples, and remains stable once the fine-tuning dataset reaches 50,000 entries. In particular, for all targeted guessing scenarios (including PII-based, reuse-based, and multi source-based), the results are derived from actual password dictionaries explicitly generated for each target user

Table 4: Experimental setup of PII-based guessing scenarios†

Scenario #	Training set	Composition	Training set size	Test set (size)
1	MixCN 80%	80% Dodonew	231,739	Dodonew 10k
2		80% CSDN		CSDN 10k
3		80% 12306		12306 10k
4	MixEN 80%	100k ClixSense	274,845	ClixSense 10k
5		80% Rookit		Rookit 10k
6		80% 000Web.		000Web. 10k
7	MixCN&EN	MixCN 80%	535,604	MixCN 10k
8		MixEN 80%		MixEN 10k

†All datasets contain PII. CN/EN=Chinese/English; 000Web.=000Webhost

Table 5: Reuse-based guessing scenarios

#	Experimental setup	Training set (size)	Test set (size)
1	$pw_1 \rightarrow pw_2$	COMB (|pw|=2) 100k	COMB (|pw|=2) 10k
2	$pw_1 \rightarrow pw_2$	126→Dodonew 188,926	126→CSDN 10k
3	$(pw_1, pw_2) \rightarrow pw_3$†	COMB (|pw|=3) 100k	COMB (|pw|=3) 10k
4	$pw_1 \sim pw_n \rightarrow pw_{n+1}$, $n \in [3,7]$	COMB (|pw|∈[3,9]) 100k	COMB (|pw|∈[3,9]) 10k

†The notion $(pw_1, pw_2) \rightarrow pw_3$ means using *two* sister passwords of the same user to crack the user's new password on another website.

(rather than using the Monte Carlo simulation).

Experimental setup. We compare PASSLLM-I with two leading PII-based targeted password models (i.e., TarGuess-I [43] and RFGuess-PII [45]). Table 4 presents our eight PII-based guessing scenarios. For each model, we use the open-source code provided by the authors and configure the parameters according to their recommended settings. Additionally, we utilize the Min_auto strategy [38] to avoid the bias of a single model (i.e., Combine TarGuess-I [43] and RFGuess-PII [45] to obtain the optimal result). The setup for each model can be seen in Appendix A.

Experimental results. We set the maximum guess number to 1,000 based on established practices in mainstream targeted guessing literature (e.g., [29, 43, 45]). Fig. 6 (and Fig. 10) shows that, within 100 guesses, PASSLLM-I outperforms both TarGuess-I [43] and RFGuess-PII [45] by 16.20%-119.87% and 15.10%-76.27%, respectively. This demonstrates that our PASSLLM-I is more effective at modeling users' vulnerable behaviors of creating passwords based on their PIIs. To clearly illustrate the advantages of PASSLLM-I, we conduct an overlap analysis of the passwords successfully guessed by the three models (see Appendix B).

4.5 Reuse attack scenarios

Experimental setup. To evaluate the guessing capabilities of PASSLLM-II in password reuse attack scenarios, we construct several password reuse datasets. Specifically, following the method employed in [29, 44, 49, 51], we match email addresses with the COMB dataset to create a large-scale dataset that contains email addresses along with multiple passwords. For each user with two or more leaked/sister passwords, we randomly select one password as the target. We then establish scenarios for single-password reuse and multi-password reuse based on the number of sister passwords. In the multi-password reuse scenarios, we distinguish between users with two leaked passwords and those with 3-9 leaked passwords (see Table 5). Particularly, in each attack scenario, we randomly select 100,000 samples from the dataset as the training set, and additionally choose 10,000 samples as the test set. For the multi-password attack scenarios (i.e., Scenario #4 and the following scenarios in Sec. 4.6), we present the distribution of the number of sister passwords per user in the dataset (see Fig. 12 in the full version `https://bit.ly/43Woqk5`).

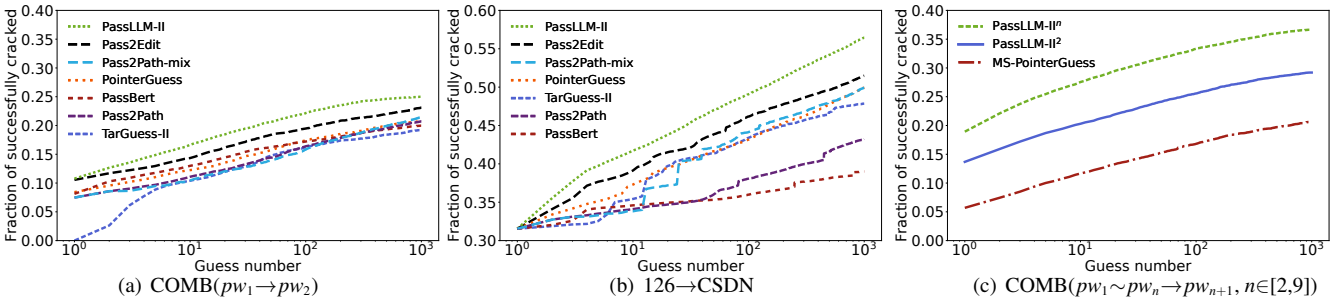
We compare our PASSLLM-II with six state-of-the-art password reuse models (i.e., TarGuess-II [43], Pass2Path [29], Pass2Edit [44], PassBERT [51], PointerGuess [49], and MS-PointerGuess [49]). We briefly describe these password models in Appendix A. Particularly, we use the best configuration parameters recommended by the authors. For our PASSLLM framework, we only need to adjust the input data from PII to (multiple) sister passwords.

Experimental results. The experimental results (see Fig. 7) show that our PASSLLM-II outperforms all password reuse models in the single-password reuse scenario. More specifically, within 1,000 guesses, its guessing success rate outperforms that of the best-performing Pass2Edit [44] by 8.27%-9.66%. A plausible reason is that during training, Pass2Edit [44] only focuses on password pairs with a cosine similarity greater than 0.3, overlooking those with smaller cosine similarities but still complying with reuse behaviors (e.g., the cosine similarity between `johndoe` and `JOHNDOE` is 0, but they are reused password pair; see more examples in Appendix C of the full version). In contrast, our PASSLLM-II avoids this issue by learning directly from the entire password pair dataset without requiring any filtering of the training set.

In the two-password reuse attack scenario, the guessing success rate of PASSLLM-II outperforms MS-PointerGuess [49] (which can exploit users' two sister passwords) by 40.45%-140.42%. Further, in multi-password ($\geq$3) scenarios, PASSLLM-II achieves a guessing success rate of 36.63% within 10^3 guesses (here the target password differs from all sister passwords). To the best of our knowledge, our PASSLLM-II is the first model capable of simultaneously utilizing *any* number of sister passwords from the same user to model their password reuse behavior. Further, we analyze the overlap and the edit distance distribution among the passwords cracked by the top three performing models (i.e., our PASSLLM-II, Pass2Edit [44], and PointerGuess [49]). Detailed results are provided in Appendix B and Appendix C of the full version.

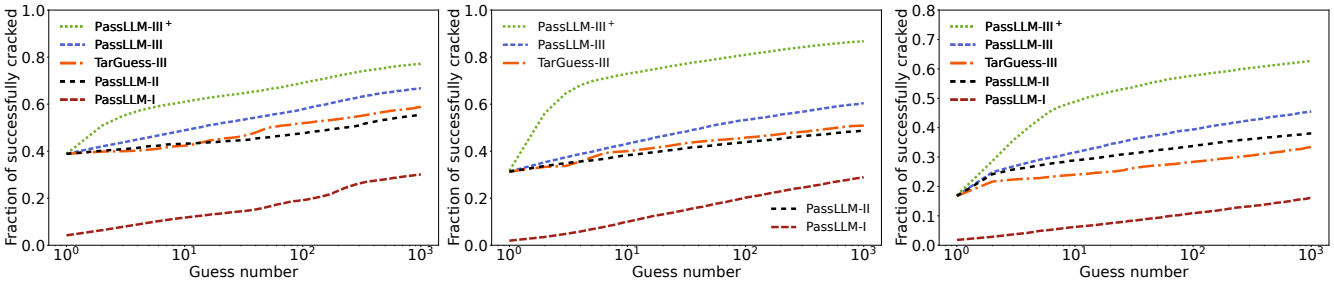
4.6 Multi-source guessing scenarios

Now, we introduce a more powerful attacker who has access to *both the user's PII and sister passwords* (which can be multiple). To the best of our knowledge, only the TarGuess-III model [43] proposed by Wang et al. in 2016 can address such attack scenarios. First, we create a dataset that includes both PII and multiple sister passwords by matching the PII datasets with the COMB dataset using email addresses. We then conduct a fair/comprehensive comparison of PASSLLM-III and TarGuess-III [43] using these datasets (see Table 6).



(a) COMB($pw_1 \rightarrow pw_2$) (b) 126→CSDN (c) COMB($pw_1 \sim pw_n \rightarrow pw_{n+1}$, $n \in [2,9]$)

Figure 7: Experimental results of our PASSLLM-II against other reuse-based password models (i.e., TarGuess-II [43], Pass2Path [29], Pass2Edit [44], PassBERT [51] and PointerGuess [49]). Note that the 126→CSDN scenario is identical to the experimental scenario in [44, 49], and is included to demonstrate that we have correctly run the relevant models and conducted a fair comparison. In Figs. 7(a) and 7(c), we have excluded the cases where the target password is the same as any of the sister passwords to show the success rate of different models in a single guess. Overall, our PASSLLM-II performs the best within 10^3 guesses.



(a) 12306-PII&COMB→CSDN-PII&COMB (b) 12306-PII&COMB→Dodonew-PII&COMB (c) 000Web.-PII&COMB→000Web.-PII&COMB

Figure 8: Experimental results of our PASSLLM-III against TarGuess-III [43] in targeted guessing scenarios. PASSLLM-III refers to the model that uses the same auxiliary information (i.e., four PIIs and one sister password) as TarGuess-III; PASSLLM-III$^+$ further includes phone number, website name and multiple sister passwords. The guessing success rates of our PASSLLM-III$^+$ are 32.97%-103.35% higher than TarGuess-III [43] within 10^2 guesses. To intuitively show the risks associated with different types of information exploited, we also compare the performance of PASSLLM-I and PASSLLM-II in the same scenario. Results show that the leakage of sister passwords poses a greater threat than PII alone, with guessing success rates further increasing when both are combined.

Table 6: Multi-source (PII and sister passwords) guessing scenarios

#	Training set	size	Test set	size	Information used
1	12306&COMB†	119,197	Dodonew&COMB CSDN&COMB	10k	Email, Name, Birthday, Username,
2	000Web.&COMB	134,676	000Web.&COMB		One sister password‡
3	12306&COMB	119,197	Dodonew&COMB CSDN&COMB	10k	4 PIIs+Phone, Web name, $n(\geq 2)$ sister passwords

†12306&COMB means that the 12306 dataset containing PII is combined with COMB, which includes emails and multiple passwords. The datasets are matched by email to obtain PII and multiple sister passwords for the same user. Then, one password is randomly selected from the set of sister passwords to be the target password.

‡TarGuess-III [43] can only utilize the five available information. Thus, we create this scenario to fairly compare PASSLLM-III with TarGuess-III. Additionally, our PASSLLM-III can leverage more information (e.g., website names and multiple sister passwords), which corresponds to Scenarios #3.

Experimental results. Experimental results (see Fig. 8) show that our PASSLLM-III, when using the same auxiliary information as TarGuess-III, achieves a significantly higher guessing success rate, outperforming TarGuess-III [43] by 13.44%-36.14% within 10^3 guesses. Furthermore, when additional auxiliary information and sister passwords are incorporated (i.e., PASSLLM-III$^+$), the success rate improves even further, surpassing 62.64%-87.78% within 10^3 guesses.

Improvement principles. TarGuess-III [43] builds on the foundations of TarGuess-I and TarGuess-II, inheriting their limitations. More specifically, it continues to suffer from the incomplete PII label definitions of TarGuess-I and the complex structural and character-level transformations of TarGuess-II. In contrast, our proposed PASSLLM-III simplifies this by directly employing the available auxiliary information (i.e., both users' PIIs and sister passwords) as part of the prompts. This allows PASSLLM-III to learn key password-related characters directly from the auxiliary information, *avoiding the intermediate precision loss caused by converting PIIs into labels or interpreting password modifications as complex transformations*.

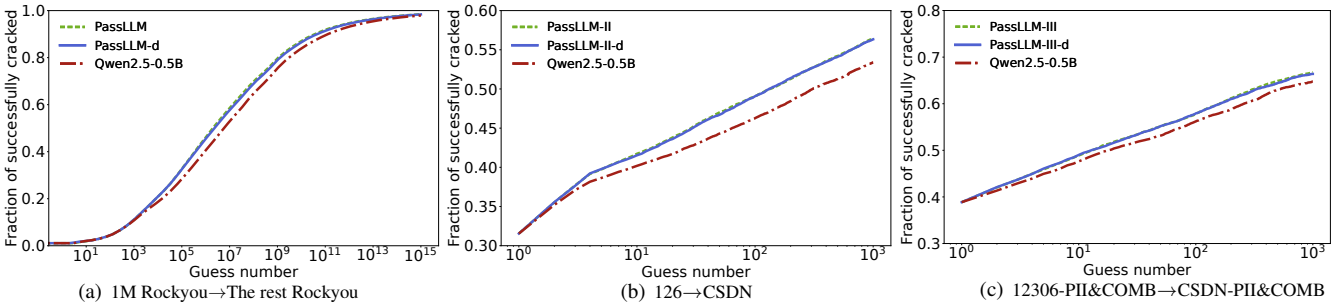
4.7 Improving computational efficiency via model distillation

We now evaluate the computational efficiency of our PASSLLM in both targeted and trawling guessing scenarios on a server with a single RTX 4090 GPU (see Table 7).

To further improve guessing efficiency, we apply the model distillation technique [17] to accelerate the generation speed of PASSLLM while maintaining its performance.

More specifically, we use the 7-billion-parameter version of PASSLLM as the *teacher model*. For an input sequence consisting of a prompt and a complete password (e.g., `prompt+qwerty`), the teacher model outputs a sequence of probability distributions, each representing the likelihood of the next character conditioned on the preceding context:

$$\mathbf{P}_T = [\mathrm{P}(c_1 \mid \text{prompt}), \ldots, \mathrm{P}(c_n \mid \text{prompt} + c_1 + \cdots + c_{n-1})], \quad (6)$$



(a) 1M Rockyou→The rest Rockyou (b) 126→CSDN (c) 12306-PII&COMB→CSDN-PII&COMB

Figure 9: Comparison of success rates between the distilled and original models across three guessing scenarios. PASSLLM-d, the distilled variant (using Qwen2.5-0.5B as student model), achieves nearly identical performance to the original model while significantly reducing parameter size. Qwen2.5-0.5B refers to the same base model fine-tuned directly on the training set without distillation, serving as a baseline to highlight the effectiveness of the distillation process. The higher success rate of Qwen2.5-0.5B in Fig. 9(a) compared to Qwen2.5-0.5B in Fig. 4(a) is due to the different LoRA trainable parameters (see Fig. 13).

Table 7: Performance comparison of the original and distilled versions of PASSLLM across tasks in trawling and targeted scenarios†.

Scenarios‡	Item	PASSLLM-7B	PASSLLM-d (0.5B)	Speedup
Trawling	Training	7 hours	12 hours	—
	Sampling	540 pw/s	5,100 pw/s	9.4 times
	Generation	260 pw/s	3,000 pw/s	11.5 times
Targeted	Training	4 hours	8 hours	—
	Generation	80-150 pw/s	460-590 pw/s	4-6 times

†These results were evaluated on a server equipped with a single RTX 4090 GPU.
‡Trawling training set: 1M RockYou; Targeted training set: 126→CSDN.

where $c_1, \ldots, c_n$ denote the characters in the password.

The *student model*, a lightweight variant (denoted as PASSLLM-d), takes the same input and is trained to approximate the output distributions of the teacher model. To achieve this, we adopt a combined training objective that balances knowledge distillation and next-character prediction. Specifically, we minimize a loss that interpolates between the Kullback–Leibler (KL) divergence from the teacher model and the standard cross-entropy loss with respect to the ground-truth characters. The overall loss function is defined as:

$$\mathcal{L}\text{combined} = \alpha \mathcal{L}_{\text{KL}} + (1-\alpha)\mathcal{L}_{\text{CE}}. \tag{7}$$

The first term, $\mathcal{L}_{\text{KL}}$, encourages the student model to align its output distributions with those of the teacher model at each character position:

$$\mathcal{L}_{\text{KL}} = \frac{1}{n}\sum_{i=1}^{n} \text{KL}\left(P_T(c_i) \parallel P_S(c_i)\right). \tag{8}$$

The second term, $\mathcal{L}_{\text{CE}}$, ensures that the student model also learns to predict the correct next character given the input sequence, using the standard cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{n} \log P_S(c_i \mid c_1, \ldots, c_{i-1}). \tag{9}$$

Here, $P_T(c_i)$ and $P_S(c_i)$ denote the predicted distributions over the vocabulary for character position i from the teacher and student models, respectively. This combination allows the student model to benefit both from the teacher's soft supervision and from direct supervision via labeled data. In our experiments, we set α=0.5.

This distillation process enables the student model (i.e., PASSLLM-d) to approximate the behavior of the larger teacher model while achieving significantly improved computational efficiency, making it better suited for large-scale password enumeration tasks.

We experiment with student models of various sizes (e.g., Qwen2.5-3B, 1.5B, and 0.5B). The experimental results (see Fig. 9 and Fig. 10(i) in Appendix) demonstrate that the 0.5B model achieves the smallest parameter size without reducing the guessing success rate. Table 7 presents the performance evaluation of the distilled model.

Since 0.5B is the smallest model size available in the Qwen2.5 family, we include a lightweight GPT-2 model with only 124M parameters for further comparison. However, the 124M GPT-2 model shows a substantial decline in guessing success rate (see Fig. 10(i)). Designing effective student models with fewer parameters remains an open challenge and will be an important direction for our future work.

5 Security implications and future work

Digital forensics. The PASSLLM framework proposed in this paper plays a significant role in digital forensics, particularly in targeted guessing scenarios. It is capable of leveraging a wide range of contextual data, including but not limited to various forms of personally identifiable information (PIIs) and multiple sister passwords. With its carefully designed training paradigm and generation algorithms, PASSLLM achieves a much higher guessing success rate compared to existing state-of-the-art password models. As a result, PASSLLM is highly effective for digital forensic applications, such as unlocking critical devices of suspects, recovering hashed password files, and other urgent forensic needs.

Password strength meter. An important application of password guessing models is to accurately evaluate password strength from an attacker's perspective, functioning as a password strength meter (PSM). Our PASSLLM can serve as a server-side or locally deployed PSM, providing rapid estimates of the guess number for a given password. Particularly, we use the unsafe error metric (which is widely

used [26,50,52]) to evaluate our PASSLLM-PSM against two leading PSM (i.e., FLA-PSM [47] and RankGuess-PSM [52]. Experimental results (see Fig. 12 in Appendix) show that our PASSLLM-PSM has fewer unsafe errors compared to FLA-PSM [47] and RankGuess-PSM [52].

Interpretable PSM in future. A major advantage of current LLMs (e.g., ChatGPT-4o) in evaluating password strength lies in their ability to provide natural language explanations and their strong conversational capabilities, allowing them to continuously offer optimization suggestions through dialogue. However, as previously mentioned, due to the limitations of training data, these LLMs can sometimes provide misleading advice (see Table 14 in the full version). To address this issue, a feasible solution is to pre-train large models using the latest corpus reflecting advancements in password security, followed by fine-tuning with expert-constructed training data that includes interpretable password strength evaluations. This approach aims to develop a tool capable of accurately evaluating password strength and offering interpretable analyses tailored to different user-provided information.

Direct password generation using LLMs. We have tested several prominent LLMs by prompting them with PII to generate a number of passwords (see Table 14 in the full version). While most models produce seemingly reasonable outputs, our findings show that their guessing success rates are generally low ($<$2% within 20 guesses). This is primarily because these models are not trained on password-specific datasets, which limits their ability to accurately model real-world user password creation behaviors. Additionally, some models, due to built-in security mechanisms, do not directly generate passwords and instead offer advice on creating strong, secure ones. Typical suggestions include creating long passwords, avoiding personal information, and using password managers. However, some of this guidance, such as requiring complex character combinations and regularly changing passwords, is *outdated and inaccurate*. For example, NIST's latest authentication guidelines [15] recommend that the most secure password policies focus on ensuring sufficient length *without additional compositional constraints*, and advise *against password expiration*, a view also supported by Tan et al. [36].

These observations highlight an important direction for our future work: building a specialized large language model for the field of password security. An interesting finding is that even models designed to restrict direct password generation (based on provided PII) can be adapted through minor prompt modifications. For example, when the prompt language is switched to Chinese, ChatGPT-4o successfully generates the corresponding passwords.

Large-scale offline guessing using LLMs. Despite the generation algorithm proposed in this paper allowing large models to efficiently generate password guesses at scale, utilizing LLMs for even larger-scale tasks (e.g., over 10^{10}) in trawling guessing remains a significant challenge. However, this challenge could potentially be addressed through an alternative approach: instead of enabling LLMs directly generate massive guess dictionaries, LLMs could be leveraged to produce scalable transformation rules based on an input set of passwords. These rules can then be applied to large-scale training data to efficiently generate extensive guess dictionaries. We consider investigating this approach as part of our future work.

6 Conclusion

This paper, for the first time, introduces efficient parameter fine-tuning for LLM-based password guessing, and designs four new guessing models for the four most representative guessing scenarios: trawling guessing, targeted guessing based on PII and/or on reuse. Extensive experiments on 11 real-world password datasets show the effectiveness and scalability of our PASSLLM. This work provides a brand new technical route for modeling users' password guessability and opens up new directions for password guessing using LLMs.

Acknowledgement

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China under Grants Nos. 62172240 and 62222208, and by the Fundamental Research Funds for the Central Universities, Nankai University (Grant No. 63253229). See the full version of this paper at `https://wangdingg.weebly.com/publications.html`.

Ethical considerations

Our study uses publicly available password datasets that were leaked through past breaches and have been widely used in prior password research (e.g., [25,26,30,32,42,43]). However, besides passwords, our study incorporates associated personally identifiable information (PII), such as usernames and emails, in the context of targeted guessing scenarios. We acknowledge that this introduces a potentially higher risk to data subjects than studies using passwords alone. Also, we recognize that the individuals whose data were exposed in these breaches have not consented to the use of their data in research, nor have any meaningful opportunity to consent. Although these datasets are already publicly available, their continued use still raises important ethical considerations.

To mitigate potential risks to affected individuals, we take four precautions to ensure that our research does not cause any additional harm to users: (1) We make sure no further dissemination of password data to other parties; (2) All our datasets are (ever) publicly available, and are stored and processed on computers not linked to the Internet; (3) We only report the aggregated statistical information and treat each individual account as confidential (so that using it in our research will not increase the risk to the corresponding victim,

i.e., no personally identifiable information can be learned); (4) We delete the sensitive information (e.g., PII) once our analysis is completed.

In particular, our analysis has been guided by the related Scenario B from [22], which explores the moral dilemma of studying stolen data that subjects have explicitly asked to be deleted. That scenario highlights the tension between deontological ethics (which emphasizes the rights of data subjects and the duty to respect privacy) and consequentialist ethics (which justifies data use if it leads to substantial public benefit). In line with the latter view, we believe the additional scientific insights gained from analyzing real-world password data can significantly enhance password security. Given the data's prior public exposure and our strict safeguards, the additional knowledge gained from the research outweighs what is believed to be a small risk of additional harm to the data subjects. Specifically, our work aims to improve password security through the responsible use of real-world leaked password datasets. By analyzing password vulnerabilities in both trawling and targeted attack scenarios, we develop practical defenses such as a password strength meter based on LLMs. This tool evaluates the strength of passwords by estimating their guessability, and does not explicitly generate passwords, thereby reducing the risk of misuse.

Open science

To support the reproducibility of this work, we release the source code used in our study[14], including (i) the fine-tuning code for adapting pre-trained large language models to password guessing tasks; (ii) the Monte Carlo simulation code used to estimate guessing performance in the trawling scenarios; (iii) the actual password generation (i.e., the two-stage generation algorithm) code for the trawling scenario; (iv) the dynamic beam search algorithm for the targeted guessing scenario; (v) the model distillation code used to reduce model size while preserving guessing effectiveness; and (vi) two trained model checkpoints that can reproduce some key experimental results in our paper (e.g., Fig. 9(a) and Fig. 9(b)).

References

[1] *COMB: The Big Password Leak*, Apr. 2021, https://www.syhunt.com/en/?n=Articles.COMBPasswordLeak2021.

[2] *North Americans are the most affected by password leaks*, Sep 2023, https://surfshark.com/research/chart/leaked-passwords.

[3] *Data breach hits Indonesia's National Civil Service Agency ahead of Independence Day*, Aug. 2024, https://www.teiss.co.uk/news/data-breach-hits-indonesias-national-civil-service-agency-ahead-of-independence-day-14467.

[4] *Tongyi Qianwen*, Nov. 2024, https://tongyi.aliyun.com/qianwen/.

[5] *What to Watch for Following The Post Millennial Data Breach*, May 2024, https://spycloud.com/blog/the-post-millennial-data-breach-analysis/.

[14] https://zenodo.org/records/15612295

[6] M. Atzori, E. Calò, L. Caruccio *et al.*, "Evaluating password strength based on information spread on social networks: A combined approach relying on data reconstruction and generative models," *Online Social Netw. Med.*, vol. 42, pp. 1–23, 2024.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR 2015*.

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

[9] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*, pp. 553–567.

[10] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. NDSS 2014*, pp. 1–15.

[11] M. Dell'Amico and M. Filippone, "Monte carlo strength evaluation: Fast and reliable password checking," in *Proc. ACM CCS 2015*.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT 2019*, pp. 4171–4186.

[13] M. Durr, *Data breach exposes logins, passwords for 184M Apple, Facebook, Google users and more*, May 2025, https://www.mlive.com/news/2025/05/data-breach-exposes-logins-passwords-for-184m-apple-facebook-google-users-and-more.html.

[14] S. Dziembowski, S. Jarecki, P. Kedzior, H. Krawczyk, C. N. Ngo, and J. Xu, "Password-protected threshold signatures," in *Proc. ASIACRYPT, 2024*, pp. 174–206.

[15] P. A. Grassi, E. M. Newton, R. A. Perlner, and et al., "NIST 800-63B digital identity guidelines: Authentication and lifecycle management," McLean, VA, Tech. Rep., Feb 2020.

[16] C. Herley and P. Van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Secur. Priv.*, vol. 10, pp. 28–36, 2012.

[17] G. E. Hinton, O. Vinyals, and J. Dean, *Distilling the Knowledge in a Neural Network*, March 2015, https://arxiv.org/abs/1503.02531.

[18] B. Hitaj, P. Gasti, G. Ateniese, and F. Pérez-Cruz, "PassGAN: A deep learning approach for password guessing," in *Proc. ACNS 2019*.

[19] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *Proc. ICLR 2022*, pp. 1–13.

[20] A. Q. Jiang, A. Sablayrolles, A. Mensch *et al.*, "Mistral 7b," 2023, https://arxiv.org/pdf/2310.06825.

[21] J. Katz, R. Ostrovsky, and M. Yung, "Efficient password-authenticated key exchange using human-memorable passwords," in *Proc. EUROCRYPT 2001*, pp. 475–494.

[22] T. Kohno, Y. Acar, and W. Loh, "Ethical frameworks and computer security trolley problems: Foundations for conversations," in *Proc. USENIX SEC 2023*, pp. 5145–5162.

[23] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[24] Y. Li, H. Wang, and K. Sun, "A study of personal information in human-chosen passwords and its security implications," in *Proc. IEEE INFOCOM 2016*, pp. 1–9.

[25] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE S&P 2014*, pp. 689–704.

[26] W. Melicher, B. Ur, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, lean and accurate: Modeling password guessability using neural networks," in *Proc. USENIX SEC 2017*, pp. 175–191.

[27] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. ACM CCS 2005*, pp. 364–372.

[28] OpenAI, *GPT-4 Technical Report*, 2024, https://arxiv.org/pdf/2303.08774.

[29] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE S&P 2019*, pp. 417–434.

[30] D. Pasquini, G. Ateniese, and C. Troncoso, "Universal neural-cracking-machines: Self-configurable password models from auxiliary data," in *Proc. IEEE S&P 2024*, pp. 36–54.

[31] D. Pasquini, M. Cianfriglia, G. Ateniese, and M. Bernaschi, "Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries," in *Proc. USENIX SEC 2021*, pp. 821–838.

[32] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, "Improving password guessing via representation learning," in *Proc. IEEE S&P 2021*, pp. 1382–1399.

[33] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 5485–5551, 2020.

[34] J. Rando, F. Pérez-Cruz, and B. Hitaj, "Passgpt: Password modeling and (guided) generation with large language models," in *Proc. ESORICS 2023*, pp. 164–183.

[35] X. Su, X. Zhu, Y. Li, Y. Li, C. Chen, and P. Esteves-Veríssimo, "Pagpassgpt: Pattern guided password guessing via generative pretrained transformer," in *Proc. IEEE/IFIP DSN 2024*, pp. 429–442.

[36] J. Tan, L. Bauer, N. Christin, and L. F. Cranor, "Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements," in *Proc. ACM CCS 2020*, pp. 1407–1426.

[37] H. Touvron, T. Lavril, G. Izacard *et al.*, "Llama: Open and efficient foundation language models," 2023, https://arxiv.org/pdf/2302.13971.

[38] 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 *Proc. USENIX SEC 2015*, pp. 463–481.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS 2017*, pp. 5998–6008.

[40] R. Veras, C. Collins, and J. Thorpe, "On semantic patterns of passwords and their security impact," in *Proc. NDSS 2014*, pp. 1–16.

[41] D. Wang and P. Wang, "The emperor's new password creation policies," in *Proc. ESORICS 2015*, pp. 456–477.

[42] D. Wang, P. Wang, D. He, and Y. Tian, "Birthday, name and bifacial-security: Understanding passwords of Chinese web users," in *Proc. USENIX SEC 2019*, pp. 1537–1555.

[43] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*, pp. 1242–1254.

[44] D. Wang, Y. Zou, Y.-A. Xiao, S. Ma, and X. Chen, "Pass2edit: A multi-step generative model for guessing edited passwords," in *Proc. USENIX SEC 2023*, pp. 983–1000.

[45] D. Wang, Y. Zou, Z. Zhang, and K. Xiu, "Password guessing using random forest," in *Proc. USENIX SEC 2023*, pp. 965–982.

[46] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE S&P 2009*, pp. 391–405.

[47] D. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX SEC 2016*, pp. 157–173.

[48] Z. Whittaker, *LoanDepot says about 17M customers had personal data and Social Security numbers stolen during cyberattack*, Feb 2024, https://techcrunch.com/2024/02/26/loandepot-millions-sensitive-personal-data-ransomware/.

[49] K. Xiu and D. Wang, "Pointerguess: Targeted password guessing model using pointer mechanism," in *Proc. USENIX SEC 2024*, pp. 5555–5572.

[50] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han, "Chunk-level password guessing: Towards modeling refined password composition representations," in *Proc. ACM CCS 2021*, pp. 5–20.

[51] 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 *Proc. USENIX SEC 2023*, pp. 1001–1018.

[52] T. Yang and D. Wang, "Rankguess: Password guessing using adversarial ranking," in *Proc. IEEE S&P 2025*, pp. 682–700.

[53] Z. Zhang, Y. Wang, and K. Yang, "Strong authentication without temper-resistant hardware and application to federated identities," in *Proc. NDSS 2020*, pp. 1–15.

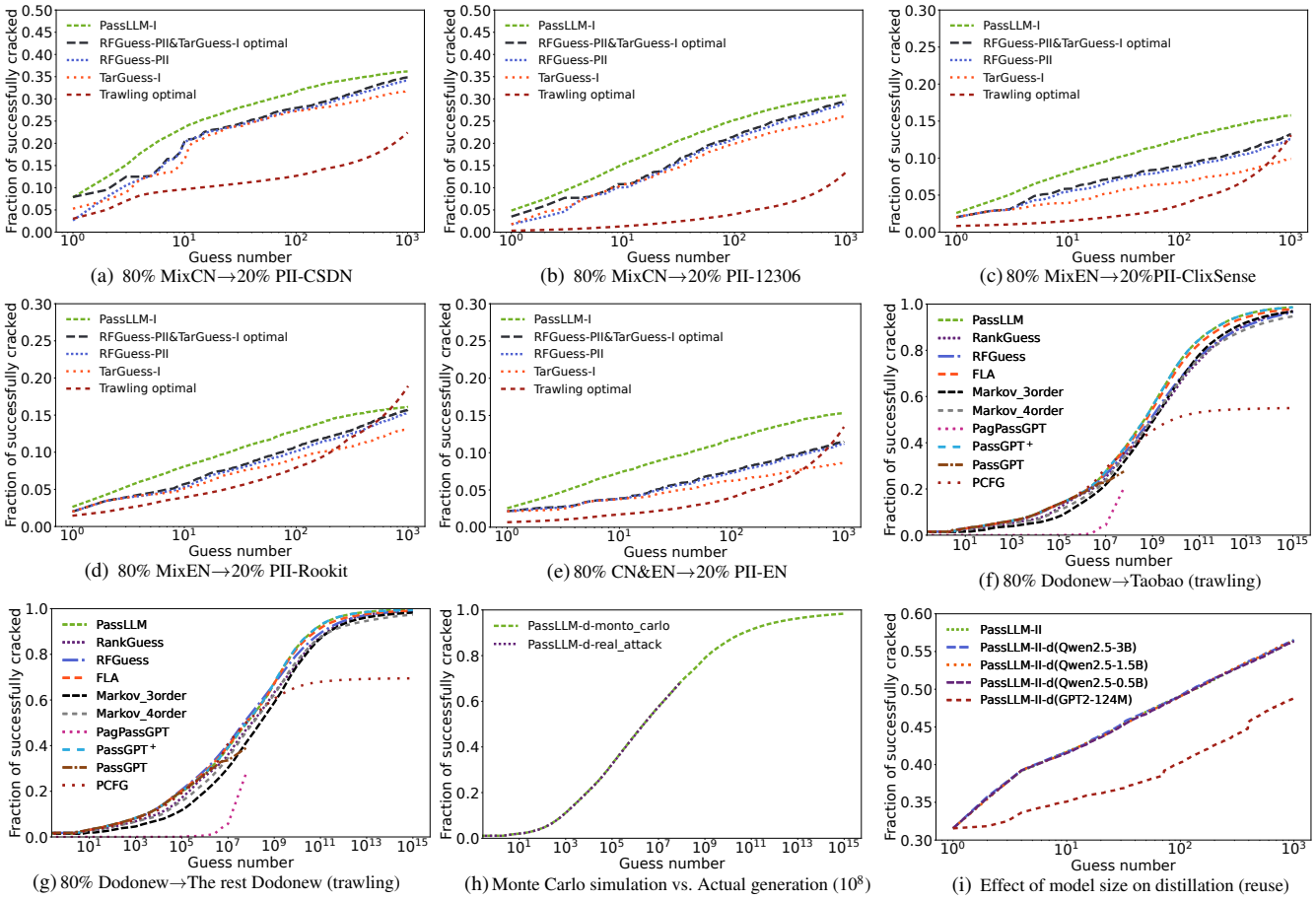
[54] V. Zimmermann, "From the quest to replace passwords towards supporting secure and usable password creation," Ph.D. dissertation, Technical University of Darmstadt, Germany, 2021.

A Password models for comparison

In this section, we provide the detailed parameter settings for each of the counterparts of PASSLLM-I∼III.

A.1 Trawling password models for comparison

- **RankGuess**. We adopt the same experimental setup as Yang-Wang [52], with a penalty factor γ of 0.1 applied, using KL divergence as the loss function. The optimizer is Adam, with an initial learning rate of 0.001 and momentum parameters β_1=0.5 and β_2=0.999. The model is trained for 20 epochs, including 5 pre-training epochs. The guesser and the ranker are trained with 17 group numbers and a dropout rate of 0.3.
- **RFGuess**. We configure RFGuess parameters following Wang et al. [45], using scikit-learn to train a random forest with 30 trees. The minimum samples per leaf is 10, and the maximum feature ratio is 80%.
- **PCFG**. The PCFG we use follows the approach in [25], where the probability of the **L**(etter) segment is derived from the training set, improving upon the original version employed by Weir et al. [46].
- **Markov**. Due to the influence of order, this work implements both 3-order and 4-order Markov simultaneously, applying Laplace smoothing and end symbol regularization as recommended by Ma et al. [25].
- **FLA**. We use the source code of FLA [26] and follow its recommended parameters. More specifically, we train a model with three LSTM layers, each containing 200 cells (i.e., the small model in [26]), and two fully connected (FC) layers, for 20 epochs.
- **PassGPT**. We use the open-source code of PassGPT [34] and follow their recommended configuration. Specifically, we train the model using the AdamW optimizer with a learning rate of 5e-5 and a linear decay schedule.
- **PagPassGPT**. We use the open-source implementation of PagPassGPT [35] and follow its original training settings. Specifically, the model is trained with the AdamW



(a) 80% MixCN→20% PII-CSDN (b) 80% MixCN→20% PII-12306 (c) 80% MixEN→20%PII-ClixSense

(d) 80% MixEN→20% PII-Rookit (e) 80% CN&EN→20% PII-EN (f) 80% Dodonew→Taobao (trawling)

(g) 80% Dodonew→The rest Dodonew (trawling) (h) Monte Carlo simulation vs. Actual generation (10^8) (i) Effect of model size on distillation (reuse)

Figure 10: Supplementary experimental results. Subfigures 10(a)-10(e) show results for PII-based targeted guessing scenarios. Subfigures 10(f) and 10(g) present results for the trawling guessing scenario. Subfigure 10(h) compares Monte Carlo simulation with actual generation (10^8 guesses) in the trawling guessing scenario. Subfigure 10(i) shows the distillation performance across different model sizes (Here we take the 126→CSDN reuse scenario for example).

optimizer and an initial learning rate of 5e-5. The architecture is based on GPT2, with 12 hidden layers and 8 attention heads per layer, a maximum input length of 32 tokens, and an embedding size of 256.

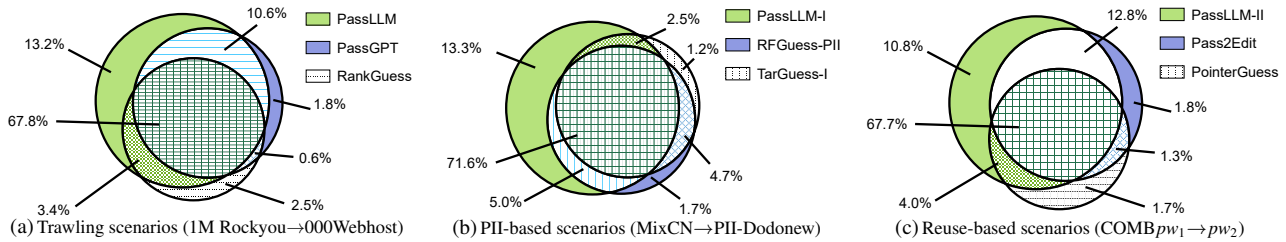
A.2 PII-based targeted password models

- **TarGuess-I**. As with [43], we employ the 26 types of labels (see Table 8 in the full version at `https://bit.ly/43Woqk5`) defined by the TarGuess-I [43] model across five categories of PIIs.
- **RFGuess-PII**. As with [45], we train a random forest model consisting of 30 decision trees, with a minimum of 10 leaf nodes per tree and a maximum feature ratio of 80%. All other hyperparameters are set to the default values provided by the scikit-learn framework.

Since RFGuess-PII [45] shares a similar modeling principle with Targeted-Markov [41], but consistently achieves higher guessing success rates (as evidenced by comprehensive experimental results in [45]), we choose RFGuess-PII as a representative baseline in our comparative evaluation.

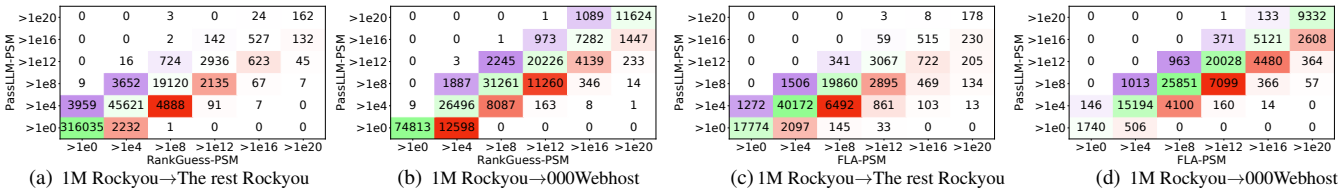
A.3 Reuse-based targeted password models

- **Pass2Edit.** It was proposed by Wang et al. [44] in 2023, which employs a "multi-step decision" mechanism for generating edit operations. It takes both the original password and the current modified password as inputs to predict the next edit operation. The predicted operation is then applied to the modified password to generate input for the subsequent timestep. As with [44], we use a 3-layer GRU and a 2-layer FC (fully connected layer). The learning rate is 0.001 and the dropout rate is 0.4.
- **PointerGuess.** This password model, proposed by Xiu and Wang [49] at USENIX Security'24, redefines the conditional password probability as the combination of copying characters from the sister password and generating new characters. As with [49], we set $\varepsilon = 1 \times 10^{-12}$ for smoothing, use Bahdanau attention [7], a learning rate of 0.001, and a dropout rate of 0.5.
- **PassBERT.** Proposed by Xu et al. [51] in 2023, PassBERT utilizes the BERT model to predict the sequence of edit operations based on each user's sister password.



(a) Trawling scenarios (1M Rockyou→000Webhost) (b) PII-based scenarios (MixCN→PII-Dodonew) (c) Reuse-based scenarios (COMB$pw_1 \rightarrow pw_2$)

Figure 11: Overlap analysis of guessed passwords among different models in three guessing scenarios (i.e., trawling, PII-based, and reuse-based). In each scenario, we select the three most effective guessing models in terms of success rate. The results show that over 67% of the guessed passwords overlap across models in all three scenarios. Notably, our PassLLM(-I/II) achieves the highest proportion of independently guessed passwords.



(a) 1M Rockyou→The rest Rockyou (b) 1M Rockyou→000Webhost (c) 1M Rockyou→The rest Rockyou (d) 1M Rockyou→000Webhost

Figure 12: We compare our PassLLM-PSM with FLA-PSM [26] and RankGuess-PSM [52] using the unsafe error metric. PassLLM-PSM significantly outperforms all its counterparts. For example, in Fig. 12(b), our PassLLM-PSM evaluates 6,838 passwords as being guessable between 10^4 and 10^8 guesses, while the RankGuess-PSM [52] rates them as requiring more than 10^8 guesses. Overestimates of strength are indicated in shades of red, underestimates in purple, and accurate estimates in green. The intensity of the color increases with the number of passwords in each cell.

As with [51], we use two attention heads, a learning rate of 10^{-5}, the Adam optimizer, a dropout rate of 0.1, and a sequence length of 32/34.

- **Pass2Path.** We use the open-source code provided by [29] and adopt their recommended hyperparameters. The model consists of a three-layer RNN with 128 hidden units per layer and a dropout rate of 0.4. It is trained using a learning rate of 0.0003 for 20 epochs, during which the validation loss converges without noticeable overfitting. We note that both Pass2Edit [44] and TarGuess-II [43] leverage an additional popular password dictionary to enhance their ability to guess common passwords. Following the same strategy, we augment Pass2Path by mixing its reuse dictionary with passwords from the popular dictionary at a 2:1 ratio (a setting identified as most effective in [44]). This enhanced version is referred to as Pass2Path-mix in Figures 7(a) and 7(b).
- **MS-PointerGuess.** It was introduced by Xiu and Wang [49] in 2024, and incorporates a "two-encoder" module that leverages two sister passwords from the same user. It is important to note that while this password model can potentially accommodate a larger number ($\geq$3) of sister passwords, its scalability is constrained, as it can only employ a fixed number of encoders.

B Overlap analysis

Trawling scenarios. We analyze the overlap among the cracked passwords generated by the three most effective models in terms of guessing success rate: PassLLM, RankGuess [52], and PassGPT [34], as shown in Figure 11(b). Among the 99,647 testing accounts, a total of 40,785 passwords are cracked within 10^{10} guesses. Specifically, PassLLM successfully cracks 38,794 passwords, surpassing RankGuess with 30,318 and PassGPT with 32,977. In terms of overlap, 27,656 passwords, accounting for 67.8% of all cracked results, are successfully guessed by all three models. More importantly, PassLLM uniquely cracks 5,399 passwords, representing 13.2% of the total cracked passwords. This is significantly higher than the proportions uniquely cracked by RankGuess and PassGPT, which are 2.5% and 1.8% respectively. These results highlight the superior generalization and guessing capability of PassLLM over existing methods.

PII-based scenarios. We analyze the passwords *uniquely* cracked by our PassLLM-I compared to TarGuess-I [43] and RFGuess-PII [45]. Figure 11(a) illustrates the overlap in the passwords cracked by PassLLM, RFGuess-PII, and TarGuess-I in the MixCN→Dodonew scenario. More specifically, over 70% of the passwords cracked by PassLLM, RFGuess-PII, and TarGuess-I overlap. Notably, PassLLM-I cracks approximately 13% more passwords that the other two models do not. We show several typical cracked examples in Appendix C of our full version paper.

Reuse-based scenarios. We analyze the overlap among the cracked passwords by PassLLM-II, Pass2Edit [44], and PointerGuess [49]. As shown in Figure 11(c), within 100 guesses, a total of 2,326 passwords are successfully cracked. Among these, PassLLM-II uniquely cracks 252 passwords, which accounts for 10.8% of all cracked passwords and is significantly higher than the numbers achieved by Pass2Edit and PointerGuess, which uniquely crack 41 and 39 passwords respectively. In addition, 67.7% of the cracked set, are successfully guessed by all three models, while 297 passwords are cracked by both PassLLM-II and Pass2Edit but missed

Table 8: Different prompts used in password guessing[†]

Prompt #	Detailed instructions
Prompt 1	As a **trawling password guessing model**, your task is to generate passwords. \n password (Instruction related to password guessing)
Prompt 2	password (**No instructions, directly input training password**)
Prompt 3	You are a **password defense advisor**. Generating human-chosen passwords is not allowed. \n password (An instruction aimed at preventing password guessing)
Prompt 4	As a **chemistry expert**, explain the basic types of chemical reactions (synthesis, decomposition, etc.) and provide examples. \n password (An instruction unrelated to the field of password security)

[†] We design four distinct prompts to evaluate the model's performance. Descriptions with scenario identifiers are highlighted in **bold**.

Name	PassLLM setting	Distillation setting
Trawling batch size	1,024	1,024
Targeted batch size	64	64
Trawling LR	1e-4	1e-4
Targeted LR	5e-4	5e-4
Epochs	3	10
Optimizer	AdamW	AdamW
Weight decay	0.05	0.05
Warmup type	Linear	Cosine
Warmup ratio	0.1	0.2
LoRA rank (r)	16	16
LoRA alpha	32	32
Trawling LoRA dropout	0.3	0.2
Targeted LoRA dropout	0.2	0.2
LoRA modules	q/k/v	q/k/v/o/mlp

Figure 13: Hyperparameter settings for our PassLLM and model distillation. Table 9 summarizes the detailed neural network architecture of PassLLM.

by PointerGuess. These findings demonstrate the strong individual effectiveness of PassLLM-II and its complementary nature with other leading approaches.

C Evaluation of PassLLM-PSM

Unsafe errors. Unsafe errors refer to the misclassification of passwords as "strong" when they are, in fact, easily guessable or weak, leading to an inaccurate evaluation of their security. These errors arise when password strength meters mistakenly label weak passwords as secure, giving users a false sense of protection. This overestimation of password strength can have serious consequences, as users may be unaware of the vulnerabilities in their passwords, exposing them to attacks and weakening overall system security. In this work, we use the unsafe error metric to evaluate and compare our PassLLM-PSM with FLA-PSM [26] and RankGuess-PSM [52]. Results (see Fig. 12) show that our PassLLM-PSM consistently demonstrates fewer unsafe errors compared to both RankGuess-PSM [52] and FLA-PSM [26]. Here, the training set and the test set are the same as Fig. 5.

D Monte carlo simulation using PassLLM

In large-scale trawling guessing scenarios, we adopt the Monte Carlo approach [11] to estimate the guess number of a target password. More specifically, PassLLM is queried with a *fixed* prompt: As a trawling password guessing model, your task is to generate passwords. This prompt induces a fixed generative distribution over the space of possible passwords, implicitly defining a probability distribution $p(\cdot|\text{prompt})$ such that the total probability mass sums to 1.0.

To estimate the guess number of a target password α in this fixed prompt-induced distribution, we compare it against a sample set $\Theta = \beta_1, \beta_2, \ldots, \beta_n$, where each β is independently sampled from PassLLM using the same prompt. Both the samples β and the target α are assigned probabilities under the same model-defined distribution $p(\cdot|\text{prompt})$, ensuring consistency within the shared probabilistic space. Following the Monte Carlo estimation approach of Dell'Amico et al. [11], we estimate the guess number $C_\Delta(\alpha)$ of password α as:

$$C_\Delta = \sum_{\beta \in \Theta} \begin{cases} \frac{1}{n \cdot p(\beta)} & \text{if } p(\beta) > p(\alpha) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

While sampling from large language models (LLMs) proportional to generation probability is generally non-trivial, the use of a fixed prompt ensures that each generated password lies within the same probability space.

Note that our two-stage generation algorithm guarantees that the Monte Carlo estimate of a password's guess number aligns with its expected rank under PassLLM.

Table 9: The neural network architecture used in our PassLLM

Component	Details
Model	MistralModel
Embedding	Embedding(32,000, 4,096)
Layers	ModuleList
Layer (0-31)	32 x MistralDecoderLayer
Self Attention	MistralSdpaAttention(q_proj, k_proj, v_proj, o_proj)
q_proj	Linear(in_features=4,096, out_features=4,096)
k_proj	Linear(in_features=4,096, out_features=1,024)
v_proj	Linear(in_features=4,096, out_features=1,024)
o_proj	Linear(in_features=4,096, out_features=4,096)
Rotary Embedding	MistralRotaryEmbedding()
MLP	MistralMLP(gate_proj, up_proj, down_proj)
gate_proj	Linear(in_features=4,096, out_features=14,336)
up_proj	Linear(in_features=4,096, out_features=14,336)
down_proj	Linear(in_features=14,336, out_features=4,096)
Activation Function	SiLU()
Input LayerNorm	MistralRMSNorm((4,096,), eps=1e-05)
Post Attention LayerNorm	MistralRMSNorm((4,096,), eps=1e-05)
Norm	MistralRMSNorm((4,096,), eps=1e-05)
LM Head	Linear(in_features=4,096, out_features=32,000)