Interpretable Probabilistic Password Strength Meters via Deep Learning

Dario Pasquini
Stevens Institute of Technology, USA
Sapienza University of Rome, Italy
Institute of Applied Computing, CNR, Italy

Giuseppe Ateniese
Stevens Institute of Technology, USA

Massimo Bernaschi
Institute of Applied Computing, CNR, Italy

Abstract

Probabilistic password strength meters have been proved to be the most accurate tools to measure password strength. Unfortunately, by construction, they are limited to solely produce an opaque security estimation that fails to fully support the user during the password composition. In the present work, we move the first steps towards cracking the intelligibility barrier of this compelling class of meters. We show that probabilistic password meters inherently own the capability to describe the latent relation between password strength and password structure. In our approach, the security contribution of each character composing a password is disentangled and used to provide explicit fine-grained feedback for the user. Furthermore, unlike existing heuristic constructions, our method is free from any human bias, and, more importantly, its feedback has a probabilistic interpretation.
In our contribution: (1) we formulate interpretable probabilistic password strength meters; (2) we describe how they can be implemented via an efficient and lightweight deep learning framework suitable for client-side operability.

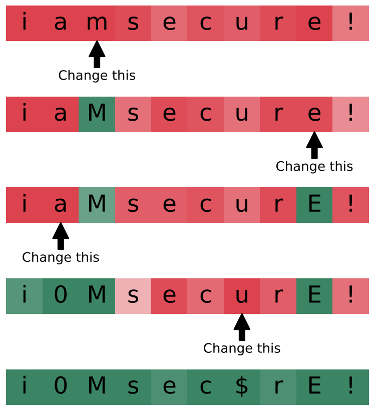


Figure 1: Example of the character-level feedback mechanism and password composition process induced by our meter. In the figure, "*iamsecure!*" is the password initially chosen by the user. Colors indicate the estimated character security: red (insecure) → green (secure).

1 Introduction

Accurately measuring password strength is essential to guarantee the security of password-based authentication systems. Even more critical, however, is training users to select secure passwords in the first place. One common approach is to rely on password policies that list a series of requirements for a strong password. This approach is limited or even harmful [10]. Alternatively, Passwords Strength Meters (PSMs) have been shown to be useful and are witnessing increasing adoption in commercial solutions [15, 26].
The first instantiations of PSMs were based on simple heuristic constructions. Password strength was estimated via either handcrafted features such as *LUDS* (which counts lower and uppercase letters, digits, and symbols) or heuristic entropy definitions. Unavoidably, given their heuristic nature, this class of PSMs failed to accurately measure password security [11, 30]. More recently, thanks to an active academic interest, PSMs based on more sound constructions and rigorous security definitions have been proposed. In the last decade, indeed, a considerable research effort gave rise to more precise meters capable of accurately measuring password strength [9, 20, 28]. However, meters have also become proportionally more opaque and inherently hard to interpret due to the increasing complexity of the employed approaches. State-of-the-art solutions base their estimates on blackbox parametric probabilistic models [9, 20] that leave no room for interpretation of the evaluated passwords; they do not provide any feedback to users on what is wrong with their password or how to improve it. We advocate for explainable approaches in password meters, where users receive additional insights and become

[1]An abridged version of this paper appears in the proceedings of the 25th European Symposium on Research in Computer Security (ESORICS) 2020.

cognizant of which parts of their passwords could straightforwardly improve. This makes the password selection process less painful since users can keep their passwords of choice mostly unchanged while ensuring they are secure.

In the present work, we show that the same rigorous probabilistic framework capable of accurately measuring password strength can also fundamentally describe the relation between password security and password structure. By rethinking the underlying mass estimation process, we create the first *interpretable probabilistic password strength* meter. Here, the password probability measured by our meter can be decomposed and used to estimate further the strength of every single character of the password. This explainable approach allows us to assign a security score to each atomic component of the password and determine its contribution to the overall security strength. This evaluation is, in turn, returned to the user who can tweak a few "weak" characters and consistently improve the password strength against guessing attacks. Figure 1 illustrates the selection process. In devising the proposed mass estimation process, we found it ideally suited for being implemented via a deep learning architecture. In the paper, we show how that can be cast as an efficient client-side meter employing deep convolutional neural networks. Our work's major contributions are: (i) We formulate a novel password probability estimation framework based on undirected probabilistic models. (ii) We show that such a framework can be used to build a precise and sound password feedback mechanism. (iii) We implement the proposed meter via an efficient and lightweight deep learning framework ideally suited for client-side operability.

2 Background and preliminaries

In this section, we offer an overview of the fundamental concepts that are important to understand our contribution. Section 2.1 covers Probabilistic Password Strength Meters. Next, in Section 2.2, we cover structured probabilistic models that will be fundamental in the interpretation of our approach. Finally, Section 2.3 briefly discusses relevant previous works within the PSMs context.

2.1 Probabilistic Password Strength Meters (PPSMs)

Probabilistic password strength meters are PSMs that base their strength measure on an explicit estimate of password probability. In the process, they resort to probabilistic models to approximate the probability distribution behind a set of known passwords, typically, instances of a password leak. Having an approximation of the mass function, strength estimation is then derived by leveraging adversarial reasoning. Here, password robustness is estimated in consideration of an attacker who knows the underlying password distribution, and that aims at minimizing the guess entropy [19] of her/his guessing attack. To that purpose, the attacker performs an optimal guessing attack, where guesses are issued in decreasing probability order (i.e., high-probability passwords first). More formally, given a probability mass function $P(\mathbf{x})$ defined on the key-space $\mathbb{X}$, the attacker creates an ordering $\mathbb{X}_{P(\mathbf{x})}$ of $\mathbb{X}$ such that:

$$\mathbb{X}_{P(\mathbf{x})} = [x^0, x^1, \ldots, x^n] \quad \text{where} \quad \forall_{i \in [0,n]} : P(x^i) \geq P(x^{i+1}) \quad . \tag{1}$$

During the attack, the adversary produces guesses by traversing the list $\mathbb{X}_{P(\mathbf{x})}$. Under this adversarial model, passwords with high probability are considered weak, as they will be quickly guessed. Low-probability passwords, instead, are assessed as secure, as they will be matched by the attacker only after a considerable, possibly not feasible, number of guesses.

2.2 Structured Probabilistic Models

Generally, the probabilistic models used by PPSMs are **probabilistic structured models** (even known as graphical models). These describe password distributions by leveraging a **graph notation** to illustrate the dependency properties among a set of random variables. Here, a random variable $\mathbf{x}_i$ is depicted as a vertex, and an edge between $\mathbf{x}_i$ and $\mathbf{x}_j$ exists whether $\mathbf{x}_i$ and $\mathbf{x}_j$ are statistically dependent. Structured probabilistic models are classified according to the orientation of edges. A direct acyclic graph (DAG) defines a **directed graphical model** (or Bayesian Network). In this formalism, an edge asserts a cause-effect relationship between two variables; that is, the state assumed from the variable x_i is intended as a direct consequence of those assumed by its parents $par(\mathbf{x}_i)$ in the graph. Under such a description, a topological ordering among all the random variables can be asserted and used to factorize the joint probability distribution of the random variables effectively. On the other hand, an undirected graph defines an **undirected graphical model**, also known as Markov Random Field (MRF). In this description, the topological order among edges is relaxed, and connected variables influence each other symmetrically. However, this comes at the cost of giving up to any simple form of factorization of the joint distribution.

2.3 Related Works

Here, we briefly review early approaches to the definition of PSMs. We focus on the most influential works as well as to the ones most related to ours.

Probabilistic PSMs: Originally thought for guessing attacks [21], Markov model approaches have found natural application in the password strength estimation context. Castelluccia et al. [9] use a stationary, finite-state Markov chain as a direct password mass estimator. Their model computes the joint probability by separately measuring the conditional probability of each pair of n-grams in the observed passwords. Melicher et al. [20] extended the Markov model approach by leveraging a character/token level Recurrent Neural Network (RNN) for modeling the probability of passwords. As discussed in the introduction, pure probabilistic approaches are not capable of any natural form of feedback. In order to partially cope with this shortcoming, a hybrid approach has been investigated in [24]. Here, the model of Melicher et al. [20] is aggregated with a series of 21 heuristic, hand-crafted feedback mechanisms such as detection of *leeting behaviors* or common tokens (e.g., keyboard walks).

Even if harnessing a consistently different form of feedback, our framework merges these solutions into a single and jointly learned model. Additionally, in contrast with [24], our feedback has a concrete probabilistic interpretation as well as complete freedom from any form of human bias. Interestingly enough, our model autonomously learns some of the heuristics hardwired in [24]. For instance, our model learned that capitalizing characters in the middle of the string could consistently improve password strength.

Token look-up PSMs: Another relevant class of meters is that based on the token look-up approach. Generally speaking, these are non-parametric solutions that base their strength estimation on collections of sorted lists of tokens like leaked passwords and word dictionaries. Here, a password is modeled as a combination of tokens, and the relative security score is derived from the ranking of the tokens in the known dictionaries. Unlike probabilistic solutions, token-based PSMs are able to return feedback to the user, such as an explanation for the weakness of a password relying on the semantic attributed to the tokens composing the password. A leading member of token look-up meters is *zxcvbn* [31], which assumes a password as a combination of tokens such as *token, reversed, sequence repeat, keyboard, and date*. This meter scores passwords according to a heuristic characterization of the guess-number [19]. Such score is described as the number of combinations of tokens necessary to match the tested password by traversing the sorted tokens lists.

zxcvbn is capable of feedback. For instance, if one of the password components is identified as *"repeat"*, *zxcvbn* will recommend the user to avoid the use of repeated characters in the password. Naturally, this kind of feedback mechanism inherently lacks generality and addresses just a few human-chosen scenarios. As discussed by the authors themselves, *zxcvbn* suffers from various limitations. By assumption, it is unable to model the relationships among different patterns occurring in the same passwords. Additionally, like other token look-up based approaches, it fails to coherently model unobserved patterns and tokens.

Another example of token look-up approach is the one proposed in [17]. *Telepathwords* discourages a user from choosing weak passwords by predicting the next most probable characters during the password typing. In particular, predicted characters are shown to the user in order to dissuade him/her from choosing them as the next characters in the password. These are reported together with an explanation of why those characters were predicted. However, as for *zxcvbn*, such feedback solely relies on hardwired scenarios (for instance, the use of profanity in the password). *Telepathwords* is server-side only.

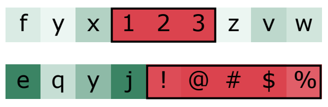
3 Meter foundations

In this section, we introduce the theoretical intuition behind the proposed meter. First, in Section 3.1, we introduce and motivate the probabilistic character-level feedback mechanism. Later, in Section 3.2, we describe how that mechanism can be obtained using undirected probabilistic models.

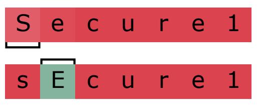
3.1 Character-level strength estimation via probabilistic reasoning

As introduced in Section 2.1, PPSMs employ probabilistic models to approximate the probability mass function of an observed password distribution, say $P(\mathbf{x})$. Estimating $P(\mathbf{x})$, however, could be particularly challenging, and suitable estimation techniques must be adopted to make the process feasible. In this direction, a general solution is to factorize the domain of the mass function (i.e., the key-space); that is, passwords are modeled as a concatenation of smaller factors, typically, decomposed at the character level. Afterward, password distribution is estimated by modeling stochastic interactions among these simpler components. More formally, every password is assumed as a realization $x = [x_1, \dots, x_\ell]$ of a random vector of the kind $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_\ell]$, where each disjoint random variable $\mathbf{x}_i$ represents the character at position i in the string. Then, $P(\mathbf{x})$ is described through **structured probabilistic models** that formalize the relations among those random variables, eventually defining a joint probability distribution. In the process, every random variable is associated with a **local conditional probability distribution** (here, referred to as Q) that describes the stochastic behavior of $\mathbf{x}_i$ in consideration of the conditional independence properties asserted from the underlying structured model i.e., $Q(\mathbf{x}_i) = P(\mathbf{x}_i \mid par(\mathbf{x}_i))$. Eventually, the joint measurement of probability is derived from the aggregation of the marginalized local conditional probability distributions, typically under the form $P(\mathbf{x}) = \prod_{i=1}^{\ell} Q(\mathbf{x}_i = x_i)$.

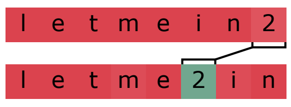
As introduced in Section 2.1, the joint probability can be employed as a good representative for password strength. However, such a global assessment unavoidably hides much fine-grained information that can be extremely valuable to a password meter. In particular, the joint probability offers us an atomic interpretation of the password strength, but it fails at disentangling the relation between password strength and password structure. That is, it does not clarify which factors



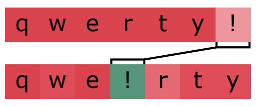
(a) Common tokens.



(b) Capitalize first/inner.



(c) Numeric last/inner.

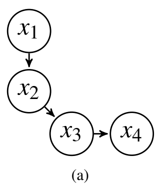


(d) Special last/inner.

Figure 2: In Panel (a), the model automatically highlights the presence of weak substrings by assigning high probabilities to the characters composing them. Panels (b), (c), and (d) are examples of self-learned weak/strong password composition patterns. In panel (b), the model assigns a high probability to the capitalization of the first letter (a common practice), whereas it assigns low probability when the capitalization is performed on inner characters. Panel (c) and (d) report similar results for numeric and special characters.

of an evaluated password are making that password insecure. However, as widely demonstrated by non-probabilistic approaches [17, 24, 31], users benefit from the awareness of which part of the chosen password is easily predictable and which is not. In this direction, we argue that the **local conditional probabilities** that naturally appear in the estimation of the joint one, if correctly shaped, can offer detailed insights into the strength or the weakness of each factor of a password. **Such character-level probability assignments are an explicit interpretation of the relation between the structure of a password and its security.** The main intuition here is that: high values of $Q(x_i)$ tell us that x_i (i.e., the character at position i in the string) has a high impact on increasing the password probability and must be changed to make the password stronger. Instead, characters with low conditional probability are pushing the password to have low probability and must be maintained unchanged. Figure 2 reports some visual representations of such probabilistic reasoning. Each segment's background color renders the value of the local conditional probability of the character. Red describes high probability values, whereas green describes low probability assignments. Such a mechanism can naturally discover weak passwords components and explicitly guide the user to explore alternatives. For instance, local conditional probabilities can spot the presence of predictable tokens in the password without the explicit use of dictionaries (Figure 2a). These measurements are able to automatically describe common password patterns like those manually modeled from other approaches [24], see Figures 2b, 2c and 2d. More importantly, they can potentially describe latent composition patterns that have never been observed and modeled by human beings. In doing this, neither supervision nor human-reasoning is required.

Unfortunately, existing PPSMs, by construction, leverage arbitrary designed structured probabilistic models that make inefficient to produce the required estimates. Hereafter, we show that reshaping the mass estimation process will allow us to implement the feedback mechanism described above. To that purpose, we have to build a new probabilistic estimation framework and simulate a complete, undirected models.



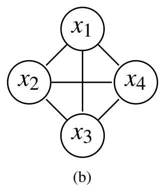


Figure 3: Two graphical models describing different interpretations of the generative probability distribution for passwords of length four. Graph (a) represents a Bayesian network. Scheme (b) depicts a Markov Random Field.

3.2 An undirected description of password distribution

To simplify our method's understanding, we start with a description of the probabilistic reasoning of previous approaches. Then, we fully motivate our solution by comparison with them. In particular, we chose the state-of-the-art neural approach proposed in [20] (henceforth, referred to as FLA) as a representative instance, since it is the least biased as well as the most accurate among the existing PPSMs.
FLA uses a recurrent neural network (RNN) to estimate password mass function at the character level. That model is **autoregressive** and assumes a stochastic process represented by a Bayesian network like the one depicted in Figure 3a. The description derived from a Bayesian Network implies a topological order among password characters. Here, characters influence their local conditional probabilities only asymmetrically; that is, the probability of $\mathbf{x}_{i+1}$ is conditioned by $\mathbf{x}_i$ but not *vice versa*. In practice, for the local conditional probabilities, this implies that the observation of the value assumed from $\mathbf{x}_{i+1}$ does not affect our belief in the value expected from $\mathbf{x}_i$, yet the opposite does. The autoregressive nature of this model eventually simplifies the joint probability estimation; the local conditional probability of each character can be easily computed as $Q(\mathbf{x}_i) = P(\mathbf{x}_i \mid \mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$, where $Q(\mathbf{x}_i)$ explicates that the i'th character solely depends on the characters that precede it in the string. Just as easily, the joint

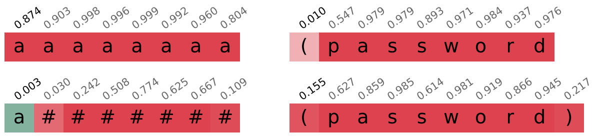
probability factorizes in:

$$P(\mathbf{x}) = \prod_{i=1}^{\ell} Q(\mathbf{x}_i) = P(\mathbf{x}_1)\prod_{i=2}^{\ell} P(\mathbf{x}_i \mid \mathbf{x}_1, \dots \mathbf{x}_{i-1})$$

by chain rule.

Unfortunately, although this approach does simplify the estimation process, the conditional probability $Q(x_i)$, *per se*, does not provide a direct and coherent estimation of the security contribution of the single character x_i in the password. This is particularly true for characters in the first positions of the string, even more so for the first character x_1, which is assumed to be independent of any other symbol in the password; its probability is the same for any possible configuration of the remaining random variables $[x_2, \dots, x_\ell]$. Nevertheless, in the context of a **sound** character-level feedback mechanism, the symbol x_i must be defined as "weak" or "strong" according to the complete context defined by the entire string. For instance, given two passwords y="*aaaaaaa*" and z="*a######*", the probability $Q(\mathbf{x}_1 = \text{'}a\text{'})$ should be different if measured on y or z. More precisely, we expect $Q(\mathbf{x}_1 = \text{'}a\text{'}|y)$ to be much higher than $Q(\mathbf{x}_1 = \text{'}a\text{'}|z)$, as observing $y_{2,7}$="*aaaaaa*" drastically changes our expectations about the possible values assumed from the first character in the string. On the other hand, observing $z_{2,7}$="*######*" tells us little about the event $\mathbf{x}_1 = \text{'}a\text{'}$. Yet, this interaction cannot be described through the Bayesian network reported in Figure 3a, where $Q(\mathbf{x}_1 = \text{'}a\text{'}|y)$ eventually results equal to $Q(\mathbf{x}_1 = \text{'}a\text{'}|z)$. The same reasoning applies to trickier cases, as for the password x="*(password)*". Here, arguably, the security contribution of the first character '*(*' strongly depends from the presence or absence of the last character[1], i.e., x_7 = '*)*'. The symbol x_1 = '*(*', indeed, can be either a good choice (as it introduces entropy in the password) or a poor one (as it implies a predictable template in the password), but this solely depends on the value assumed from another character in the string (the last one in this example). It should be apparent that the autoregressive nature of the model prevents the resulting local conditional probabilities from being sound descriptors of the real character probability as well as of their security contribution. Consequently, such measures cannot be used to build the feedback mechanism suggested in Section 3.1. The same conclusion applies to other classes of PPSMs [9, 29] which add even more structural biases on top of those illustrated by the model in Figure 3a.

Differently, we base our estimation on an **undirected and complete**[2] **graphical model**, as this represents a handier description of the password generative distribution. Figure 3b depicts the respective Markov Random Field (MRF) for passwords of length four. According to that description, the local conditional probability of the character x_i directly depends on any other character in the string, i.e., the full context. In other words, we model each variable $\mathbf{x}_i$ as a stochastic function of all the others. This intuition is better captured from the evaluation of local conditional probability (Eq. 2).



(a) (b)

Figure 4: Estimated local conditional probabilities for two pairs of passwords. The numbers depicted above the strings report the $Q(x_i)$ value for each character (rounding applied).

$$Q(\mathbf{x}_i) = \begin{cases} P(\mathbf{x}_i \mid \mathbf{x}_{i+1}, \dots \mathbf{x}_\ell) & i = 1 \\ P(\mathbf{x}_i \mid \mathbf{x}_1, \dots \mathbf{x}_{i-1}) & i = \ell \\ P(\mathbf{x}_i \mid \mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots \mathbf{x}_\ell) & 1 < i < \ell \end{cases} . \tag{2}$$

Henceforth, we use the notation $Q(\mathbf{x}_i)$ to refer to the local conditional distribution of the i'th character within the password x. When x is not clear from the context, we write $Q(\mathbf{x}_i \mid x)$ to make it explicit. The notation $Q(\mathbf{x}_i = s)$ or $Q(s)$, instead, refers to the marginalization of the distribution according to the symbol s.

Eventually, such undirected formalization intrinsically weeds out all the limitations observed for the previous estimation process (i.e., the Bayesian network in Figure 3a). Now, every local measurement is computed within the context offered by any other symbol in the string. In the example, y="*aaaaaaa*" / z="*a######*", indeed, the local conditional probability of the first character can be now backward-influenced from the context offered from the subsequent part of the string. This is clearly observable from the output of an instance of our meter reported in Figure 4a, where the value of $Q(\mathbf{x}_1 = \text{'}a\text{'})$ drastically varies between the two cases, i.e., y and z. As expected, we have $Q(\mathbf{x}_1 = \text{'}a\text{'}|y) \gg Q(\mathbf{x}_1 = \text{'}a\text{'}|z)$ verified in the example. A similar intuitive result is reported in Figure 4b, where the example x="*(password)*" is considered. Here, the meter first scores the string x'="*(password* ", then it scores the complete password x="*(password)*". In this case, we expect that the presence of the last character '*)*' would consistently influence the conditional measurement of the first bracket in the string. Such expectation is perfectly captured from the reported output, where appending at the end of the string the symbol '*)*' increases the probability of the first bracket of a factor ~ 15.

However, obtaining these improvements does not come for free. Indeed, under the MRF construction, the productory over the local conditional probabilities (better defined

[1] Even if not so common, strings enclosed among brackets or other special characters often appear in password leaks.

[2] "Complete" in the graph-theory sense.

as potential functions or factors within this context) does not provide the exact joint probability distribution of $\mathbf{x}$. Instead, such product results in a unnormalized version of it: $P(\mathbf{x}) \propto \prod_{i=1}^{\ell} Q(\mathbf{x}_i) = \tilde{P}(\mathbf{x})$ with $P(x) = \frac{\tilde{P}(\mathbf{x})}{Z}$. In the equation, Z is the partition function. This result follows from the Hammersley–Clifford theorem [16]. Nevertheless, the unnormalized joint distribution preserves the core properties needed to the meter functionality. Most importantly, we have that:

$$\forall x, x' \; : \; P(x) \geq P(x') \Leftrightarrow \tilde{P}(x) \geq \tilde{P}(x') \quad . \tag{3}$$

That is, if we sort a list of passwords according to the true joint $P(\mathbf{x})$ or according to the unnormalized version $\tilde{P}(\mathbf{x})$, we obtain the same identical ordering. Consequently, no deviation from the adversarial interpretation of PPSMs described in Section 2.1 is implied. Indeed, we have $\mathbb{X}_{P(\mathbf{x})} = \mathbb{X}_{\tilde{P}(\mathbf{x})}$ for every password distribution, key-space, and suitable sorting function. Furthermore, the joint probability distribution, if needed, can be approximated using suitable approximation methods, as discussed in Appendix A. Alternatively, we can use the energy-based models framework to capture the proposed procedure.

It is important to highlight that, although we present our approach under the character-level description, our method can be directly applied to n-grams or tokens without any modification.

3.2.1 Details on the password feedback mechanism and further applications

Joint probability can be understood as a compatibility score assigned to a specific configuration of the MRF; it tells us the likelihood of observing a sequence of characters during the interaction with the password generative process. On a smaller scale, a local conditional probability measures the impact that a single character has in the final security score. Namely, it indicates how much the character contributes to the probability of observing a certain password x. Within this interpretation, low-probabilities characters push the joint probability of x to be closer to zero (secure), whereas high-probability characters (i.e., $Q(x_1) \lesssim 1$) make no significant contribution to lowering the password probability (insecure). Therefore, users can strengthen their candidate passwords by substituting high-probability characters with suitable lower-probability ones (e.g., Figure 1).
Unfortunately, users' perception of password security has been shown to be generally erroneous [25], and, without explicit guidelines, it would be difficult for them to select suitable lower-probability substitutes. To address this limitation, one could adopt our approach based on local conditional distributions as an effective mechanism to help users select secure substitute symbols. Indeed, $\forall_i Q(\mathbf{x}_i)$ are able to clarify which symbol is a secure substitute and which is not for each character x_i of x. In particular, a distribution $Q(\mathbf{x}_i)$, defined on the whole alphabet Σ, assigns a probability to every symbol s that the character $\mathbf{x}_i$ can potentially assume. For a symbol $s \in \Sigma$, the probability $Q(\mathbf{x}_i = s)$ measures how much the event $\mathbf{x}_i = s$ is probable given all the observable characters in x. Under this interpretation, a candidate, secure substitution of x_i is a symbol with very low $Q(\mathbf{x}_i = s)$ (as this will lower the joint probability of x). In particular, every symbol s s.t. $Q(\mathbf{x}_i = s) < Q(\mathbf{x}_i = x_i)$ given x is a secure substitution for x_i. Table 1 better depicts this intuition. The Table reports the alphabet sorted by $Q(\mathbf{x}_i)$ for each x_i in the example password $x =$ "*PaSsW0rD!*". The bold symbols between parenthesis indicate x_i. Within this representation, all the symbols below the respective x_i for each $\mathbf{x}_i$ are suitable substitutions that improve password strength. This intuition is empirically proven in Section 5.2. It is important to note that the suggestion mechanism must be randomized to avoid any bias in the final password distribution.[3] To this end, one can provide the user with k random symbols among the pool of secure substitutions, i.e., $\{s \mid Q(\mathbf{x}_i = s) < Q(\mathbf{x}_i = x_i)\}$.

Table 1: First seven entries of the ordering imposed on Σ from the local conditional distribution for each character of the password $x =$ "*PaSsW0rD!*"

Rank	•aSsW0rD!	P•SsW0rD!	Pa•sW0rD!	PaS•W0rD!	PaSs•0rD!	PaSsW•rD!	PaSsW0•D!	PaSsW0r•!	PaSsW0rD•
0	**{P}**	A	s	S	w	O	R	d	1
1	S	**{a}**	**{S}**	**{s}**	**{W}**	o	**{r}**	**{D}**	S
2	p	@	c	A	#	**{0}**	N	t	2
3	B	3	n	T	f	I	0	m	s
4	C	4	t	E	k	i	L	l	3
5	M	I	d	H	1	#	D	k	**{!}**
6	1	1	r	O	F	A	n	e	5
7	c	5	x	$	3	@	X	r	9
8	s	0	$	I	0	)	S	f	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

In summary, in this section, we presented and motivated an estimation process able to unravel the feedback mechanism described in Section 3.1. Maintaining a purely theoretical focus, no information about the implementation of such methodology has been offered to the reader. Next, in Section 4, we describe how such a meter can be shaped via an efficient deep learning framework.

4 Meter implementation

In this section, we present a deep-learning-based implementation of the estimation process introduced in Section 3.2. Here, we describe the model and its training process. Then, we explain how the trained network can be used as a building block for the proposed password meter.

Model training. From the discussion in Section 3.2, our procedure requires the parametrization of an exponentially

[3] That is, if weak passwords are always perturbed in the same way, they will be easily guessed.

large number of interactions among random variables. Thus, any tabular approach, such as the one used from Markov Chains or PCFG [29], is *a priori* excluded for any real-world case. To make such a meter feasible, we reformulate the underlying estimation process so that it can be approximated with a neural network. In our approach, we simulate the Markov Random Field described in Section 3.2 using a deep convolutional neural network trained to compute $Q(\mathbf{x}_i)$ (Eq. 2) for each possible configuration of the structured model. In doing so, we train our network to solve an *inpainting*-like task defined over the textual domain. Broadly speaking, inpainting is the task of reconstructing missing information from mangled inputs, mostly images with missing or damaged patches [32]. **Under the probabilistic perspective, the model is asked to return a probability distribution over all the unobserved elements of x, explicitly measuring the conditional probability of those concerning the observable context**. Therefore, the network has to disentangle and model the semantic relation among all the factors describing the data (e.g., characters in a string) to reconstruct input instances correctly.
Generally, the architecture and the training process used for inpainting tasks resemble an auto-encoding structure [8]. In the general case, these models are trained to revert self-induced damage carried out on instances of a train-set $\mathbf{X}$. At each training step, an instance $x \in \mathbf{X}$ is artificially mangled with an information-destructive transformation to create a mangled variation $\tilde{x}$. Then, the network, receiving $\tilde{x}$ as input, is optimized to produce an output that most resembles the original x; that is, the network is trained to reconstruct x from $\tilde{x}$.
In our approach, we train a network to infer missing characters in a mangled password. In particular, we iterate over a password leak (i.e., our train-set) by creating mangled passwords and train the network to recover them. The mangling operation is performed by removing a randomly selected character from the string. For example, the train-set entry x=“*iloveyou*” is transformed in $\tilde{x}$=“ilov•you" if the 5’th character is selected for deletion, where the symbol ‘•’ represents the *“empty character"*. A compatible proxy-task has been previously used in [22] to learn a suitable password representation for guessing attacks.

We chose to model our network with a deep *residual* structure arranged to create an autoencoder. The network follows the same general Context Encoder [23] architecture defined in [22] with some modifications. To create an information bottleneck, the encoder connects with the decoder through a latent space junction obtained through two fully connected layers. We observed that enforcing a latent space, and a prior on that, consistently increases the meter effectiveness. For that reason, we maintained the same regularization proposed in [22]; a maximum mean discrepancy regularization that forces a standard normal distributed latent space. The final loss function of our model is reported in Eq. 4. In the equation, *Enc* and *Dec* refer to the encoder and decoder network respectively, s is the *softmax* function applied row-wise[4], the distance function d is the cross-entropy, and *mmd* refers to the *maximum mean discrepancy*.

$$\mathbb{E}_{x,\tilde{x}}[d(x,\ s(Dec(Enc(\tilde{x})))] + \alpha\mathbb{E}_{z\sim N(0,\mathbb{I})}[mmd(z, Enc(\tilde{x}))] \tag{4}$$

Henceforth, we refer to the composition of the encoder and the decoder as $f(x) = s(Dec(Enc(x)))$. We train the model on the widely adopted *RockYou* leak [7] considering an 80/20 train-test split. From it, we filter passwords presenting fewer than 5 characters. We train different networks considering different maximum password lengths, namely, 16, 20, and 30. In our experiments, we report results obtained with the model trained on a maximum length equal to 16, as no substantial performance variation has been observed among the different networks. Eventually, we produce three neural nets with different architectures; a large network requiring 36MB of disk space, a medium-size model requiring 18MB, and a smaller version of the second that requires 6.6MB. These models can be potentially further compressed using the same quantization and compression techniques harnessed in [20].[5]

Model inference process. Once the model is trained, we can use it to compute the conditional probability $Q(x_i)$ (Eq. 2) for each i and each possible configuration of the MRF. This is done by querying the network f using the same mangling trick performed during the training. The procedure used to compute $Q(x_i)$ for x is summarized in the following steps:

1. We substitute the i’th character of x with the *empty character* ’•’, obtaining a mangled password $\tilde{x}$.
2. Then, we feed $\tilde{x}$ to a network that outputs a probability distribution over Σ of the unobserved random variable $\mathbf{x}_i$ i.e., $Q(\mathbf{x}_i)$.
3. Given $Q(\mathbf{x}_i)$, we marginalize out x_i, obtaining the probability:
   $Q(x_i) = P(\mathbf{x}_i{=}x_i \mid \tilde{x})$.

For instance, if we want to compute the local conditional probability of the character *‘e’* in the password x = “*iloveyou*”, we first create $\tilde{x}$ =“ilov•you" and use it as input for the net, obtaining $Q(\mathbf{x}_5)$, then we marginalize that (i.e., $Q(\mathbf{x}_5 = \text{‘}e\text{’})$) getting the probability $P(\mathbf{x}_5 = \text{‘}e\text{’} \mid \tilde{x})$. From the probabilistic point of view, this process is equivalent to fixing the observable variables in the MRF and querying the model for an estimation of the single unobserved character.

At this point, to cast both the feedback mechanism defined in Section 3.1 and the unnormalized joint probability of the string, we have to measure $Q(x_i)$ for each character x_i of the

[4]The Decoder outputs ℓ estimations; one for each input character. Therefore, we apply the softmax function separately on each of those to create ℓ probability distributions.

[5]The code, pre-trained models, and other materials related to our work are publicly available at: `https://github.com/pasquini-dario/InterpretablePPSM`.

$$\begin{tikzcd}
 & & \boxed{x = \text{“}love1\text{”}} \arrow[lld] \arrow[ld] \arrow[d] \arrow[rd] \arrow[rrd] & & \\
f(\text{“}\bullet ove1\text{”}) \arrow[d] & f(\text{“}l\bullet ve1\text{”}) \arrow[d] & f(\text{“}lo\bullet e1\text{”}) \arrow[d] & f(\text{“}lov\bullet 1\text{”}) \arrow[d] & f(\text{“}love\bullet\text{”}) \arrow[d] \\
P(\mathbf{x}_1 \mid \text{“}\bullet ove1\text{”}) & P(\mathbf{x}_2 \mid \text{“}l\bullet ve1\text{”}) & P(\mathbf{x}_3 \mid \text{“}lo\bullet e1\text{”}) & P(\mathbf{x}_4 \mid \text{“}lov\bullet 1\text{”}) & P(\mathbf{x}_5 \mid \text{“}love\bullet\text{”})
\end{tikzcd}$$

Figure 5: Graphical depiction of the complete inference process for the password x=“*love1*”. The function f refers to the trained autoencoder and the symbol '•' refers to the deleted character.

tested password. This is easily achieved by repeating the inference operation described above for each character comprising the input string. A graphical representation of this process is depicted in Figure 5. It is important to highlight that the ℓ required inferences are independent, and their evaluation can be performed in parallel (i.e., batch level parallelism), introducing almost negligible overhead over the single inference. Additionally, with the use of a feed-forward network, we avoid the sequential computation that is intrinsic in recurrent networks (e.g., the issue afflicting [20]), and that can be excessive for a reactive client-side implementation. Furthermore, the convolutional structure enables the construction of very deep neural nets with a limited memory footprint.

In conclusion, leveraging the trained neural network, we can compute the potential of each factor/vertex in the Markov Random Field (defined as local conditional probabilities in our construction). As a consequence, we are now able to cast a PPSM featuring the character-level feedback mechanism discussed in Section 3.1. Finally, in Section 5, we empirically evaluate the soundness of the proposed meter.

5 Evaluation

In this section, we empirically validate the proposed estimation process as well as its deep learning implementation. First, in Section 5.1, we evaluate the capability of the meter of accurately assessing password strength at string-level. Next, in Section 5.2, we demonstrate the intrinsic ability of the local conditional probabilities of being sound descriptors of password strength at character-level.

5.1 Measuring meter accuracy

In this section, we evaluate the accuracy of the proposed meter at estimating password probabilities. To that purpose, following the adversarial reasoning introduced in Section 2.1, we compare the password ordering derived from the meter with the one from the ground-truth password distribution. In doing so, we rely on the guidelines defined in [15] for our evaluation. In particular, given a test-set (i.e., a password leak), we consider a weighted rank correlation coefficient between ground-truth ordering and that derived from the meter. The ground-truth ordering is obtained by sorting the unique entries of the test-set according to the frequency of the password observed in the leak. In the process, we compare our solution with other fully probabilistic meters. A detailed description of the evaluation process follows.

Test-set. For modeling the ground-truth password distribution, we rely on the password leak discovered by 4iQ in the Dark Web [1] on 5th December 2017. It consists of the aggregation of ~ 250 leaks, consisting of 1.4 billion passwords in total. In the cleaning process, we collect passwords with length in the interval $5-16$, obtaining a set of $\sim 4 \cdot 10^8$ unique passwords that we sort in decreasing frequency order. Following the approach of [15], we filter out all the passwords with a frequency lower than 10 from the test-set. Finally, we obtain a test-set composed of 10^7 unique passwords that we refer to as X_{BC}. Given both the large number of entries and the heterogeneity of sources composing it, we consider X_{BC} an accurate description of real-world passwords distribution.

Tested Meters. In the evaluation process, we compare our approach with other probabilistic meters. In particular:

- The Markov model [13] implemented in [3] (the same used in [15]). We investigate different n-grams configurations, namely, 2-grams, 3-grams and 4-grams that we refer to as MM_2, MM_3 and MM_4, respectively. For their training, we employ the same train-set used for our meter.
- The neural approach of Melicher et al. [20]. We use the implementation available at [4] to train the main architecture advocated in [20], i.e., an RNN composed of three LSTM layers of 1000 cells each, and two fully connected layers. The training is carried out on the same train-set used for our meter. We refer to the model as FLA.

Metrics. We follow the guidelines defined by Golla and Dürmuth [15] for evaluating the meters. We use the **weighted** *Spearman* correlation coefficient (ws) to measure the accuracy of the orderings produced by the tested meters, as this has been demonstrated to be the most reliable correlation metric within this context [15]. This metric is defined as

$$\text{ws}(t,m) = \frac{\sum_i^n [w_i(t_i - \bar{t})(m_i - \bar{m})]}{\sqrt{\sum_i^n [w_i(t - \bar{t}_i)^2] \sum_i^n [w_i(m - \bar{m}_i)^2]}} \quad ,$$

where t and m are the sequence of rank assigned to the test-set from the ground-truth distribution and the tested meter,

Table 2: Rank correlation coefficient computed between X_{BC} and the tested meters.

	MM_2	MM_3	MM_4	FLA	ours (large)	ours (middle)	ours (small)
Weighted Spearman ↑	0.154	0.170	0.193	0.217	0.207	0.203	0.199
Required Disk Space ↓	1.1MB	94MB	8.8GB	60MB	36MB	18MB	6.6MB

respectively, and where the bar notation (e.g., $\bar{t}$) expresses the weighted mean in consideration of the sequence of weights w. The weights are computed as the normalized inverse of the ground-truth ranks (Eq. 5).

$$w = \frac{q}{\sum_i^n q_i} \quad \text{with} \quad q = \frac{1}{t+1} \quad . \tag{5}$$

In this metric, the weighting increases the relevance of weak passwords (i.e., the ones with small ranks) in the score computation; that is, the erroneous placing of weak passwords (i.e., asserting a weak password as strong) is highly penalized. Unlike [15], given the large cardinality and diversity of this leak, we directly use the ranking derived from the password frequencies in X_{BC} as ground-truth. Here, passwords with the same frequency value have received the same rank in the computation of the correlation metric.

Results. Table 2 reports the measured correlation coefficient for each tested meter. In the table, we also report the required storage as auxiliary metric.
Our meters, even the smallest, achieve higher or comparable score than the most performant Markov Model, i.e., MM_4. On the other hand, our largest model cannot directly exceed the accuracy of the state-of-the-art estimator FLA, obtaining only comparable results. However, FLA requires more disk space than ours. Indeed, interestingly, our convolutional implementation permits the creation of remarkably lightweight meters. As a matter of fact, our smallest network shows a comparable result with MM_4 requiring more than a magnitude less disk space.
In conclusion, the results confirm that the probability estimation process defined in Section 3.2 is indeed sound and capable of accurately assessing password mass at string-level. The proposed meter shows comparable effectiveness with the state-of-the-art [20], whereas, in the *large* setup, it outperforms standard approaches such as Markov Chains. Nevertheless, we believe that even more accurate estimation can be achieved by investigating deeper architectures and/or by performing hyper-parameters tuning over the model.

5.2 Analysis of the relation between local conditional probabilities and password strength

In this section, we test the capability of the proposed meter to correctly model the relation between password structure and password strength. In particular, we investigate the ability of the measured local conditional probabilities of determining the tested passwords' insecure components.
Our evaluation procedure follows three main steps. Starting from a set of weak passwords X:

1. We perform a guessing attack on X in order to estimate the guess-number of each entry of the set.
2. For each password $x \in X$, we substitute n characters of x according to the estimated local conditional probabilities (i.e., we substitute the characters with highest $Q(\mathbf{x}_i)$), producing a perturbed password $\tilde{x}$.
3. We repeat the guessing attack on the set of perturbed passwords and measure the variation in the attributed guess-numbers.

Hereafter, we provide a detailed description of the evaluation procedure.

Passwords sets. The evaluation is carried out considering a set of weak passwords. In particular, we consider the first 10^4 most frequent passwords of the X_{BC} set.

Password perturbations. In the evaluation, we consider three types of password perturbation:
(1) The first acts as a baseline and consists of the substitution of random positioned characters in the passwords with randomly selected symbols. Such a general strategy is used in [24] and [14] to improve the user's password at composition time. The perturbation is applied by randomly selecting n characters from x and substituting them with symbols sampled from a predefined character pool. In our simulations, the pool consists of the 25 most frequent symbols in X_{BC} (i.e., mainly lowercase letters and digits). Forcing this character-pool aims at preventing the tested perturbation procedures to create artificially complex passwords such as strings containing extremely uncommon *unicode* symbols. We refer to this perturbation procedure as **Baseline**.
(2) The second perturbation partially leverages the local conditional probabilities induced by our meter. Given a password x, we compute the conditional probability $Q(x_i)$ for each character in the string. Then, we select and substitute the character with maximum probability, i.e., $\arg\max_{x_i} Q(x_i)$. The symbol we use in the substitution is randomly selected from the same pool used for the baseline perturbation (i.e., top-25 frequent symbols). When n is greater than one, the procedure is repeated sequentially using the perturbed password obtained from the previous iteration as input for the next step. We refer to this procedure as **Semi-Meter**.
(3) The third perturbation extends the second one by exploiting the local conditional distributions. Here, as in the Semi-Meter-based, we substitute the character in x with the highest probability. However, rather than choosing a substitute symbol in the pool at random, we select that according to the

Table 3: Strength improvement induced by different perturbations. The last two rows of the table report the AGI ratio between the two meter-based approaches and the baseline.

	$n=1$	$n=2$	$n=3$
Baseline (PNP)	0.022	0.351	0.549
Semi-Meter (PNP)	0.036	0.501	0.674
Fully-Meter (PNP)	0.066	0.755	0.884
Baseline (**AGI**)	$3.0\cdot10^{10}$	$3.6\cdot10^{11}$	$5.6\cdot10^{11}$
Semi-Meter (**AGI**)	$4.6\cdot10^{10}$	$5.1\cdot10^{11}$	$6.8\cdot10^{11}$
Fully-Meter (**AGI**)	$8.2\cdot10^{10}$	$7.7\cdot10^{11}$	$8.9\cdot10^{11}$
Semi-Meter / Baseline (**AGI**)	**1.530**	**1.413**	**1.222**
Fully-Meter / Baseline (**AGI**)	**2.768**	**2.110**	**1.588**

distribution $Q(\mathbf{x}_i)$, where i is the position of the character to be substituted. In particular, we choose the symbol the minimize $Q(\mathbf{x}_i)$, i.e., $\arg\min_{s\in\Sigma'} Q(\mathbf{x}_i=s)$, where Σ' is the allowed pool of symbols. We refer to this method as **Fully-Meter**.

Guessing Attack. We evaluate password strength using the *min-auto* strategy advocated in [27]. Here, guessing attacks are simultaneously performed with different guessing tools, and the guess-number of a password is considered the minimum among the attributed guess-numbers. In performing such attacks, we rely on the combination of three widely adopted solutions, namely, HashCat [2], PCFG [6,29] and the Markov chain approach proposed in [5,13]. For tools requiring a training phase, i.e., OMEN and PCFG, we use the same train-set used for our model (i.e., 80% of *RockYou*). Similarly, for HashCat, we use the same data set as input dictionary[6] and *generated2* as rules set. During the guesses generation, we maintain the default settings of each implementation. We limit each tool to produce 10^{10} guesses. The total size of the generated guesses is $\sim$ 3TB.

Metrics. In the evaluation, we are interested in measuring the increment of password strength caused by an applied perturbation. We estimate that value by considering the Average Guess-number Increment (henceforth, referred to as AGI); that is, the average delta between the guess-number of the original password and the guess-number of the perturbed password:

$$\mathrm{AGI}(X) = \frac{1}{|X|}\sum_{i=0}^{|X|}[g(\tilde{x}^i) - g(x^i)]$$

where g is the guess-number, and $\tilde{x}^i$ refers to the perturbed version of the i'th password in the test set. During the computation of the guess-numbers, it is possible that we fail to guess a password. In such a case, we attribute an artificial guess-number equals to 10^{12} to the un-guessed passwords. Additionally, we consider the average number of un-guessed passwords as an ancillary metrics; we refer to it with the name of Percentage Non-Guessed Passwords (PNP) and compute it as:

$$\mathrm{PNP(X)} = \frac{1}{|X|}|\{x^i \mid g(x^i) \neq \perp \,\wedge\, g(\tilde{x}^i) = \perp\}|,$$

where $g(x) = \perp$ when x is not guessed during the guessing attack.

Results. We perform the tests over three values of n (i.e., the number of perturbed characters), namely, 1, 2, and 3. Results are summarized in Table 3. The AGI caused by the two meter-based solutions is always greater than that produced by random perturbations. On average, that is twice more effective with respect to the Fully-Meter baseline and about 35% greater for the Semi-Meter. The largest relative benefit is observable when $n=1$, i.e., a single character is modified. Focusing on the Fully-Meter approach, indeed, the guidance of the local conditional probabilities permits a guess-number increment 2.7 times bigger than the one caused by a random substitution in the string. This advantage drops to ~ 1.5 when $n=3$, since, after two perturbations, passwords tend to be already out of the dense zone of the distribution. Indeed, at $n=3$ about 88% of the passwords perturbed with the Fully-Meter approach cannot be guessed during the guessing attack (i.e., PNP). This value is only $\sim$ 55% for the baseline. More interestingly, the results tell us that substituting two ($n=2$) characters following the guide of the local conditional probabilities causes a guess-number increment greater than the one obtained from three ($n=3$) random perturbations. As a matter of fact, the AGI for the Fully-Meter perturbation is $\sim 7.6\cdot10^{11}$ for $n=2$ whereas is $\sim 5.7\cdot10^{11}$ for the baseline when $n=3$.
In the end, these results confirm that the local conditional distributions are indeed sound descriptors of password security at the structural level.

Limitations. Since the goal of our evaluation was mainly to validate the soundness of the proposed estimation process, we did not perform user studies and we did not evaluate human-related factors such as password memorability although we recognize their importance.

6 Conclusion

In this paper, we showed that it is possible to construct interpretable probabilistic password meters by rethinking the underlying password mass estimation. We presented an undirected probabilistic interpretation of the password generative process that can be used to build precise and sound password feedback mechanisms. Moreover, we demonstrated that such an estimation process could be instantiated via a lightweight deep learning implementation. We validated our undirected description and deep learning solution by showing that our meter achieves comparable accuracy with other existing approaches while introducing a unique character-level feedback mechanism that generalizes any heuristic construction.

[6]In this case, passwords are unique and sorted in decreasing frequency.

References

[1] "1.4 Billion Clear Text Credentials Discovered in a Single Database". https://tinyurl.com/t8jp5h7.

[2] "hashcat GitHub". https://github.com/hashcat.

[3] "NEMO Markov Model GitHub". https://github.com/RUB-SysSec/NEMO.

[4] "Neural Cracking GitHub". https://github.com/cupslab/neural_network_cracking.

[5] "OMEN GitHub". https://github.com/RUB-SysSec/OMEN.

[6] "PCFG GitHub". https://github.com/lakiw/pcfg_cracker.

[7] "RockYou Leak". https://downloads.skullsecurity.org/passwords/rockyou.txt.bz2.

[8] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[9] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. Adaptive Password-Strength Meters from Markov Models. In *NDSS*, 2012.

[10] Luke St. Clair, Lisa Johansen, William Enck, Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Trent Jaeger. Password exhaustion: Predicting the end of password usefulness. In *Information Systems Security*, pages 37–55, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[11] M. Dell' Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, March 2010.

[12] Matteo Dell'Amico and Maurizio Filippone. Monte Carlo Strength Evaluation: Fast and Reliable Password Checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 158–169, New York, NY, USA, 2015. Association for Computing Machinery.

[13] Markus Duermuth, Fabian Angelstorf, Claude Castelluccia, Daniele Perito, and Abdelberi Chaabane. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. In *International Symposium on Engineering Secure Software and Systems*, milan, Italy, March 2015.

[14] Alain Forget, Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. Improving Text Passwords through Persuasion. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, page 1–12, New York, NY, USA, 2008. Association for Computing Machinery.

[15] Maximilian Golla and Markus Dürmuth. On the Accuracy of Password Strength Meters. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1567–1582, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[17] Saranga Komanduri, Richard Shay, Lorrie Faith Cranor, Cormac Herley, and Stuart Schechter. Telepathwords: Preventing Weak Passwords by Reading Users' Minds. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 591–606, San Diego, CA, August 2014. USENIX Association.

[18] Jianzhu Ma, Jian Peng, Sheng Wang, and Jinbo Xu. Estimating the Partition Function of Graphical Models Using Langevin Importance Sampling . In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 433–441, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

[19] J. L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, June 1994.

[20] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, Austin, TX, August 2016. USENIX Association.

[21] Arvind Narayanan and Vitaly Shmatikov. Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[22] Dario Pasquini, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, and Mauro Conti. Improving Password Guessing via Representation Learning. In *2021 42th IEEE Symposium on Security and Privacy*, May 2021.

[23] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context Encoders: Feature Learning by Inpainting. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. Design and Evaluation of a Data-Driven Password Meter. In *CHI '17*, 2017.

[25] Blase Ur, Jonathan Bees, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Do users' perceptions of password security match reality? In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 3748–3760, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 65–80, Bellevue, WA, 2012. USENIX.

[27] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 463–481, Washington, D.C., August 2015. USENIX Association.

[28] D. Wang, D. He, H. Cheng, and P. Wang. fuzzyPSM: A New Password Strength Meter Using Fuzzy Probabilistic Context-Free Grammars. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 595–606, June 2016.

[29] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405, May 2009.

[30] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 162–175, New York, NY, USA, 2010. Association for Computing Machinery.

[31] Daniel Lowe Wheeler. zxcvbn: Low-Budget Password Strength Estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX, August 2016. USENIX Association.

[32] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012.

Appendix

A Estimating guess-numbers

Within the context of PPSMs, a common solution to approximate guess-numbers [19] is using the Monte Carlo method proposed in [12]. With a few adjustments, the same approach can be applied to our meter. In particular, we have to derive an approximation of the partition function Z. This can be done by leveraging the Monte Carlo method as follows:

$$Z \simeq N \cdot \mathbb{E}_x[P(x)] \tag{6}$$

where N is the number of possible configurations of the MRF (i.e., the cardinality of the key-space), and x is a sample from the posterior distribution of the model. Samples from the model can be obtained in three ways: (1) sampling from the latent space of the autoencoder (as done in [22]), (2) performing Gibbs sampling from the autoencoder, or (3) using a dataset of passwords that follow the same distribution of the model. Once we have an approximation of Z, we can use it to normalize every joint probability, i.e., $P(x) = \frac{\tilde{P}(x)}{Z}$ and then apply the method in [12]. Alternatively, we could adopt a more articulate solution as in [18]. In any event, the estimation of the partition function Z is performed only once and can be done offline.

B Model Architectures and hyper-parameters

In this section, we detail the technical aspects of our deep learning implementation.

Architectures As previously described, we base our networks on a *resnet* structure. We use a bottleneck residual block composed of three mono-dimensional convolutional layers as the atomic building block of the networks. A graphical description of that is depicted in Figure 6. We construct three different networks with different sizes (intended as the number of trainable parameters). We determine the size of the networks by varying the number of residual blocks, the kernel size of the convolutional layers in the blocks, and the number of filters. The three architectures are reported in Tables 5, 6 and 7.

Training process Table 4 reports the used hyper-parameters. During the training, we apply label smoothing, which is controlled from the parameter ε. We found our models taking

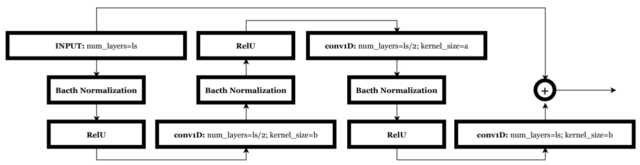

INPUT: num_layers=ls
Bacth Normalization
RelU
conv1D: num_layers=ls/2; kernel_size=b
Bacth Normalization
RelU
conv1D: num_layers=ls/2; kernel_size=a
Bacth Normalization
RelU
conv1D: num_layers=ls; kernel_size=b
+

Figure 6: Depiction of: `ResblockBneck1D`[fn$=ls$, ks$=(a,b)$]

Table 4: Hyper-parameters used to train our AE

Hyper-parameter	**Value**
α	10
Batch size	3024
Learning rate	0.0001
Optimizer	*Adam*

Train Epochs	small$=$10
	medium$=$5
	large$=$5

ε	0.05

particular advantage from large batch-sizes. We limit that to 3072 for technical limitations; however, we believe that bigger batches could further increment the quality of the password estimation.

Table 5: Small architecture.

`cov1d`[3, 128, *same*, *linear*]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`Flatten`
`FullyConnected`[128, *linear*]
`FullyConnected`[*MaxPasswordLength* · 128, *linear*]
`Reshape`[*MaxPasswordLength*, 128]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`ResblockBneck1D`[fn$=$128, ks$=(3,1)$]
`Flatten`
`FullyConnected`[*MaxPasswordLength* · *AlphabetCardinality*, *linear*]

C Supplementary resources

This Section reports additional resources. Table 8 reports examples of password perturbation performed using the method **Fully-meter** on the three values of n. The example passwords (first column) are sampled from X_{BC}. Figure 7 reports additional examples of the feedback mechanism. The depicted passwords have been randomly sampled from the tail of the *RockYou* leak.

Table 6: Medium architecture.

`cov1d`[5, 128, *same*, *linear*]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`Flatten`
`FullyConnected`[80, *linear*]
`FullyConnected`[*MaxPasswordLength* · 128, *linear*]
`Reshape`[*MaxPasswordLength*, 128]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$128, ks$=(5,3)$]
`Flatten`
`FullyConnected`[*MaxPasswordLength* · *AlphabetCardinality*, *linear*]

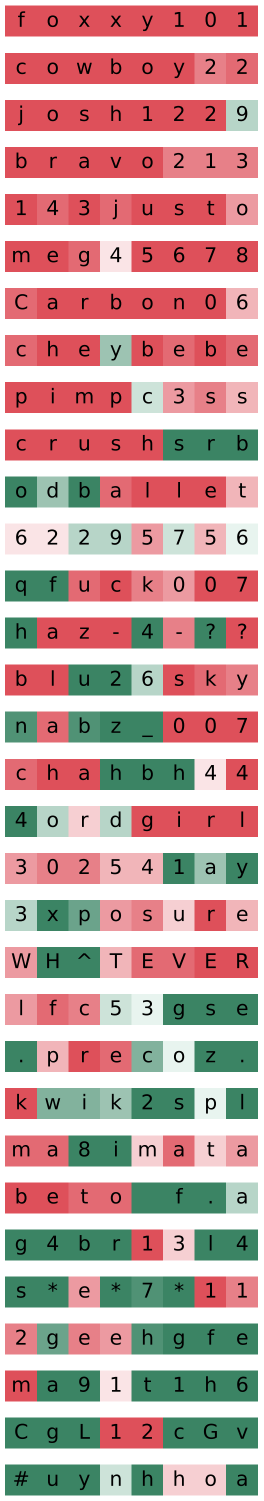
Table 7: Large architecture.

`cov1d`[5, 128, *same*, *linear*]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`Flatten`
`FullyConnected`[80, *linear*]
`FullyConnected`[*MaxPasswordLength* · 128, *linear*]
`Reshape`[*MaxPasswordLength*, 128]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$]
`ResblockBneck1D`[fn$=$200, ks$=(5,3)$] `Flatten`
`FullyConnected`[*MaxPasswordLength* · *AlphabetCardinality*, *linear*]

Table 8: Examples of password perturbation **automatically** produced by the method **Fully-meter**. Symbols in bold are the ones substituted by the meter.

x	$n=1$	$n=2$	$n=3$
heaven7	**2**eaven7	**2**eav**1**n7	**2**e**9**v**1**n7
corvette	corv**l**tte	c**5**rv**l**tte	c**5**rv**l**tt**b**
mariah	m**3**riah	m**3u**iah	**u3u**iah
373737	37373**l**	**t**7373**l**	**t**737**ul**
veronica1	v**s**ronica1	v**s**ron**5**ca1	v**s**r**s**n**5**ca1
ariana1	aria**o**a1	**3**ria**o**a1	**3**r**6**a**o**a1
goodgirl	goodgir**3**	g**9**odgir**3**	**u9**odgir**3**
cheer	c**9**eer	c**9**e**h**r	c**9yh**r
mahalko	m**l**halko	m**l**h**8**lko	m**l**h**8**lk**t**
19981998	19981**n**98	**u**9981**n**98	**u**9**o**81**n**98
123456aa	**i**23456aa	**i**234**r**6aa	**i**23**nr**6aa
helena	hele**a**a	h**4**le**a**a	h**4i**e**a**a
montana1	monta**2**a1	mo**3**ta**2**a1	mo**3a**a**2**a1
vancouver	vancouv**m**r	va**9**couv**m**r	va**9**co**6**v**m**r
fuck12	f**h**ck12	f**h**ck1**n**	f**h**c**8**1**n**
patriots1	pat**9**iots1	pat**9**ioti**1**...	p**5**t**9**ioti1
evelyn1	eve**4**yn1	**6**ve**4**yn1	**6**ve**4**yn**r**
pancho	panc**2**o	pa**6**c**2**o	**9**a**6**c**2**o
malibu	m**5**libu	m**5y**ibu	m**5y**ib**6**
ilovemysel	ilo**0**emysel	i**i**o**0**emysel	i**i**o**0**emys**8**l
galatasaray	galatasar**4**y	galat**6**sar**4**y	g**8**lat**6**sar**4**y
tootsie1	to**5**tsie1	to**5**ts**9**e1	to**5**t**n9**e1
sayangku	saya**8**gku	s**3**ya**8**gku	s**3**ya**8**gk**8**
moneyman	mo**5**eyman	mo**5**eym**d**n	**u**o**5**eym**d**n
theboss	th**9**boss	th**9**bos**4**	th**9**b**t**s**4**
112211	**o**12211	**o**1221**u**	**oe**221**u**
k12345	k123**y**5	k12**oy**5	k**n**2**oy**5
alexis	**9**lexis	**9**l**r**xis	**9**l**r**x**o**s
princess7	princ**4**ss7	pri**h**c**4**ss7	pr**rh**c**4**ss7
rooster1	roo**3**ter1	roo**3**t**l**r1	r**6**o**3**t**l**r1
june15	jun**m**15	jun**mr**5	j**l**n**mr**5
samurai1	**0**amurai1	**0**amu**0**ai1	**0e**mu**0**ai1
surfer1	s**9**rfer1	s**9**rf**n**r1	s**9**rf**n**r**3**
lokomotiv	l**h**komotiv	l**h**komot**6**v	l**h**ko**8**ot**6**v
rfn.irf	rfn.i**5**f	**5**fn.i**5**f	**5e**n.i**5**f
melisa	m**t**lisa	m**t**lis**l**	m**t**l**6**s**l**
minime	**3**inime	**3**inim**t**	**3**i**i**im**t**
peaceout	pea**a**eout	**8**ea**a**eout	**8**ea**a**eo**1**t
louise	lo**4**ise	l**r4**ise	l**r4**is**r**
Liverpool	Live**h**pool	Live**h**p**2**ol	Li**6**e**h**p**2**ol
147896	1**d**7896	1**d**78**y**6	1**d**78**yy**
aditya	ad**l**tya	**4**d**l**tya	**4**d**l**ty**i**
qwerty13	qw**m**rty13	qw**m**r**9**y13	qw**m**r**9**y**u**3
070809	**i**70809	**i**708**d**9	**i**7**r**8**d**9
emmanuel1	emm**9**nuel1	emm**9**nue**i**1	e**0**m**9**nue**i**1
beautiful2	be**1**utiful2	be**1**utif**1**l2	be**1**ut**n**f**1**l2

Figure 7: Additional examples of the feedback mechanism.


f o x x y 1 0 1
c o w b o y 2 2
j o s h 1 2 2 9
b r a v o 2 1 3
1 4 3 j u s t o
m e g 4 5 6 7 8
C a r b o n 0 6
c h e y b e b e
p i m p c 3 s s
c r u s h s r b
o d b a l l e t
6 2 2 9 5 7 5 6
q f u c k 0 0 7
h a z - 4 - ? ?
b l u 2 6 s k y
n a b z _ 0 0 7
c h a h b h 4 4
4 o r d g i r l
3 0 2 5 4 1 a y
3 x p o s u r e
W H ^ T E V E R
l f c 5 3 g s e
. p r e c o z .
k w i k 2 s p l
m a 8 i m a t a
b e t o f . a
g 4 b r 1 3 l 4
s * e * 7 * 1 1
2 g e e h g f e
m a 9 1 t 1 h 6
C g L 1 2 c G v
# u y n h h o a