Generating and Attacking Passwords with Misspellings by Leveraging Homophones

Shiva Houshmand(✉), Smita Ghosh, and Jared Maeyama

Santa Clara University, Santa Clara, CA, USA
{shiva.houshmand,sghosh3,jmaeyama}@scu.edu

Abstract. Despite the ineffectiveness of enforcing specific password policies, the influence of long-standing recommendations persists. Many companies advise avoiding dictionary words and promoting the use of misspellings or nonstandard spellings. This study is the first to examine the impact of misspellings in offline password cracking, introducing two novel techniques for generating similar-sounding misspellings using the linguistic concept of homophones. The first technique, ProbP2G, uses phoneme-grapheme correspondences, while the second, LSTM-P2G, employs a deep learning model for phoneme-to-grapheme conversion. We expand attack dictionaries with these homophones and evaluate their impact on password cracking across multiple datasets. Our results show that incorporating these misspellings significantly improves cracking success, highlighting the vulnerabilities in current password policies and offering a new approach to creating more effective attack dictionaries.

Keywords: Password Cracking · Attack Dictionaries · Misspellings

1 Introduction

Many systems still rely on user-created passwords for authentication. Despite security concerns and alternative techniques, passwords remain the most widely used approach. For years, experts have advised adhering to certain rules and policies when creating passwords to increase security. While password policies are constantly evolving, with many sites opting not to enforce specific rules but instead using strength meters [15,23,32] to assess password vulnerability, the influence of long-standing password policies and recommendations persists. An examination of today's password policies and advice from different websites reveals that many major companies suggest avoiding dictionary words altogether, even advocating for using *misspelled* or *non-standard spellings*. See examples of some of these recommendations below [6].

- Dropbox: "Use non-standard spelling (for example, "spellllllling")."
- Paypal: "Misspelled words are stronger because they are not in the dictionary used by attackers."

Published by Springer Nature Switzerland AG 2025
L. Nemec Zlatolas et al. (Eds.): SEC 2025, IFIP AICT 746, pp. 67–81, 2025.
https://doi.org/10.1007/978-3-031-92886-4_5

– Fordham University: "Totally misspell an easily remembered word or phrase."
– Amazon: "Do not use a word that is in a dictionary."
– Twitch: "Be creative! Don't use single dictionary words."
– X (Twitter): "Do not use common dictionary words."

Additionally, in the last decade, slang and Internet acronyms have shaped online communication, especially on social media, allowing users to convey messages efficiently within character limits or fast-paced conversations. For example, LOVE is often spelled as LUV, BECAUSE as BCOZ, and LIFE as LYF. As password policies discourage dictionary words, users are increasingly turning to similar-sounding alternative spellings when creating passwords.

In this work, we explore the effectiveness of password policies that encourage misspellings and non-standard spellings, demonstrating their vulnerability to guessing attacks by expanding attack dictionaries. Password cracking involves generating guesses to find the correct password, with traditional crackers relying on dictionaries of common words and predefined mangling rules, which remain highly effective today. While some internet slang terms can be incorporated by mining social media, there hasn't been a systematic method for generating such words. Research indicates that users often choose memorable elements for passwords [10]. To create words that are not found in dictionaries, users often modify familiar English words to create misspellings. We focus on how to generate these non-standard spellings and propose treating these as *homophones*. Linguistically, homophones are words that sound the same but differ in spelling, meaning, or origin (e.g., loan and lone). Here, our focus is on generating homophonous misspellings– misspelled versions of given words that maintain similar pronunciation–such as LUV for LOVE. Throughout this paper, we use the term *homophone* to refer to any group of characters pronounced the same as another group, not necessarily the words from the language.

Homophones and homographs (words with the same spelling but different meanings or pronunciations) are often studied in speech recognition and text-to-speech systems. Several studies have concentrated on identifying and disambiguating homographs and homophones in automatic speech recognition systems [8,19,35,36,40]. However, there hasn't been much work on generating homophones. In this paper, we present two novel approaches for generating English homophones (or, more accurately, homophonous misspellings). Our *ProbP2G* method first uses a Grapheme-to-phoneme (G2P) model to extract the sequence of phonemes representing the input word, then employs phoneme-grapheme correspondences to generate the top k most likely homophones. In our second method *LSTM-P2G*, we develop an LSTM-based Phoneme-to-grapheme (P2G) model, which generates the corresponding written representation from given pronunciations. We chose this technique because deep learning models have not been extensively explored for the P2G conversion task.

G2P models transliterate graphemes (orthographic symbols) to phonemes (units of the sound system). For example, given the word "chimney" it predicts the pronunciation `/CH IH M N IY/`. P2G models reverse this process, predicting the correct spelling from a given phonetic sequence, such as converting `/CH AH K/` to "chuck".

Our primary contributions can be summarized as follows: (1) We present two novel approaches for generating similar-sounding misspellings of English words by applying the linguistic concept of homophones. (2) This paper is the first comprehensive study on utilizing homophones and misspellings in user-generated passwords and password cracking. (3) We trained and evaluated several G2P models and developed a P2G model for our homophone generation process. (4) We established a metric to assess homophones and evaluate their degree of pronunciation similarity, and (5) demonstrated the impact of enhancing attack dictionaries with misspelled and non-standard spellings by conducting extensive password cracking experiments. Our results show password-cracking improvements ranging from 2% to 20% across multiple datasets. Our research suggests that password recommendations like "do not use a single dictionary word" or "use non-standard spellings" do not necessarily enhance password security. While any password policy might be effective initially and contribute to creating stronger passwords, attackers quickly adapt their methods to exploit these guidelines. Thus, such recommendations may only provide a temporary security benefit.

2 Background and Related Work

Grapheme-to-Phoneme (G2P). Modern text-to-speech (TTS) models rely on G2P conversion to generate pronunciations, especially for out-of-vocabulary (OOV) words. While predefined pronunciation dictionaries are used, they often lack coverage, making G2P models essential for translating OOVs.

Recent advancements in G2P conversion have shifted from rule-based methods [27,44] and conditional and joint probabilistic models [9,12,17] to machine learning and deep learning approaches [16,26,50]. Rao et al. [39] proposed a G2P model based on a Long Short-Term Memory (LSTM) recurrent neural network as LSTMs have the flexibility of taking into consideration the full context of graphemes and transform the problem from a series of grapheme-to-phoneme conversions to a word-to-pronunciation conversion. More recently, Yolchuyeva et al. [51] proposed a novel CNN-based sequence-to-sequence architecture for G2P conversion by including an end-to-end CNN with residual connections. In [11], the authors devised a G2P model employing an encoder-decoder deep neural network and enhanced it with a near-optimal active learning strategy for pronunciation dictionary development. Transformers, which use self-attention mechanisms, have further advanced G2P by capturing long-range dependencies and outperforming traditional models in several studies [41,43,47,52–54].

Phoneme-to-Grapheme (P2G). P2G conversions play a key role in applications like speech recognition and language translation. While not central to speech recognition systems, P2G conversion has gained attention for addressing challenges such as out-of-vocabulary (OOV) word recognition. For instance, Decadt et al. [14] proposed generating phoneme strings for OOVs using a phoneme recognizer, followed by a P2G converter using TIMBL [13], a memory-based learning tool, to derive likely orthographic transcriptions. Masumura et al.

[30] enhanced end-to-end speech recognition by pre-training transformer-based models with extensive phoneme-to-grapheme data, addressing the scarcity of paired speech-to-text datasets. Given the limited focus on P2G conversion for the English language, our LSTM-P2G model offers a foundation for advancing P2G techniques and future studies.

Password Guessing. Traditional password crackers rely on dictionaries of leaked passwords combined with hard-coded mangling rules to generate guesses. Modern tools like Hashcat [2] and John the Ripper (JTR) [3] build on these techniques. Over the last two decades, more advanced approaches have emerged. Markov models, introduced by Narayanan et al. [34] and later refined by others [25], improved password guessing by modeling character probabilities. Weir et al. [49] proposed probabilistic context-free grammars (PCFG), which generates guesses in decreasing probability order and has been known as the state-of-the-art for some time. Others have expanded this work by including semantic classifications [46], or adding multiword and keyboard patterns [24].

Neural network-based models have recently gained traction in password guessing. Melicher et al. [32] utilized a simple recurrent neural network to model passwords' resistance to guessing attacks. Generative adversarial networks have also been explored [21]. Hybrid approaches have emerged, such as combining LSTMs with PCFGs. In [29], the authors compared word-level and character-level training for LSTM-based models to enhance guessing accuracy. More recently, Rando et al. [38] leveraged large language models (LLMs) to model password patterns.

3 Generating Homophones

In this section, we outline the process of generating homophones. We begin by evaluating multiple G2P models to identify the most suitable one, followed by a detailed description of our homophone generation approaches and their evaluation. The datasets used for evaluating both G2P and P2G models are as follows.

Datasets. In order to evaluate the proposed models, we used four different datasets that are frequently used by researchers in Text-to-Speech and Speech Recognition applications.

– The CMUv0.7b pronunciation dictionary [1] is a North American English dataset that we preprocessed by removing acronyms and stress symbols, naming it CMUDICT. It includes 124,643 unique words transcribed using the 39-symbol ARPAbet phonetic alphabet for American English. Some words have multiple pronunciations, all of which were used as references during evaluation. CMUDICT was used for both training and testing, divided into training (87,250), validation (31,160), and testing (6,233) subsets for comprehensive model evaluation.

- The Nettalk dataset [42] contains 20,008 English words with phonetic transcriptions. Used exclusively for testing, this test set includes 8,888 unique words not present in the CMU training set.
- The TIMIT corpus [18], originally created for acoustic-phonetic studies and speech recognition, includes recordings from 630 speakers across eight American English dialects. Its dictionary pairs words from the prompts with their pronunciations. Used exclusively for testing, excluding words from the CMU training set, this set contains 1,973 unique words.
- Buckeye [37] consists of recordings from 40 speakers engaged in casual conversations with an interviewer. It has been carefully transcribed and phonetically labeled. The test set (excluding words from the CMU training set) includes 3,222 unique words.

3.1 Finding an Effective Grapheme to Phoneme (G2P) Model

G2P models are tasked with mapping graphemes (e.g., "picture") to their corresponding phonetic representations (`/P IH K CH ER/`). Since obtaining the accurate pronunciation of a given word is the first step in generating similar-sounding words, the G2P phase is a critical component of our homophone generation process and needs to be accurate. To identify the most suitable G2P model for our system, we evaluated three available G2P models.

AttG2P [7] This model employs an attention-based encoder-decoder framework, integrating Gated Recurrent Units (GRUs) in both the encoder and decoder components. We trained this model with the CMUDICT dataset.

LG2P [5] This model also features an encoder-decoder structure but uses the LSTM model. We trained this model with CMUDICT.

Transphone [28] This model uses a diverse array of monolingual G2P pretrained models. It includes a joint n-gram model employing Weighted Finite State Transducers (WFSTs), and two sequence-to-sequence models implemented using LSTM networks and Transformer architectures.

G2P Evaluation Results The accuracy of G2P conversion is measured in terms of Phoneme Error Rate (PER) [9]. PER is defined as the *minimum edit distance* between the `reference` phoneme sequence and the `hypothesis` phoneme sequence, divided by the length of the `reference`. The PER metric effectively quantifies the error rate of the phoneme predictions by measuring the minimal number of insertions, deletions, and substitutions (Levenshtein edit distance) required to match the hypothesis with the reference. The total PER over the test set is then calculated as the sum of the Levenshtein distances between the predicted and the reference phoneme sequences, divided by the sum of the reference lengths. Another metric commonly used is the Word Error Rate (WER). In G2P applications, WER is the ratio of wrongly predicted words (words that have at least one phoneme error) over the total number of words. WER values are typically higher than PERs, as WER measures the accuracy of entire phoneme sequences and is less forgiving of minor errors. For example, with the

word "arrowhead" and a reference pronunciation of `/AE R OW HH EH D/`, if the hypothesis is `/AA R OW HH EH D/`, the PER is $\frac{1}{6}$, accounting for one incorrect phoneme out of six, while WER treats this as a single error, disregarding the five correct phonemes. Although we include WER in our results, we consider PER a more precise metric for evaluating G2P model performance.

The results of evaluating the G2P models are shown in Table 1. Across the datasets, AttG2P consistently shows lower PER and WER compared to LG2P and Transphone, indicating superior accuracy in phoneme prediction. Although the performance of different models on the CMUDICT dataset is comparable, AttG2P stands out by achieving the lowest PER and WER on other datasets. Given its superior performance across different metrics and diverse datasets, we select the AttG2P model as the primary G2P model in our proposed system for generating homophones. All training times are based on a machine with 2.4 GHz processor and 384 GB memory.

Table 1. Comparison of the G2P models on different datasets.

Models	**Datasets**								
	CMUDICT		Timit		Buckeye		Nettalk		Training
	PER	WER	PER	WER	PER	WER	PER	WER	time hh:mm
AttG2P	0.12	0.45	0.09	0.37	0.08	0.32	0.12	0.47	01:24
LG2P	0.13	0.46	0.15	0.60	0.22	0.67	0.19	0.60	00:14
Transphone	0.15	0.58	0.15	0.47	0.14	0.54	0.17	0.67	pretrained

3.2 Proposed Methods for Generating Homophones

Our model generates homophones by first converting input words (e.g., "high-five") into phonetic representations using a G2P model (e.g., /HH AY F AY V/). It then produces graphemic variations of these phonetic sequences (e.g., "hyphive", or "hifyv") using our P2G model. Generating similar-sounding words involves reversing phoneme sequences into graphemes, a complex task due to challenges like homophones (e.g.,"night" vs. "knight"). Since we aim to create misspelled homophones, traditional P2G methods for speech-to-text systems fall short. To address this, we propose two novel techniques for generating homophones, described below.

1) ProbP2G This method uses phoneme-grapheme correspondences and their probabilities to generate the most likely homophones by substituting each phoneme with its most frequent grapheme in English text. Hanna et al. [20] conducted a seminal study on mapping phonemes to graphemes, offering a rigorous methodology that remains foundational despite its age. This task is inher-

ently complex, as each phoneme can correspond to multiple graphemes, requiring disambiguation through context, such as subsequent phonemes. Hanna et al.'s study utilized 55 phonemes, while more recent datasets like CMUDICT use 39 ARPAbet phonemes. To align with current standards, we carefully consolidated and mapped the phonemes, creating a frequency and percentage table of phoneme-grapheme correspondences. The table includes the phoneme (ph_i), its corresponding grapheme (g_j), and the percentage of occurrences of each phoneme-grapheme pair relative to all other graphemic options, $P(g_j|ph_i)$, where $\sum_j P(g_j|ph_i) = 1$ (Example 1). To generate a homophone, we start by using our G2P model to extract the word's pronunciation as a sequence of phonemes (e.g., $/ph_1\ ph_2\ \ldots\ ph_t/$). Each phoneme ph_i is then substituted with its corresponding grapheme g_j from the table, prioritized by their probability $P(g_j|ph_i)$. This process generates new words in decreasing probability order, where the homophone's probability is the product of the probabilities of its constituent graphemes: $P(homophone = g_1 g_2 \ldots g_t) = \prod_{i=1}^{t} P(g_j|ph_i)$.

Example 1. For example, the phoneme `/AY/` (as in b**i**te) can map to graphemes 'i', 'y', ''igh', or 'ie' with probabilities of 0.8, 0.1, 0.07, and 0.03 respectively.

2) LSTM-P2G This model employs an encoder-decoder architecture using Long Short-Term Memory (LSTM) networks [22]. The encoder processes the input phoneme sequence $\mathbf{p} = (p_1, p_2, \ldots, p_n)$. It transforms the input into a series of hidden states $\mathbf{h} = (h_1, h_2, \ldots, h_n)$ through recursive transformations: $h_i = \text{LSTM}(h_{i-1}, p_i)$, where h_{i-1} represents the hidden state from the previous time step, and p_i is the current input feature. Upon processing the final element, the encoder yields the last hidden state h_n, encapsulating the entire input sequence's contextual information. The last hidden state serves as the initial hidden state for the decoder.

The decoder, generates the output sequence $\mathbf{g} = (g_1, g_2, \ldots, g_m)$ in a stepwise fashion. The initial step utilizes the encoder vector (and the start of sequence token as input) to produce the first output element: $g_1, h'_1 = \text{LSTM}(h_n)$. Subsequent outputs are then generated recursively, with the decoder LSTM taking its previous hidden state and the latest generated output as inputs: $g_i, h'_i = \text{LSTM}(h'_{i-1}, g_{i-1})$, where h'_i and g_i represent the decoder's hidden state and the output produced at step i, respectively. This iterative process continues until the decoder emits a special end-of-sequence token or a predefined length of the output sequence is reached. The output of this model corresponds to the grapheme sequence for the input phoneme sequence. While an ideal P2G model would reproduce the original word, our objective is to generate intentionally misspelled variants. Due to the complexity of the P2G task, intentionally halting the training process before the model reaches convergence enables us to generate similar-sounding homophones. The training time of this model was about 16 min.

Evaluation of LSTM-P2G Model To evaluate the performance of our P2G conversion model, we use Word Error Rate (WER) and Character Error Rate

(CER). WER measures the proportion of predicted words that fail to match the reference word exactly. It categorizes outcomes as either 0 or 1 (match or mismatch), with even a single character mismatch leading to the entire word being classified as incorrect. CER, in contrast, computes the Levenshtein edit distance between the predicted and the reference word, divided by the length of the reference word. While each metric may suit different applications, CER is more appropriate for this study, as our goal is to use P2G for homophone generation (See Example 2). Table 2 shows the evaluation results. CER values are low, indicating high character-level accuracy, except for Buckeye. However, WER values are higher, suggesting that more than half of the predictions do not exactly match the reference.

Example 2. Our LSTM-P2G predicts "nacks" for phoneme `/N AE K S/` while the reference is "knacks". This results in a WER of 1 due to the mismatch, but despite the high WER, the result is desirable for homophone generation.

Table 2. Evaluation results of the LSTM-P2G model.

	CMUDICT	Timit	Buckeye	Nettalk
CER	0.18	0.17	0.42	0.20
WER	0.66	0.57	0.83	0.68

Table 3. Total PER value of the two proposed homophone generation methods.

	CMUDICT	Timit	Buckeye	Nettalk
ProbP2G	0.25	0.25	0.26	0.27
LSTM-P2G	0.17	0.20	0.21	0.22

3.3 Evaluation of Homophones

For evaluating the homophones generated by both methods, we focus on assessing the phonetic similarity between the homophone and the initial word using the Phoneme Error Rate (PER) metric. For the original words, true pronunciations are provided as phonetic transcriptions in the test sets. However, the generated homophones lack true phoneme labels. We, therefore, use our G2P model to predict their pronunciations. The PER is then computed between the phoneme sequence of the original word and the predicted phonemes of the generated homophones.

Table 3 presents the total PER results for both models. A lower PER indicates greater phonetic similarity. Both models achieve strong results, validating

their effectiveness in homophone generation. However, the LSTM-P2G model consistently outperforms ProbP2G across all datasets, highlighting its potential for applications requiring precise graphemic transcription. For expanding attack dictionaries through homophone generation, each model offers unique trade-offs. LSTM-P2G produces closer phonetic matches but is limited to generating one variant per word, resulting in a smaller expanded attack dictionary. In contrast, ProbP2G generates multiple word variants, making it more effective for password cracking, as shown in the next section.

4 Password Guessing Experiments

In this section, we describe our experimental setup and present the empirical results of password cracking using homophones. We use PCFG [49] as the primary method for password generation. PCFG trains on revealed password datasets to derive a probabilistic context-free grammar by breaking passwords into components: alpha strings (L), digits (D), and special characters (S), along with their lengths and probabilities. For example, '`lettuce1980!`' is denoted as $L_7D_4S_1$. Guesses are generated in decreasing probability order by substituting components with terminal values, where the probability of a guess is the product of its components' probabilities. L components are replaced with words from the attack dictionary; if a required L component is absent, the password cannot be cracked without brute forcing.

We chose PCFG for its state-of-the-art status and ability to replace L components in base structures with a custom dictionary. Unlike general dictionaries which may include leaked passwords or mixed characters, PCFG's dictionary is limited to alphabetic words. Our homophone generation expands this dictionary with similar-sounding words, focusing exclusively on the alphabetical parts of passwords. Using PCFG ensures consistency while isolating the impact of dictionary words on password cracking. The effect of adding homophones is evident in the increased number of passwords cracked (Sect. 4.1).

Dictionaries. We use three different dictionaries for our tests, generating homophones for each word using both the ProbP2G and LSTM-P2G techniques. After removing duplicates, this produced two distinct expanded versions for each dictionary. Note that ProbP2G generates up to five homophone variations for each word, hence resulting in significantly larger dictionaries.

- **Dic1:** The English dictionary from the Ubuntu system. Dictionary sizes are: original Dic1: 99,207; Dic1_LSTMP2G: 143,947; Dic1_ProbP2G: 461,405.
- **Dic2:** The CMUDICT described before. Dictionary sizes are: original Dic2: 124,863; Dic2_LSTMP2G: 180,105; Dic2_ProbP2G: 582,979.
- **Dic3:** dic0294 [3], a commonly used dictionary for password cracking, compiled over the years by incorporating the most common strings found in leaked passwords. Dictionary sizes: original Dic3: 790,270; Dic3_LSTMP2G: 1,407,744; Dic3_ProbP2G: 3,530,192.

Datasets. We used multiple password sets, sourced from past website breaches that are commonly used by researchers. Passwords were randomly selected to ensure the test set was distinct from the training set. Our training set, called *Mixed-training*, is composed of 0.5 million from RockYou [45] (attacked in 2009), combined with 30,998 from MySpace [31] (revealed in 2006) and 4,874 from Hotmail [48] (from 2009). *Mixed-test* set contains the same number of passwords from each of the datasets mentioned, excluding passwords from Mixed-training. *Yahoo-test* contains 142,762 passwords from Yahoo [33] (2012), *RockYou-test* contains 142,762 passwords from Rockyou, *MySpace-test* contains 33,481 passwords from MySpace, and *phpBB-test* contains 255,421 passwords from phpBB [4] (2009), and 720,302 passwords from a more recent dataset 000Webhost (2015).

4.1 Result and Discussion

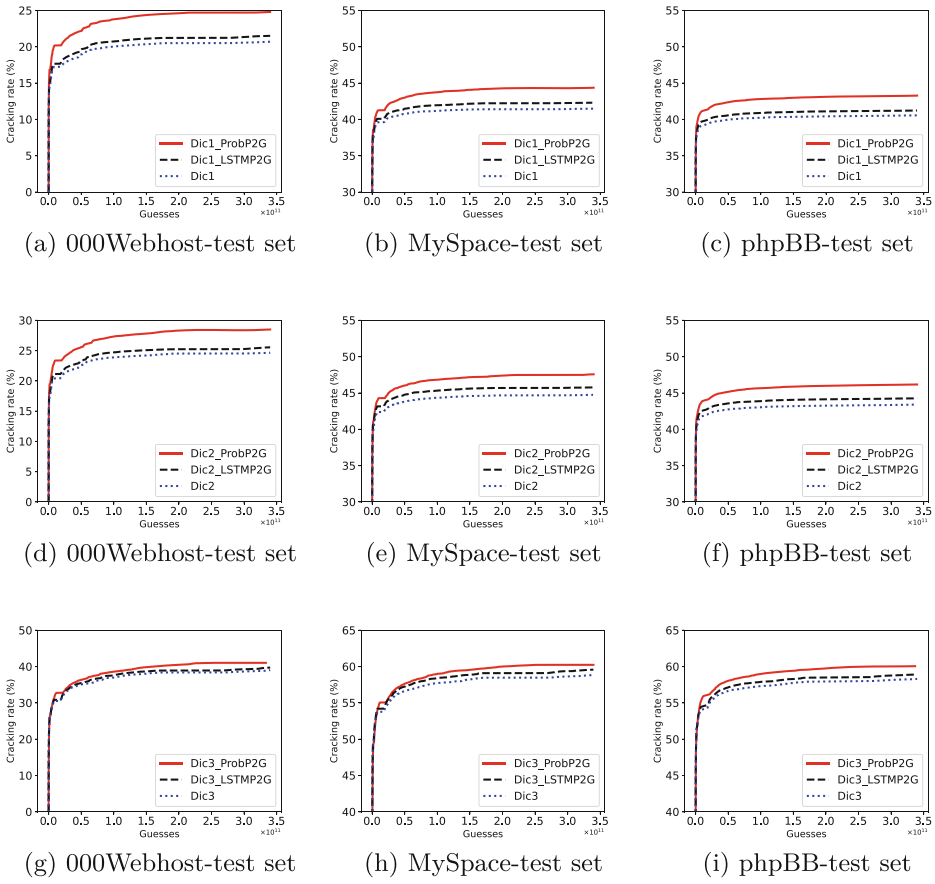
Password Cracking. We ran the PCFG password cracker with all three dictionaries and their expanded versions across multiple test sets. The number of generated guesses is limited to 340 billion in all cracking sessions. Figure 1 shows the cracking curves on three of our test sets. The graphs for other test sets were very similar, and we show the percentage improvement of those in Table 4. The graphs plot the percentage of passwords cracked on the Y-axis against the number of guesses generated on the X-axis. Overall, the results show that adding homophones to dictionaries significantly improves password cracking, with ProbP2G showing a clear advantage over LSTM-P2G due to its larger homophone selection.

Table 4 shows the percentage improvement of the ProbP2G version of each dictionary over the original. The addition of homophones consistently leads to significant improvements, with Dic1 and Dic2 benefiting far more than Dic3. This is because Dic3 is a specialized password cracking dictionary that already performs better on its own. This demonstrates that our approach of systematically generating homophonous misspellings can create effective attack dictionaries from standard English dictionaries like Dic1 and Dic2.

The improvement percentages across these datasets range from 2% to 20%. The ProbP2G dictionary's success becomes evident early in the cracking session (after about 3 billion guesses) as evident in Fig. 1. This early success is crucial when time is a critical factor. The password sets used in our experiments are well-known classical datasets often cited in password research. However, the older datasets do not reflect current password creation habits. When they were cre-

Table 4. Password cracking improvement: ProbP2G dictionary vs. original.

	000Webhost	MySpace	phpBB	Yahoo	Mixed	RockYou
Dic1	19.8%	7.0%	6.7%	4.7%	4.5%	4.4%
Dic2	15.5%	6.3%	6.4%	4.8%	4.5%	4.3%
Dic3	5.1%	2.4%	3.0%	1.9%	2.4%	2.5%



(a) 000Webhost-test set (b) MySpace-test set (c) phpBB-test set

(d) 000Webhost-test set (e) MySpace-test set (f) phpBB-test set

(g) 000Webhost-test set (h) MySpace-test set (i) phpBB-test set

Fig. 1. Result of password cracking using the three dictionaries and their respective expanded versions. Here, we show the cracking curves for 000Webhost-test, MySpace-test and phpBB-test. Figures (a)-(c) show the result of Dic1, (d)-(f) present the result for Dic2, and (g)-(i) show the result for Dic3.

ated, policies banning single dictionary words were rare, and social media slang and abbreviations were less common. This explains why our approach shows the greatest improvements (19.8%, 15.5%, and 5.1% for Dic1, Dic2, and Dic3, respectively) on the more recent 000Webhost dataset. Still, homophones appear across all datasets, and with today's password policies, incorporating homophones in cracking dictionaries is likely to yield even better results.

Data Analysis. The effect of adding homophones can also be measured by analyzing the passwords that were cracked exclusively using the expanded dictionary. For example, in the 000Webhost dataset, Dic2_ProbP2G cracked 27,626 more passwords compared to Dic2. These passwords contained 11,517 unique alpha strings, of which 10,415 were exclusively included in Dic2_ProbP2G and

not in the original Dic2. Example 3 shows a few examples of the homophones that led to successfully guessing the passwords.

In PCFG, when replacing words from the dictionary, all same-length words have equal probabilities. Adding too many words can dilute the probability of guesses for an L component, altering the guess order and potentially delaying those guesses in a cracking session. Therefore, when adding words to a dictionary, both the content and the size could significantly affect the result [24]. Simply expanding the dictionary and increasing the number of generated guesses does not guarantee better cracking. This highlights the importance of our work and the effectiveness of our generated homophones. Additionally, note that other password guessing techniques, such as neural network-based password guessers and the Markov approach, may struggle to adapt to generating passwords with misspellings without a relevant training set. In contrast, our method of constructing a dictionary with misspelled words is compatible with any dictionary-based attack and can generate misspelled words as training data for other techniques.

Example 3. For example, passwords "piramid2870", "7piramid", and "piramid123" were cracked due to the addition of **piramid** in Dic2_ProbP2G as a homophone for **pyramid**. Similarly, the password "no_palz" was cracked due to the addition of **palz** in Dic2_ProbP2G as a homophone for **pals**.

5 Conclusion

We have developed techniques to systematically generate similar-sounding misspellings using the linguistic concept of homophones, training and comparing grapheme-to-phoneme models and creating a new phoneme-to-grapheme model. While our models are trained and tested in English, our approach is adaptable to other languages. By augmenting attack dictionaries with phonetically similar misspellings, our experiments showed a significant boost in password-cracking performance. This highlights the limited security benefit of recommendations for non-standard spellings, as such strategies, while effective initially, can be soon exploited. Future work will assign probabilities to generated homophones, enabling password crackers to prioritize substitutions based on likelihood, further enhancing cracking performance.

References

1. The cmu pronouncing dictionary. http://www.speech.cs.cmu.edu/cgi-bin/cmudict
2. Hashcat: Advanced password recovery. http:/hashcat.net/hashcat
3. Openwall: John the ripper password cracker. http://www.openwall.com/john
4. Password datasets. https://wiki.skullsecurity.org/index.php/Passwords
5. Word Pronunciation Generation using LSTMs. Kaggle https://www.kaggle.com/code/quadeer15sh/word-pronunciation-generation-using-lstms/notebook
6. Archive of password recommendations (2025). https://github.com/shibba/homophone_passwords/blob/main/Password_Policies_Archive.pdf
7. Ajini, M.H.S.: https://github.com/mohamad-hasan-sohan-ajini/G2P

8. Ansari, M.Z., Ahmad, T., Khan, S., Mabood, F., Faizan, M.: Homograph language identification using machine learning techniques. In: Proceedings of International Conference on Data Science and Applications. Springer (2023)
9. Bisani, M., Ney, H.: Joint-sequence models for grapheme-to-phoneme conversion. Speech Commun. **50**(5), 434–451 (2008)
10. Brown, A.S., Bracken, E., Zoccoli, S., Douglas, K.: Generating and remembering passwords. Appl. Cogn. Psychol. **18**(6), 641–651 (2004)
11. Cao, D., Zhao, Y., Wu, L.: Near-optimal active learning for multilingual grapheme-to-phoneme conversion. Appl. Sci. **13**(16), 9408 (2023)
12. Chen, S.F., et al.: Conditional and joint models for grapheme-to-phoneme conversion. In: INTERSPEECH, pp. 2033–2036 (2003)
13. Daelemans, W., Zavrel, J., Van Der Sloot, K., Van den Bosch, A.: Timbl: Tilburg memory-based learner. Tilburg University (2004)
14. Decadt, B., Duchateau, J., Daelemans, W., Wambacq, P.: Memory-based phoneme-to-grapheme conversion: A method for dealing with out-of-vocabulary items in speech recognition. In: Computational Linguistics in the Netherlands. Brill (2002)
15. Dell'Amico, M., Filippone, M.: Monte carlo strength evaluation: Fast and reliable password checking. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15, Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2810103.2813631
16. Dong, L., Guo, Z.Q., Tan, C.H., Hu, Y.J., Jiang, Y., Ling, Z.H.: Neural grapheme-to-phoneme conversion with pre-trained grapheme models. In: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6202–6206. IEEE (2022)
17. Galescu, L., Allen, J.F.: Bi-directional conversion between graphemes and phonemes using a joint n-gram model. In: 4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis (2001)
18. Garofolo, J.S., et al.: TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1 (1993)
19. Gorman, K., Mazovetskiy, G., Nikolaev, V.: Improving homograph disambiguation with supervised machine learning. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018) (2018)
20. Hanna, P., Hanna, J., HODGES, R., RUDORF, E.: Phoneme-grapheme correspondences as cues to spelling improvement (1966)
21. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: Passgan: A deep learning approach for password guessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) Applied Cryptography and Network Security, pp. 217–237. Springer International Publishing, Cham (2019)
22. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation pp. 1735–80 (12 1997)
23. Houshmand, S., Aggarwal, S.: Building better passwords using probabilistic techniques. In: Proceedings of the 28th Annual Computer Security Applications Conference. ACM (2012)
24. Houshmand, S., Aggarwal, S., Flood, R.: Next gen pcfg password cracking. IEEE Trans. Inf. Forensics Secur. **10**(8), 1776–1791 (2015)
25. Jiang, J., Zhou, A., Liu, L., Zhang, L.: Omecdn: a password-generation model based on an ordered markov enumerator and critic discriminant network. Appl. Sci. **12**(23), 12379 (2022). https://doi.org/10.3390/app122312379
26. Jyothi, P., Hasegawa-Johnson, M.: Low-resource grapheme-to-phoneme conversion using recurrent neural networks. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5030–5034. IEEE (2017)

27. Kłosowski, P.: A rule-based grapheme-to-phoneme conversion system. Appl. Sci. **12**(5), 2758 (2022)
28. Li, X., Metze, F., Mortensen, D., Watanabe, S., Black, A.: Zero-shot learning for grapheme to phoneme conversion with language ensemble. In: Muresan, S., Nakov, P., Villavicencio, A. (eds.) Findings of the Association for Computational Linguistics: ACL, pp. 2106–2115. Association for Computational Linguistics, Dublin (2022).https://doi.org/10.18653/v1/2022.findings-acl.166
29. Liu, Y., et al.: Genpass: A general deep learning model for password guessing with pcfg rules and adversarial generation. In: 2018 IEEE International Conference on Communications (ICC) (2018)
30. Masumura, R., Makishima, N., Ihori, M., Takashima, A., Tanaka, T., Orihashi, S.: Phoneme-to-grapheme conversion based large-scale pre-training for end-to-end automatic speech recognition. In: INTERSPEECH (2020)
31. McMillan., R.: Phishing attack targets myspace users. http://www.infoworld.com/d/security-central/phishing-attack-targets-myspace-users-614
32. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: 25th USENIX Security Symposium. USENIX Association (2016)
33. Musil, S.: Yahoo credentials (2012). https://www.cnet.com/news/privacy/hackers-post-450k-credentials-pilfered-from-yahoo/
34. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: Proceedings of the 12th ACM Conference on Computer and Communications Security CCS '05 (2005)
35. Nicolis, M., Klimkov, V.: Homograph disambiguation with contextual word embeddings for tts systems. In: Interspeech Workshop on Speech Synthesis (2021)
36. Pasandi, H.B., Pasandi, H.B.: Evaluation of asr systems for conversational speech: a linguistic perspective. In: Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22) (2023)
37. Pitt, M., et al.: Buckeye corpus of conversational speech. In: Department of Psychology, Ohio State University (Distributor) (2007). https://buckeyecorpus.osu.edu/
38. Rando, J., Perez-Cruz, F., Hitaj, B.: Passgpt: Password modeling and (guided) generation with large language models. In: Computer Security – ESORICS 2023, pp. 164–183. Springer, Cham (2024)
39. Rao, K., Peng, F., Sak, H., Beaufays, F.: Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2015)
40. Rena Nemoto, I.V., Adda-Decker, M.: Speech errors on frequently observed homophones in french: Perceptual evaluation vs automatic classification. In: Proceedings of the International Conference on Language Resources and Evaluation (2008)
41. Řezáčková, M., Švec, J., Tihelka, D.: T5g2p: Using text-to-text transfer transformer for grapheme-to-phoneme conversion (2021)
42. Sejnowski, T., Rosenberg, C.: Connectionist Bench (Nettalk Corpus). UCI Machine Learning Repository (1954). https://doi.org/10.24432/C5VP6T
43. Sevinj, Y., Géza, N., Bálint, G.T.: Transformer based grapheme-to-phoneme conversion. Proc. Interspeech **2019**, 2095–2099 (2019)
44. Torstensson, N.: Grapheme-to-phoneme conversion, a knowledge-based approach. Speech Music and Hearing TMH-QPSR-Fonetik **44**, 117–120 (2002)
45. Vance, A.: If your password is 123456, just make it hackme. http://www.nytimes.com/2010/01/21/technology/21password.html

46. Veras, R., Collins, C., Thorpe, J.: On semantic patterns of passwords and their security impact. In: 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego (2014)
47. Vesik, K., Abdul-Mageed, M., Silfverberg, M.: One model to pronounce them all: Multilingual grapheme-to-phoneme conversion with a transformer ensemble. In: Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology (2020)
48. Warren., T.: Thousands of hotmail passwords leaked. http://www.neowin.net/news/main/09/10/05/thousands-of-hotmail-passwords-leaked-online
49. Weir, M., Aggarwal, S., Medeiros, B.d., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 391–405 (2009). https://doi.org/10.1109/SP.2009.8
50. Yao, K., Zweig, G.: Sequence-to-sequence neural net models for grapheme-to-phoneme conversion arXiv preprint arXiv:1506.00196 (2015)
51. Yolchuyeva, S., Németh, G., Gyires-Tóth, B.: Grapheme-to-phoneme conversion with convolutional neural networks. Appl. Sci. **9**(6), 1143 (2019)
52. Yolchuyeva, S., Németh, G., Gyires-Tóth, B.: Transformer based grapheme-to-phoneme conversion. arXiv preprint arXiv:2004.06338 (2020)
53. Yu, M., et al: Multilingual grapheme-to-phoneme conversion with byte representation. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8234–8238. IEEE (2020)
54. Zhu, J., Zhang, C., Jurgens, D.: Byt5 model for massively multilingual grapheme-to-phoneme conversion. arXiv preprint arXiv:2204.03067 (2022)