Can Foundation LLMs Accurately Estimate Password Strength and Provide Appropriate Password Feedback?

Madison Pickering, Garrison Hinson-Hasty, Luca Dovichi,
Helena Williams, Nathaniel Kim, Aybala Esmer, Blase Ur
University of Chicago

***Abstract*—Password-strength models help users choose strong passwords. These models typically identify predictable passwords by training on millions of stolen passwords. We instead investigate if pretrained large language models (LLMs) can assess a password's strength based on semantic properties. We adapt LLMs in two distinct ways. First, mirroring LLMs' typical usage, we directly prompt four LLMs to estimate password strength. Second, mirroring prior probabilistic password models, we explicitly calculate the probability of passwords using LLMs fine-tuned on password data. We uncover technical challenges with doing so, developing strategies to reduce computational costs by a median of 25–97% depending on a password's length. We also perform an ablation study to gauge how key design dimensions (e.g., pretraining, data quantity) impact fine-tuned LLMs' accuracy. We find many fine-tuned models outperform prompted models despite being far smaller. Across both prompting and fine-tuning, LLMs modeled word-like passwords well—including for out-of-distribution passwords (e.g., with recent cultural references)—yet struggled on more complex passwords. We also prompted LLMs to suggest how to improve specific passwords. The LLMs often gave poor or predictable feedback (e.g., making common substitutions, appending an exclamation point, or inserting "blue").**

1. Introduction

Text passwords remain a common form of authentication. To keep an account secure, a password must be hard for an attacker to guess under realistic threat models [1], [2]. To support this goal, systems often give users estimates of their password's strength [3]. Many existing approaches to modeling strength require millions of high-quality passwords, which were relatively plentiful in the early 2010s [4], [5], [6], [7]. Since then, major websites have migrated away from storing passwords insecurely, resulting in substantially less high-quality data being released on the internet [8]. As a result, existing models' understanding of password strength may be "frozen in time" with no clear way to update them.

We consequently investigate the extent to which pretrained large language models (**LLMs**) can rate the strength of passwords based on their *semantic properties*, as opposed to their frequency in prior password leaks. For example, existing models may incorrectly rate the password `oliviarodrigo1` as strong as it rarely appears in passwords from the 2010s. However, an LLM may correctly find it to be weak as it references a pop star. LLMs may also be able to leverage their semantic understanding to give rich explanations to users about how to improve their password. Our study investigates if pretrained LLMs can give users: (i) accurate estimates of the strength of their password and (ii) appropriate feedback on how to improve their password. In exploring these questions, we establish how LLMs can fill gaps in existing password strength tools, not replace them.

Configuring prior probabilistic password models [4], [6], [9] is relatively straightforward—they simply need to be trained on sets of passwords. As LLMs are multi-purpose and can be used in myriad ways, how to use them to model passwords is less obvious. In their typical usage, LLMs are prompted with a task, generating output text. Thus, mirroring common LLM usage (yet differing from traditional approaches to password modeling), we first investigate directly prompting pretrained foundation LLMs to rate the strength of passwords without any additional fine-tuning. We test four recent LLMs (e.g., Gemini-2.5 Pro, Llama 4) using a variety of different prompting strategies.

We find that LLMs excel at modeling the strength of passwords that resemble natural language (common under basic composition policies), but struggle when passwords do not. We also find that LLMs identify weak passwords that prior approaches miss (e.g., referencing Adele song lyrics, Wendy's Baconator sandwich, or non-Western names).

We also evaluate a second approach to LLM-based password modeling that more closely mirrors prior password models: treating the LLM itself as a probabilistic model. Specifically, we fine-tune models from the Llama 2 and GPT-2 model families, which contain older and smaller LLMs for which fine-tuning is more feasible computationally. In preparing to do so, we uncover and solve a key technical challenge with treating an LLM as a probabilistic model: LLMs tokenize inputs very differently than existing password models, meaning that a given password can be represented in a potentially very large number of ways by the model. We introduce and evaluate pruning strategies that overcome this challenge. We then conduct extensive experiments on how key variables (e.g., quantity of training data, model size, data formatting) impact the accuracy of these fine-tuned models.

Surprisingly, we find that our fine-tuned models outperform our prompted models despite relying on much

older and smaller LLMs. Similarly, we find evidence that LLMs can perform well after fine-tuning on as few as 10,000 passwords, deriving much of their abilities from pretraining data. They are also fairly robust to how examples are formatted. As with prompted LLMs, fine-tuned LLMs excel at modeling basic, word-like passwords, but struggle in modeling complex passwords, those containing leet substitutions, and randomly generated passwords. We again observe that they capture cultural knowledge in recognizing non-Western names and pop-culture references traditional password-modeling methods miss. Unlike when they are prompted, fine-tuned LLMs also excel at modeling passphrases that prior state-of-the-art approaches miss.

Given the high degree to which users rely on modern LLMs for advice and guidance, we also explore the feedback LLMs give about passwords. Specifically, we prompt LLMs for **abstract recommendations** (e.g., "add a symbol") and **concrete modifications** (e.g., `monkey` $\rightarrow$ `m0nkey!`) for improving specific passwords. Taken individually, many recommendations superficially seem sensible (e.g., making passwords longer), though they also reflect common tropes (e.g., add `!` to the end, substitute `e` $\rightarrow$ `3`). Analyzing concrete modifications at scale, though, it becomes clear that LLMs' design in generating likely completions of text largely works *against* them when providing feedback. LLMs often repeat the same recommendations (e.g., add "blue," "giraffe," or "correcthorse") that would not meaningfully increase security because of their predictability.

To our knowledge, we are the first to deeply explore and compare prompting-based and fine-tuning-based approaches to adapting LLMs to model passwords. We are also the first to explore LLMs' suggestions for improving passwords. While it is likely misguided to fully replace existing password models with computationally heavyweight LLMs, we demonstrate that LLMs can have a place in the password-modeling ecosystem. Specifically, we show that LLMs' internet-scale pretraining can capture nuances of language and complex semantics without much (or any) password-specific training data, appropriate versions of which may or may not be available to a system administrator. Thus, better understanding how to adapt LLMs to model current passwords can enable more adaptable and robust models.

2. Background and Related Work

We first discuss our threat model. We then summarize existing work on password modeling, including via LLMs.

2.1. Threat Model and Setting

In a password-guessing attack, the attacker tries to compromise accounts by guessing passwords [10], [11], [12]. We consider only **untargeted**, or trawling, attacks. Given a fixed number of guesses, the attacker's goal is to compromise as many accounts as possible [13], [14], as opposed to any specific account. The attacker is assumed to have knowledge about what passwords users in aggregate might choose, but no knowledge about any specific user's personal information or passwords on other sites. In modern untargeted attacks, attackers typically leverage general knowledge of password distributions learned from password stores stolen from other websites [15]. To date, hundreds of websites have suffered such breaches [8]. We assume the adversary follows an optimal strategy by guessing in descending probability order (i.e., the password they believe is most probable comes first).

Attacks can be online or offline, the latter of which is our focus. In an online attack, an attacker attempts to log into a live system. The system will likely employ rate limiting, bounding the total number of guesses [2]. Florêncio et al. estimated that a password that is not roughly among the million most likely is likely to withstand an online attack [1]. If an attacker instead carries out an offline attack and compromises a service's password database, which ought to be salted and hashed [8], formal rate limiting is not possible. Instead, the attacker's limiting factor is the amount of compute they dedicate to the attack. A service can further slow an attack by choosing an intentionally slow and memory-hard hash function [16]. Florêncio et al. estimated that a password not among the 10^{14} most probable may withstand an offline attack [1] depending on the hash function used. We discuss these assumptions further in Section 3.3.

To estimate password strength, one can simulate a guessing attack [10], [17]. The results of such simulations are often presented as a **guessing curve** [18], [19], [20] showing the proportion of a password distribution guessed as the adversary's guessing budget increases. Each password is assigned a **guess number** indicating how many guesses in the simulated attack it would take to guess that password. For instance, the password with guess number 100 would be the 100th most probable under that model.

Targeted attacks are also possible. Some rely on users' tendency to reuse passwords across websites; when one service is breached, the attackers try similar pairs of usernames and passwords for others [21], [22], [23]. Other targeted attacks leverage the user's personal information [24], [25]. We consider both forms of targeted attacks out of scope because the user's passwords on other services are typically not available when rating a user's password.

2.2. Prior Approaches to Guessing Passwords

Many password-guessing models have been proposed. They usually generate guesses by modeling the likelihood of passwords based on large datasets of leaked passwords.

Mangled Wordlists: Widely used software tools like Hashcat [26] let attackers define mangling rules (transformations that simulate common password substitutions). These transformations are applied to frequency-ordered lists of passwords [27]. While this approach produces an ordered list of guesses, it does not assign probabilities to passwords.

Probabilistic Approaches: Weir et al. [6] proposed a Probabilistic Context-Free Grammar (**PCFG**) for passwords. This approach assigns probabilities to passwords' character-class structures and each structure's contents. A password's probability is the product of these probabilities. Like PCFGs,

Markov models learn a probability distribution from input password sets [5]. In contrast to PCFGs, Markov Models define an *order* specifying how many previous characters to consider when transitioning states. For instance, an order-5 Markov model considers the previous five characters. Ma et al. found order-6 models with additive smoothing to be optimal [9]. Other probabilistic methods use deep learning. Melicher et al. proposed using recurrent neural networks (**RNNs**) to model passwords and analyzed how model characteristics affect password guessability [4]. Other researchers have also used deep learning for modeling passwords [3], [28], [29], including for targeted attacks [23]. Hitaj et al. [28] proposed using a Generative Adversarial Network (**GAN**) to model passwords. A GAN uses two neural networks: one that generates samples and one that attempts to distinguish those samples from a reference distribution. The literature is split on this approach's effectiveness [30].

LLMs: There has been limited prior work on using LLMs to guess passwords, and even less focusing on using LLMs to model password strength. Much of this work uses LLM-like architectures that have *not* been pretrained on internet-scale data [31], [32], [33] and/or focuses specifically on targeted guessing attacks [34], [35], [36], [37]. In contrast, we focus specifically on *pretrained foundation models* and how they could be used to *model password strength*, not guess passwords. Furthermore, we are the first to compare prompting LLMs with fine-tuning LLMs. To our knowledge, Atzori et al. [36] are the only others to consider prompting LLMs to estimate password strength, and they do so only in passing (and without extensive testing).

Arguably closest to our fine-tuning experiments, Zou et al.'s recent PassLLM proposes fine-tuning foundation LLMs to guess passwords [38]. While we extensively evaluate the impacts of tokenization strategies, their approach uses only character-by-character tokenization, which we find in Section 5.1.2 to be highly unreliable in capturing a password's true probability mass according to an LLM. Furthermore, they focus on adversarially guessing passwords, not estimating password strength. As such, they detail strategies for efficiently generating unique guesses, but they do not compare how the guess numbers their approach would assign to passwords compare to existing benchmarks like PGS, nor do they consider LLM prompting at all. We also uniquely perform extensive ablation studies of how different fine-tuning approaches impact the accuracy of strength estimates.

2.3. Relevant Background About LLMs

The process of training LLMs differs from prior probabilistic password models. LLMs use an attention mechanism to model dependencies in natural language, which is more computationally efficient than the convolution or recurrence used by prior deep-learning models [39]. Consequently, a model developer (e.g., Meta) performs the bulk of model **pretraining** on internet-scale data [40]. Downstream users often simply prompt the pretrained model with their task. However, downstream developers (e.g., researchers) can instead **fine-tune** the pretrained model to their task.

3. Datasets and Configurations

Here, we describe the LLMs, evaluation sets, strength ratings for existing models, and metrics we employed.

3.1. LLMs

Prompted LLMs: We focused our prompting experiments on recent, state-of-the-art LLMs. Specifically, we experimented on Anthropic's Claude Opus 4 (20250514-v1:0), Google's Gemini-2.5 Pro (v1.0), Meta's Llama 4 Maverick 17B Instruct (v1:0), and OpenAI's Gpt-4.1 (2025-04-14). These models were not tool-connected and could not search the internet during prompting. While we also tried experimenting on GPT-2 and Llama 2 (the LLMs used for our fine-tuning experiment), they were not trained to follow instructions and consequently rarely returned strength estimates. We set the temperature to 0 for all prompting experiments to minimize response nondeterminism. While we specified the desired output format in our prompts, LLMs sometimes did not adhere to it. As detailed in Appendix B.3, we used a combination of automated follow-up parsing prompts and regular expressions to parse model responses.

Fine-Tuned LLMs: For fine-tuning experiments, we required unfettered access to a model's "logit" layer. This access is typically restricted for proprietary models [41], [42], [43], [44]. We therefore used the open-weight GPT-2 XL and Llama 2 7B models [45], [46]. We chose the older GPT-2 model family because it is comparatively lightweight and well-studied [47], [48]. We chose Llama 2 because it is likely more representative of modern LLMs, yet still lightweight [46]. As listed in Appendix B, we used the same training parameters between models, but used different GPUs for the GPT-2 and Llama 2 families due to their computational needs. Unlike prompted LLMs, we were able to control the computing environment and hardware on which the models ran, making the models' behavior deterministic.

Because many password breaches contain passwords specific to the service on which they were created [49], [50], we decided to use a compilation of passwords from many services for fine-tuning. We chose the XATO collection [51] of 10 million passwords. This set has been used in a number of prior studies [17], [20], [27], [52], [53]. Due to the high cost of fine-tuning LLMs, we downsampled XATO to two sets of 10,000 passwords and two of 100,000 passwords, enabling us to test the impact of the amount of fine-tuning data. To test advice to de-duplicate fine-tuning data [54], [55], we created two sets of each size: one that de-duplicated passwords prior to down-sampling and one that did not.

3.2. Evaluation Password Sets

A key challenge in evaluating LLM-based approaches on existing data is that they have been pretrained on (often unspecified) internet-scale data, making it difficult to ascertain if a particular document was included in the training data [42]. As the most frequently studied password leaks have been widely distributed, it is likely that they are

included in LLMs' training data. Thus, many candidate evaluation sets are likely to be included in LLMs' pretraining data, confounding results. We also note that some passwords may be dictionary words or common across leaks, potentially resulting in train/test overlap. This type of overlap is common in the password modeling space [3], [4], [5], [38] due to users' tendency to reuse passwords or use dictionary words as a basis for passwords [21], [56], [57], contributing to natural overlap between the passwords a model is trained on and the "unseen" passwords encountered when deployed.

For our evaluation sets, we thus selected two types of data less likely to feature prominently in LLMs' pretraining data. First, we randomly sampled 500 passwords (≥ 8 characters in length) from a data breach that had recently become public and was not yet widely distributed. Specifically, the book-sharing website **Bookcrossing** suffered a data breach in August 2022 consisting of a pre-2013 database backup of plaintext passwords [58], [59]. To facilitate comparisons with prior work, we also wanted to evaluate a well-studied leak. Thus, we also sampled 500 passwords from the prominent 2012 **Linkedin** breach [60] using the same method.

The Bookcrossing breach began circulating in July 2023. This is after our fine-tuned LLMs' knowledge cutoffs: GPT-2's in November 2019 [61] and Llama 2's in September 2022 [62]. However, Linkedin began circulating in 2012, and it is likely our fine-tuned LLMs were pretrained on it. Our prompted LLMs have knowledge cutoffs between June 2024 and March 2025 [63], [64], [65], [66]. Our prompted LLMs could have been trained on the Bookcrossing and Linkedin breaches. It is particularly likely that they were trained on Linkedin due to its wider circulation.

To help ensure the LLMs were tested on data they could not have seen during training, we also evaluated our approaches on four sets of passwords collected in highly cited academic studies from researchers at Carnegie Mellon University (CMU) [18]. These evaluation sets enabled us to test both LLM approaches on passwords of varying composition characteristics. Each dataset represents passwords created under a particular password-composition policy: (i) **CMU-Basic** (length 8+ characters); (ii) **CMU-LongBasic** (length 16+ characters); (iii) **CMU-Complex** (length 8+ characters, includes lowercase letters, uppercase letters, digits, and symbols); and (iv) **CMU-LongComplex** (length 12+ characters, includes at least three character classes). These passwords were originally collected from crowdworkers in IRB-approved user studies. We wrote to the principal investigators who collected that data, and they agreed to share with us de-identified copies of the passwords without any identifiers, pseudonyms, or metadata. Our institution's IRB formally determined that our secondary analysis of this data was not considered human-subjects research. We randomly sampled 500 passwords from each set.

3.3. Existing Models' Strength Ratings

We required a way to ensure that LLMs' strength ratings were reasonably correct. The closest the research community has to a standardized benchmark is CMU's **Password Guessability Service (PGS)** [67], which simulates a number of widely studied password-guessing approaches [18]. PGS contains Melicher et al.'s neural network [4], Weir et al.'s PCFG [6], Ma et al.'s Markov model [9], John the Ripper [68], and Hashcat [26]. PGS calculates an ensemble strength rating (**MinAuto**) for each password equal to the minimum guess number given by any model. Ur et al. found MinAuto to be a conservative estimate of a password's strength under attacks by skilled adversaries [18]. Since PGS does not contain a heuristic password meter, we augmented MinAuto by taking the minimum of PGS's MinAuto and Wheeler's zxcvbn to be our baseline [7]. We call this baseline **PGS++**. The *most recent* password breach on which PGS++ models were trained is the 2012 Yahoo breach [69]. As a result, despite its state-of-the-art models, PGS++ might miss modern references in newer passwords.

We set a guessing cutoff of 10^{15} guesses, which is slightly above where Florêncio et al. [1] estimated a password would likely withstand an offline attack. Many factors influence cutoffs, such as password length, allowed characters, password storage function, and the adversary's hardware [70], [71], [72]. We chose our specific cutoff pragmatically as we were ultimately interested in accurately assessing strength. This is inherently harder to judge for very infrequently chosen (high guess number) passwords [19]. When PGS++ or an LLM reports a guess number above 10^{15}, we set it to $10^{15} + 1$ to prevent our metrics from being skewed by outliers.

3.4. Key Metrics

To characterize the relationship between the LLM and PGS++ strength ratings, we used Spearman's rank correlation coefficient (ρ), which ranges from -1 to 1. This test measures the relative ordering of passwords when listed from weak to strong under a specified model. As recommended by Golla et al. [3], we used the *weighted* version of this test, which applies greater penalties to mistakes in the relative ordering of weak passwords than of strong passwords [3]. A ρ of 1 indicates a perfect monotonic relationship between the LLM and PGS++, and a ρ of 0 indicates no correlation. To build intuition, a password model that mistakenly rates 5% of weak passwords as strong has $\rho = 0.70$, whereas $\rho = 0.72$ if 5% of strong passwords are rated as weak [3].

We also measured the discrepancy between the LLM and PGS++ guess numbers. The interpretation of differences between LLM guess numbers and PGS++ guess numbers is important, yet nuanced. It is far worse for an LLM to rate a weak password as strong than a strong password as weak. Users who erroneously believe that their weak password is strong may continue to use that weak password. As a result, LLM **overestimates** of password strength relative to PGS++ are especially problematic and unsafe. **Underestimates** of password strength (the LLM rating a password as weaker than PGS++) are potentially annoying for users, yet less detrimental to security. Underestimates also occur when the LLM has more recent cultural information and thus correctly rates a contemporary password as weaker than PGS++ does.

4. LLM Prompting for Password Strength

As our first approach, we investigate if a pretrained LLM can accurately rate a password's strength directly by *prompting* a foundation model that has not been fine-tuned.

4.1. Prompt Variants

We varied LLM prompts across three key dimensions in a full-factorial design (i.e., trying all possible combinations).

(Dimension 1) Output Format: We tested two variants. Mirroring the output of PGS (and the literature), we requested **guess numbers** as follows: "*We are interested in calculating the guess number of a password. The guess number is used to determine if a password is strong or not. Specifically, a password's guess number is defined as 1 plus the number of passwords that are weaker (easier to guess) than that password. Thus, stronger passwords will have higher guess numbers.\n Question: What is the guess number of the password on the next line? Even if you cannot answer exactly, provide your best guess.\n <password>*."

Mirroring user-facing strength estimates, we also requested ratings on a **scale**: "*We are interested in rating the strength of a password on a scale from 1 to 100. A rating of 1 indicates a very weak password (one that is very easy to guess), while a rating of 100 indicates a very strong password (one that is very hard to guess). Thus, stronger passwords will have higher ratings.\n Question: On this scale of 1 to 100, what is the rating of the password on the next line? Even if you cannot answer exactly, provide your best guess.\n <password>*."

To ensure that data analysis is consistent regardless of output format, we map 1–100 ratings back to guess numbers using the inverse of the equation in the following section.

(Dimension 2) N-shot Learning: The idea of n-shot learning is to include n examples alongside a base prompt. We set $n = 20$, appending 20 passwords sampled from XATO alongside their strength estimate, tab-separated with each password on a new line. We used our non-deduplicated sample of 10,000 XATO passwords (see Section 3.2).

To give the model an illustrative set of examples, we used stratified sampling. Specifically, the twenty examples represented strength at five-percentile intervals (i.e., the first example was sampled from the weakest 5% of passwords, the second from the 5%-10% weakest passwords, etc.) As applicable, the strength estimate was either the MinAuto guess number (as described in Section 3.3) or a 1–100 scale rating. We remap a guess number g to the range 1–100 as follows, then revert scale ratings back to guess numbers for analysis by applying its inverse:

$$\begin{cases} \max(\frac{20}{3}\log_{10}(g), 1) & 1 \leq g < 10^{15} \\ 100 & g \geq 10^{15} \end{cases}$$

(Dimension 3) Chain of Thought (CoT): Widely used CoT prompting methods ask an LLM to reason about its answer when producing a response. We tried asking an LLM to perform CoT reasoning explicitly, implicitly, or not at all. For **explicit CoT**, we provided a list of relevant heuristics that would affect password strength (e.g., the presence of keyboard patterns) and ask the LLM to reason about them when producing its answer. Our description (Appendix A.2) used the heuristics employed by the zxcvbn password meter [7]. For **implicit CoT**, we again asked the LLM to reason about its answer, yet we did not specify any heuristics.

TABLE 1: Weighted Spearman's ρ between prompted LLM (best-performing prompt) and PGS++ guess numbers.

Evaluation Set	Claude Opus 4	Gemini 2.5 Pro	GPT-4.1	Llama 4
Linkedin	0.90	0.82	0.88	0.88
Bookcrossing	0.91	0.85	0.91	0.95
CMU-Basic	0.80	0.80	0.80	0.75
CMU-LongBasic	0.68	0.70	0.64	0.64
CMU-Complex	0.42	0.53	0.53	0.37
CMU-LongComplex	0.48	0.57	0.60	0.58

4.2. Results

When prompting optimally, LLMs correlated more strongly with PGS++ for passwords created under basic composition policies and less strongly for complex policies (Table 1). We consider the optimal prompt to be the one with the highest average correlation coefficient for a model. For all models, ρ ranged from 0.75 to 0.95 for the Bookcrossing, Linkedin, and CMU-Basic sets. In contrast, ρ dropped to 0.37–0.60 for the Complex and LongComplex sets.

We observed that LLMs excelled when passwords resembled natural language. CMU-Basic and Linkedin contained very word-like passwords (e.g., `water_lily`). CMU-LongComplex passwords somewhat resembled natural language (e.g., `Margar1taV$lle`), yet often had numbers or punctuation in surprising places due to leet substitutions. Many CMU-Complex passwords (e.g., `k9io45vbAV!!`) did not resemble natural language.

Although the LLMs agreed with PGS++ on the relative ordering of passwords, the numeric difference in strength scores differed based on the password (Figure 1). We did not observe clear trends for medium-strength or weak passwords. However, we found the LLMs consistently guessed passwords that PGS++ rated as strong. To understand how LLMs compared to PGS++'s models, we analyzed the 100 passwords with the greatest and smallest median discrepancy in guess numbers. We calculated this across all LLM prompts and models to ensure our analysis revealed properties of LLMs, not specific prompts. We then qualitatively coded these passwords. Two coders inductively developed a codebook (Appendix C) and then independently coded the 100 passwords in each set. Afterwards, the coders met to discuss discrepancies in the codes applied and jointly resolved them to reach full code agreement. We report properties of high- and low-discrepancy passwords as percentages.

4.2.1. Gaps Identified. PGS++ did not guess any of the 100 high-discrepancy passwords, likely due to training data

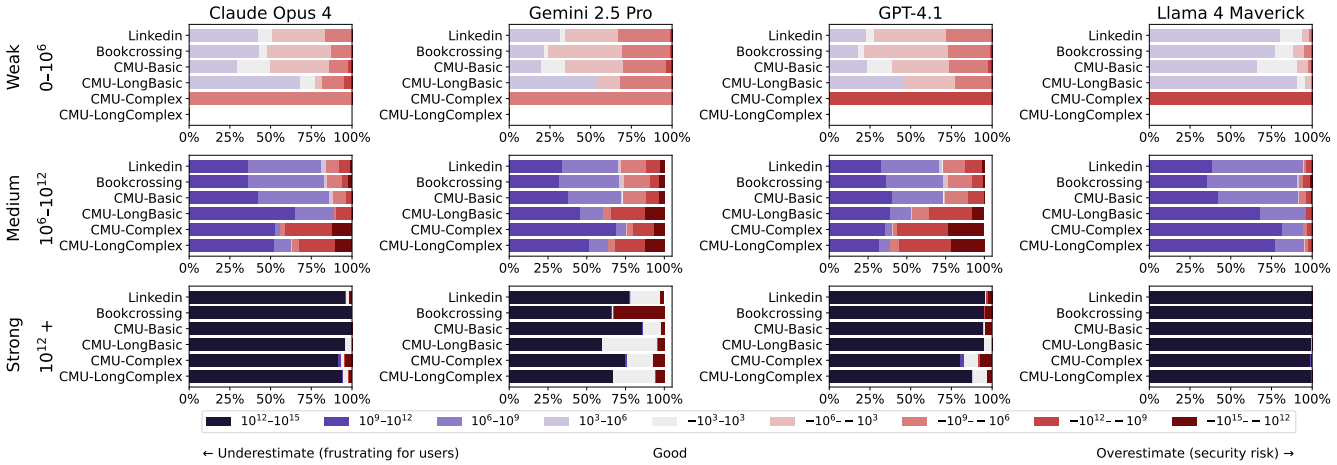

Figure 1: The distribution of discrepancy for our best-performing models and PGS++ (each using the prompt variant with highest average ρ). Purples denote underestimates. Increasingly dark reds indicate increasingly severe overestimates, the most critical for security. Note that no CMU-LongComplex passwords had a guess number under 10^6.

TABLE 2: The difference in Spearman's ρ if one prompting strategy is used over another (denoted *from* → *to*). Better outcomes are highlighted in green, and worse in red.

Evaluation Set	Scale → Guess #	0-Shot → 20-Shot	No CoT → Imp. CoT	No CoT → Exp. CoT
Linkedin	-0.01	-0.01	0.04	-0.01
Bookcrossing	-0.04	0.01	0.00	-0.05
CMU-Basic	0.06	0.07	0.02	-0.02
CMU-LongBasic	0.02	0.06	0.03	-0.01
CMU-Complex	0.04	0.09	-0.02	-0.09
CMU-LongComplex	0.04	0.02	0.00	-0.01

limitations. The LLMs filled gaps related to PII (48%) and cultural references (33%). Of the PII contained in these passwords, 5% were email addresses, 16% were Western names (e.g., `Karleen-Toma1`), and the other 27% were non-Western names (e.g., `Ahmed=mahmoud1`). Among cultural references, 9% were non-specific (e.g., a quote from John 3:16), 16% were pop-culture references (e.g., `baconnatorbaconnator`), and 8% referenced platforms or websites that were much less popular or did not exist before PGS++'s data leaks (e.g., `FaceBookTwitter`).

LLMs and PGS++ typically agreed when passwords were obviously very weak or very strong. 87% of low-discrepancy passwords were extremely weak (e.g., `password`). The remaining 13% appeared to be randomly generated (e.g., `e#%^2jL93xV,ZhK^)`.

4.2.2. Impact of Prompt Variants. Table 2 shows how the median ρ changes based on each prompt dimension. We discuss how this changes from the mean (which is more vulnerable to outliers, representing model brittleness due to the prompt) when appropriate. We denote a change with Δ.

On our first dimension, the output format (guess number vs. 100-point scale), we found that the guess number approach was usually more successful than the scale approach. While its effect is small in Table 2 ($\Delta \in$ [-0.04, 0.06]), its effect was much larger when examining means ($\Delta \in$ [-0.12, 0.81]), suggesting that it also stabilizes model outputs.

On our second dimension, we found that 20-shot prompting (the inclusion of 20 example passwords and their MinAuto strength estimates) was almost universally beneficial. Similar to the output format, 20-shot prompting helped stabilize model behavior: $\Delta \in$ [0.08, 0.44] for all test sets except Linkedin, which had a $\Delta = -0.27$ (worse behavior). We speculate that this could be because the passwords in Linkedin were almost universally weak, whereas our examples included both weak and strong passwords.

On our third dimension, we found that implicit CoT generally helped ($\Delta \geq 0$ for all test sets except CMU-Complex). However, the means remained fairly close to 0, suggesting that implicit CoT did not meaningfully reduce model brittleness. Explicit CoT was more nuanced; while it generally did not help ($\Delta < 0$ for all test sets), it *did* reduce the effect of outliers for all test sets other than Linkedin and CMU-Complex. Precisely, when examining means, $\Delta > 0$ for all sets other than Linkedin ($\Delta = -0.20$) and CMU-Complex ($\Delta = -0.11$).

Despite these trends, the specific prompt with the largest average ρ depended on the LLM. For Gemini 2.5 Pro, it was "Guess Number, 20-Shot, Explicit CoT." For both GPT 4.1 and Claude Opus 4, it was "Scale, 20-Shot, Implicit CoT." For Llama 4, it was "Scale, 0-Shot, Explicit CoT."

5. Technical Challenges with Applying Fine-Tuned LLMs for Password Strength Modeling

Probabilistic models have traditionally been used to model password strength by training them on passwords, then simulating a guessing attack on a target password breach [4]. Simulation uses the Monte Carlo approach [17], [19]. With this approach, a large sample of passwords are

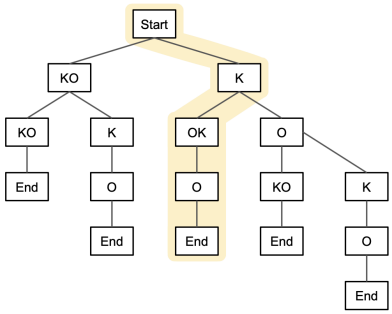

Figure 2: Prefix tree for KOKO using GPT-2 XL's tokenizer. The canonical tokenization is highlighted.

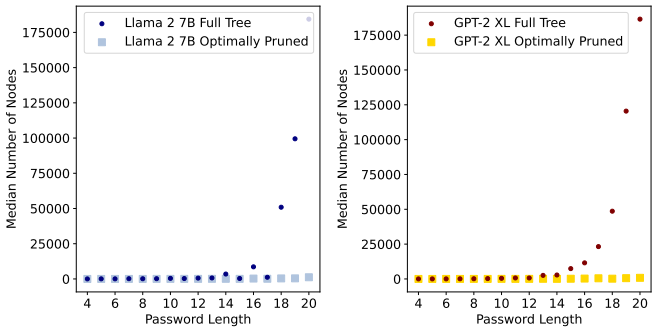

Figure 3: The median number of nodes evaluated in a password's prefix tree if the full tree is evaluated or our best pruning strategy is used for Llama 2 7B and GPT-2 XL (10 epochs).

randomly generated from the model. Then, the model is used to calculate the probability of a target password. The guess number for the target is calculated using the probabilities of the generated passwords. However, the way that LLMs ingest and output (generate) passwords is different from prior models, introducing novel technical challenges. We proceed by outlining how LLM fine-tuning works, then highlight the challenges encountered.

LLM Fine-Tuning Details: Like prior models, LLMs can be trained on passwords. Unlike prior models, which tokenize input and output passwords character-by-character, LLMs both ingest and output groups of characters called **tokens**. For modern LLMs, a token can represent a word, a part of a word, or even a single character. When passwords are fed into the LLM for training, they are first **tokenized**, or split into groups of characters. For example, "dog" might be tokenized as a single token, [dog], or instead as three individual letters depending on the particular LLM's tokenizer. Tokenization of input data is deterministic for a given LLM. We call this deterministic tokenization a password's **canonical tokenization**. The LLM updates its weights according to the canonical tokenizations of passwords during training.

While an input password's tokenization is deterministic, LLMs' output tokenizations are stochastic. While "dog" must be input to the LLM as the canonical tokenization, an LLM may generate "dog" via many different combinations of tokens, each of which may be generated with a different probability. For example, [dog] may have probability 0.05, while [d][og] may have probability 0.01. This state of affairs introduces a complication. If someone wants to know the probability of a password, they must decide for which tokenization to return the probability. We observe that the probability of generating a password equals the sum of the probabilities of each of its possible tokenizations. We call this the **total probability mass** of a password.

(Challenge 1) Computational Cost: The total probability mass of a password is calculated after an LLM is fine-tuned. First, one enumerates all valid tokenizations of a password. This could be done efficiently by using a prefix tree to eliminate duplicate computation (Figure 2). Even with the tree optimization, we observe the amount of required computation grows roughly exponentially with a password's length (Figure 3).

A natural question is whether *all* possible tokenizations actually need to be evaluated. If instead a subset of tokenizations captured a consistent fraction (e.g., 1/2) of the total probability mass, the probability of that subset could be rescaled by a constant (e.g., 2) to approximate the total probability mass. We call an algorithm that determines which subset of tokenizations to compute a **pruning strategy**.

(Challenge 2) Sampling Passwords, Not Natural Language Overall: Guess number estimation requires that passwords are randomly sampled from a model [17]. However, LLMs will not inherently access the "passwords" part of their knowledge during random sampling and may instead generate news articles, tweets, or other natural language.

Unlike Challenge 1, solving this is straightforward. We employ prompt fine-tuning to teach an LLM to distinguish passwords from the rest of its pretraining. Prompt fine-tuning involves prefixing and suffixing each password with a prompt, then fine-tuning on the pre-processed password [73], [74]. Afterwards, one prompts the model with the prefix to "inform" the model that it is generating or evaluating a password. A generated password is complete when the suffix is produced (see Appendix A.1 for specific tokens). We devote the rest of this section to Challenge 1.

5.1. Simple Pruning Strategies

We first investigate whether two simple pruning strategies can be used. First, we consider the pruning strategy that considers only the **Canonical Tokenization**. Second, we consider **Character-Wise Pruning**, which calculates the probability of a password using its character-by-character tokenization, as in past probablistic password models.

5.1.1. Methods. To evaluate different strategies, we first calculate the total probability mass for various passwords. The high computational cost of doing so necessitated the creation of a smaller dataset. We drew passwords from the Rockyou [75], 000webhost [76], and XATO [51] sets. Because computational costs grow sharply with password length, we performed stratified sampling based on length. We sampled

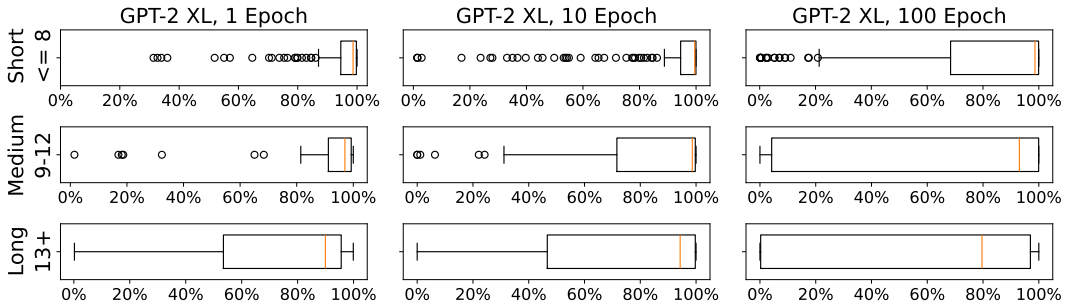

Figure 4: The proportion of probability mass the canonical tokenization contributes when compared to the full probability mass over all tokenizations, for three LLMs exposed to increasing amounts of training. See Section 5.1.1 for LLM details.

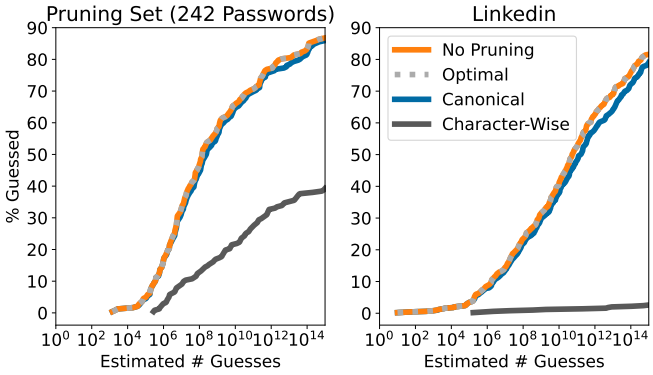

Figure 5: Guessing curves estimated using Monte Carlo simulation for various pruning strategies. Note that the "Optimal" pruning strategy overlaps heavily with "No Pruning."

passwords such that 25% each were drawn from (i) Rockyou, (ii) 000webhost, (iii) XATO passwords on which the model was trained, and (iv) XATO passwords *not* used for model training. We sampled passwords of each length (4 to 20 characters), inclusive, using $y = \lceil (1000/n^2) \rceil$ to set the number of passwords, y, sampled for each length n. We sampled 242 passwords in total, and we fine-tuned three different GPT-2 XL models, varying in the number of epochs for which we fine-tuned. Calculating the total probabilities across the 242 passwords took 1,066,983 forward passes per GPT-2 XL instance (one forward pass/node), taking around 2.5 days on an NVIDIA RTX 6000 (6GB VRAM).

We fine-tuned Llama 2 and GPT-2 XL LLMs on our 100k, non-deduplicated XATO subset (see Section 3) for 1, 10, and 100 epochs to ensure pruning strategies are robust to model fit. We use an arbitrary precision floating point class to minimize the impact of floating point error.

5.1.2. Results. In this section, we summarize our results.

Canonical Pruning was insufficient. As Figure 4 illustrates, the canonical tokenization varied significantly in the fraction of a password's total probability mass captured. While this variance was correlated with password length, it also increased as the models seemed to overfit past 10 training epochs. Variance due to fit/training is undesirable because strength judgments are most useful for passwords

TABLE 3: Correlation (ρ) between no-pruning and pruning strategies' guess numbers estimated for Linkedin passwords. Optimal Pruning has $\rho = 1$; Canonical's ρ rounds to 1.

Evaluation Set	Optimal Pruning	Character-Wise	Canonical
Linkedin	1.00	0.44	0.99
Pruning	1.00	0.25	1.00

not previously observed by the model. A password model using only the canonical tokenization would likely be very brittle in practice. Even very short passwords had wide variance; `Coal`'s canonical tokenization represented only 11% of the total probability mass for our 100 epoch model.

Character-Wise Pruning was insufficient. There was high standard deviation in the proportion of probability mass captured (>9% or >35% for all GPT and Llama model instances, respectively). Manual examination suggested that probability is mostly a function of password length. Weak-but-long passwords like `princess123456` were erroneously reported as strong. For GPT-2 XL (10 epochs), the password had rank 200/242 under Character-Wise Pruning, but rank 14/242 when looking at the total probability mass. We substantiated this by calculating weighted Spearman's ρ between this approach and the full probability mass: $\rho \in [0.68, 0.82, 0.84]$ for 1, 10, and 100 epochs, respectively. Surprisingly, Zou et al.'s recently proposed PassLLM scheme [38] relies exclusively on Character-Wise Pruning.

Effect of Pruning on Guessing Curves. We characterize how pruning strategies affect estimated guess numbers, finding that our "Optimal Pruning" strategy (detailed in Section 5.3) has low error, Canonical Pruning has moderate error, and Character-Wise Pruning has substantial error (Figure 5). We estimated guessing curves using GPT-2 XL fine-tuned for 10 epochs with a sample size of 1,000 passwords. We evaluated over multiple password sets to ensure we did not test on the same passwords used to select an ideal strategy. Virtually none of Linkedin was guessed with Character-Wise Pruning, although more was cracked for our pruning set. Canonical Pruning somewhat mirrored the "No Pruning" approach, albeit with observable error. We also quantified the degree to which pruning strategies caused inversions in the relative ordering of passwords (Table 3). There were

no inversions for our Optimal Pruning strategy ($\rho = 1$), marginal changes for Canonical Pruning ($\rho \geq 0.99$), and major changes for Character-Wise Pruning ($\rho \leq 0.44$).

5.2. Other Pruning Strategies

As the two aforementioned strategies were unsuitable, we explored other pruning strategies. These strategies aim to prune paths in the prefix tree of all possible tokenizations that are unlikely to contribute meaningfully to the password's total probability mass. Specifically, each strategy identifies nodes to prune from the prefix tree, and all paths (i.e., tokenizations) containing that node are subsequently ignored in estimating that password's total probability mass. We did not consider strategies that partially complete paths, although future work could consider this. Some of the pruning strategies are parameterized. We chose parameters based on empirical testing, rather than exhaustive exploration.

Path Threshold Pruning: Our simplest algorithm states that we should not consider a path if the total probability mass from root to a node becomes $< t$ for some user-defined t. In our tests, we tried $t \in \{1^{-20}, 1^{-25}, 1^{-30}, 1^{-35}\}$.

Depth Pruning: We speculated that very long tokenizations may not contribute meaningfully to a password's total probability mass as the probabilities of each token are multiplied. We defined Depth Pruning to prune nodes that occur deeper than some user-defined depth, d. We tried $d \in \{4, 5, 6, 7, 10, 15, 20\}$.

Canonical Length Pruning: Alternatively, one could consider that the canonical tokenization often contributes substantial probability mass. Thus, longer tokenizations (i.e., those containing more tokens) might be less important. Canonical Length Pruning prunes a path once it exceeds some multiple t of the canonical tokenization's length. We tried $t \in \{1, 1.25, 1.5, 1.75, 2, 4, 8\}$.

Canonical Probability Pruning: Since the canonical tokenization often contributes substantially to the total probability mass, perhaps only a relatively small set of other high-probability tokenizations would need to be considered. Canonical Probability Pruning prunes nodes once the accumulated probability on a path from the root to the current node times t is less than the canonical tokenization's probability. We tried $t \in \{\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 1.25, 1.5, 1.75, 2\}$.

Floating Point Error Pruning: IEEE-754 floating point numbers can only provide a certain amount of precision for any given exponent. Since each node in a tree has a value at most 1, each multiplication operation results in increasingly large negative exponents, in turn causing a number of bits of the product to become insignificant. Floating Point Error Pruning prunes a path once the product's floating point value does not change as a result of evaluating the node.

5.3. Pruning Results

In this section, we only report on the optimal parameterization we determined empirically for each strategy. Appendix C presents the alternate parameterizations for our best strategy, while our OSF repository contains analogous graphs for all other strategies. We evaluate success by checking whether a large proportion of a password's tree is pruned while capturing a *predictable, consistent* fraction of the password's total probability mass.

TABLE 4: Pruning strategy performance for **GPT-2 XL** (10 epochs) by password length. Higher percentages are better.

Pruning Strategy	Password Length	Median % Pruned	Std. Dev. in % of Mass (Pruned/Full)
Canon Prob 2.0	≤ 8	25.29	0.13
	9-12	82.77	0.47
	13+	96.53	0.34
Canon Len 1.25	≤ 8	17.11	0.25
	9-12	24.64	0.04
	13+	63.53	0.03
Thresh 1e-35	≤ 8	0.00	0.00
	9-12	13.81	0.00
	13+	75.65	0.01
Float Error	≤ 8	0.14	0.00
	9-12	4.30	0.00
	13+	25.44	0.00
Depth 10	≤ 8	0.00	0.00
	9-12	0.00	0.00
	13+	15.15	0.15

TABLE 5: Pruning strategy performance for **Llama 2 7B** (10 epochs) by password length. Higher percentages are better.

Pruning Strategy	Password Length	Median % Pruned	Std. Dev. in % of Mass (Pruned/Full)
Canon Prob 2.0	≤ 8	56.25	0.06
	9-12	82.43	0.22
	13+	93.63	0.26
Canon Len 1.25	≤ 8	22.90	0.11
	9-12	18.47	0.00
	13+	27.91	0.00
Thresh 1e-35	≤ 8	0.00	0.00
	9-12	15.94	0.00
	13+	75.66	0.00
Float Error	≤ 8	0.00	0.00
	9-12	2.44	0.00
	13+	47.05	0.00
Depth 15	≤ 8	0.00	0.00
	9-12	0.00	0.00
	13+	0.01	16.66

Canonical Probability Pruning performed the best (Figure 3). In the prefix trees for passwords in our evaluation set, it pruned a median number of nodes greater than 56% regardless of password length, training epochs, or model family (Table 4 and Table 5.) It also had consistent behavior: its standard deviation in the proportion of probability mass captured was always $\leq$ 1.64%. Long passwords are the most computationally heavyweight since the number of paths (tokenizations) increases roughly exponentially with length. For long passwords, Canonical Probability Pruning pruned a median of over 94% of GPT-2 XL's nodes and over 84% of Llama 2's nodes regardless of whether the model was trained for 1, 10, or 100 epochs. At its worst (for the

100 epoch models), Canonical Probability Pruning reduced compute costs by 96.8% for both GPT-2 XL and Llama 2.

Probability-based pruning methods tended to be more successful than length-based methods. Depth and Canonical Length Pruning performed among the worst, suggesting that the length of a tokenization only loosely correlates with probability. Depth Pruning was the only strategy with different optimal parameterizations across model families, which we suspect may have been due to differing tokenizers.

Relative approaches were more successful than threshold-based approaches, and the canonical tokenization's probability was far more useful for pruning than its length. Canonical Probability Pruning outperformed Floating Point and Path Threshold Pruning, and Canonical Length Pruning similarly outperformed Depth Pruning. Since Canonical Probability Pruning was more successful than Canonical Length Pruning (and the most successful overall), we remark that the main benefit of the canonical tokenization is that it is often representative of "meaningful" probability mass. However, training interventions (which inherently change the probabilities assigned to certain tokenizations) may reduce the utility of the canonical tokenization. We observed this happening during ablation experiments for a non-pretrained model (see Section 6).

6. Fine-Tuned LLMs for Password Strength

Our optimal pruning strategy (Section 5.3) makes it computationally feasible to simulate an offline guessing attack. This enabled us to understand passwords an attacker might guess with an LLM, and how a defender could use one for password strength modeling. To understand the factors affecting LLM behavior, we performed ablation experiments.

6.1. Fine-Tuning Ablation Variants

We identified six relevant fine-tuning parameters from related work on NLP and password modeling. We then fine-tuned an LLM with that variant to understand the effect of that parameter. We varied parameters based on hypotheses about overfitting, prior knowledge, model ability, and how data should be formatted. Namely, we varied the size of the LLM (larger is thought to be better [45], [77]), whether or not the model has been pretrained (pretrained is thought to be better [78], [79]), the number of fine-tuning epochs (which affects LLM fit [4], [80]), the quantity of fine-tuning data (more data is often better [81], [82]), whether or not the fine-tuning data was de-duplicated (de-duplicating is thought to be better [54], [55]), and the prompt used for finetuning (a natural-language prompt may help the LLM relate its fine-tuning to its pretraining [73], [74], [83]).

We set our **Baseline** to use the *largest pretrained* version of the LLM we felt a well-equipped research group could run locally (GPT-2 XL, Llama 2 7B) and fine-tuned it for *10 epochs* on a *greater* quantity of *de-duplicated* fine-tuning data (100k) that has been formatted with a *natural language* prompt. Our **Smaller Model** LLM utilizes GPT-2's "Base" size; we omit this variant for Llama 2 since 7B is the smallest model size. **No Pretraining** means the model is not pretrained, **Five Epochs** and **One Epoch** indicate models fine-tuned for five and one epochs, respectively, **Less Data** means fine-tuned on 10k passwords, **Non-deduplicated** means fine-tuned on passwords not de-duplicated prior to downsampling, and **Random Prompt** uses random tokens for prompt-finetuning (see Appendix A.1). We sampled 15,000 passwords to estimate guess numbers.

TABLE 6: Spearman's ρ for the best fine-tuned variants.

Evaluation Set	GPT-2 XL (One Epoch)	Llama 2 7B (One Epoch)
Linkedin	0.93	0.75
Bookcrossing	0.97	0.98
CMU-Basic	0.91	0.90
CMU-LongBasic	0.79	0.80
CMU-Complex	0.64	0.59
CMU-LongComplex	0.64	0.61

6.2. Results

Surprisingly, the single epoch variants performed the best (Table 6) and outperformed their prompted counterparts (Table 1) despite being billions of parameters smaller. See Appendix C for the other variants. GPT-2 XL fine-tuned for one epoch was our best model. Both GPT and Llama had excellent performance on basic passwords ($\rho \geq 0.79$) for the CMU-Basic, LongBasic, and Bookcrossing sets, and worse performance on Complex and LongComplex ($\rho \leq 0.64$).

6.2.1. Gaps Identified. Our best LLM outperformed existing models at the beginning of a guessing attack, especially for weak passwords (Figure 6). Notably, the LLM outperformed Melicher et al.'s Neural Network [4] for all sets except CMU-Complex. The greatest improvements were for Bookcrossing and LongBasic, where the LLM guessed 20% and 5% more, respectively. Looking at PGS++ broadly, the LLM guessed roughly 10% or more of Linkedin, Bookcrossing, and CMU-Basic within 100 guesses. PGS++ guessed <5% each. The LLM approached PGS++'s performance at 10^{15} guesses for the same sets.

As when prompted, the LLM struggled with complex passwords. Unlike when prompted, the LLM correlated better with PGS++ for LongBasic passwords. However, it struggled to guess them. This suggests that the LLM can recognize the relative differences in strength for LongBasic passwords, but overall does not rate them as likely. Precisely, the LLM guessed roughly 40% of the Complex, LongComplex, and LongBasic sets as compared to PGS++, which guessed approximately 60% or greater for each.

Using the same approach as in Section 4.2, we manually analyzed 100 passwords each where LLMs and PGS++ agreed or disagreed. Unlike in that section, we restricted our analysis to our best-performing LLMs when analyzing LLMs' ability to fill gaps. We did this as some LLMs performed poorly (e.g., the "no-pretraining" variant.)

We found that the LLMs identified similar gaps when fine-tuned as when prompted. They guessed passwords

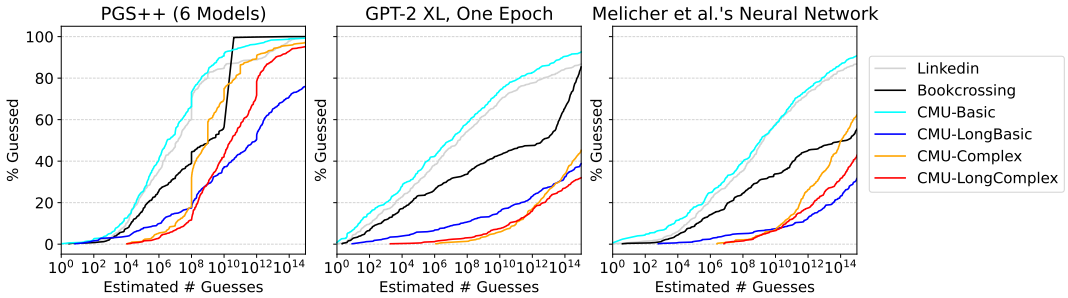

Figure 6: Simulated guessing curves for PGS++, the best performing fine-tuned LLMs, and the neural network approach.

TABLE 7: Change in ρ as training epochs are reduced.

Model Family	Evaluation Set	10 (Baseline) → 5 Epochs	10 (Baseline) → 1 Epoch
GPT-2	Linkedin	0.03	0.09
	Bookcrossing	0.01	0.02
	CMU-Basic	0.02	0.02
	CMU-LongBasic	0.03	0.05
	CMU-Complex	0.07	0.11
	CMU-LongComplex	0.07	0.18
Llama 2	Linkedin	0.19	0.26
	Bookcrossing	0.02	0.04
	CMU-Basic	0.01	0.00
	CMU-LongBasic	0.01	0.07
	CMU-Complex	0.05	0.08
	CMU-LongComplex	0.12	0.19

TABLE 8: Change in ρ from Baseline due to training data.

Model Family	Evaluation Set	No Pre-training	Non-Dedup.	Less Data
GPT-2	Linkedin	-0.06	0.04	0.08
	Bookcrossing	-0.09	-0.13	0.01
	CMU-Basic	-0.07	-0.02	0.01
	CMU-LongBasic	-0.16	0.09	-0.01
	CMU-Complex	0.09	0.02	-0.07
	CMU-LongComplex	0.03	0.12	0.02
Llama 2	Linkedin	0.48	0.32	0.28
	Bookcrossing	0.02	-0.38	0.02
	CMU-Basic	-0.05	0.00	-0.01
	CMU-LongBasic	-0.26	0.04	0.07
	CMU-Complex	-0.28	0.03	-0.05
	CMU-LongComplex	-0.30	0.19	0.15

containing non-Western names (28%), short phrases like `ilovegoingtothebeach` (20%), pop-culture references (18%), and passwords predominantly in a foreign language (5%), such as `11ranzig` ("rancid" in German). However, fine-tuned LLMs filled *fewer* gaps than when they were prompted: 682 passwords had consistently lower guess numbers than PGS++ when fine-tuned, vs. 2,457 when prompted.

As when prompted, LLMs and prior models agreed on what passwords were very strong (100% of low-discrepancy passwords). Unlike when prompted, moments where PGS++ and the LLMs disagreed almost universally reflected LLM errors. LLMs predominantly struggled with Complex passwords: 76% had four or more character classes, 12% had three or more, and 12% had fewer than three (Table 6). Semantically, the LLMs struggled to guess passwords that appeared to be randomly generated (36%) or contained leet substitutions (28%), name-based PII (14%), pop-culture references (3%), or other cultural references (3%).

6.2.2. Impact of Fine-Tuning Variants. Fewer epochs of training were beneficial, and the "One Epoch" variant performed the best (highest average ρ) for both GPT-2 and Llama 2 (Table 7). Training for fewer epochs predominantly improved performance on Linkedin, CMU-Complex and CMU-LongComplex passwords. We speculate that this could be due to weaker passwords dominating password breaches, causing models to find Complex passwords increasingly unlikely when trained for longer.

Of our data interventions, using a pretrained model, less data, and data with duplicate passwords were all beneficial (Table 8). GPT-2 suffered performance degradation when not pretrained ($\Delta \in [-0.16, -0.06]$) for all non-Complex datasets, while Llama-2 suffered broadly for all sets other than Linkedin and Bookcrossing ($\Delta \in [-0.30, -0.05]$). Curiously, LLMs did not seem to need much password data: performance increased when training on 10k vs. 100k passwords for all test sets other than CMU-Complex. Unlike for many other NLP tasks [55], LLMs did not seem to benefit from training on de-duplicated passwords. *Not* de-duplicating improved performance for long and complex passwords ($\Delta \geq 0.12$ for CMU-LongComplex), but reduced performance for short, basic passwords ($\Delta \in [-0.38, 0]$ for Bookcrossing, CMU-Basic). De-duplication is often done to prevent LLM memorization. We speculate that some memorization could be useful [84] because of natural overlap in weak passwords (e.g., "password" vs. "password1").

TABLE 9: Change in ρ from Baseline due to approach.

Model Family	Evaluation Set	Random Prompt	Smaller Model
GPT-2	Linkedin	0.02	0.06
	Bookcrossing	0.00	-0.01
	CMU-Basic	0.00	-0.02
	CMU-LongBasic	0.04	0.06
	CMU-Complex	0.07	0.01
	CMU-LongComplex	0.08	0.20
Llama 2	Linkedin	0.37	n/a
	Bookcrossing	0.01	n/a
	CMU-Basic	0.01	n/a
	CMU-LongBasic	0.03	n/a
	CMU-Complex	0.04	n/a
	CMU-LongComplex	0.09	n/a

Reducing an LLM's sample efficiency via a smaller LLM and random prompt helped (Table 9). The smaller

TABLE 10: An example of GPT-4.1's *abstract recommendations* for the input password `moonlight2111`.

- Add special characters (e.g., !, @, #) to increase complexity: moonlight2111! - Use a mix of uppercase and lowercase letters: MoonLight2111 - Replace part of the password with unrelated words or phrases to increase length and unpredictability: moonlight2111Sunrise$"

TABLE 11: An example of GPT-4.1's *concrete modifications* for the input password `peppy123`.

Here are three improved versions of the password: 1. **Add special characters:** peppy!123$ 2. **Use a mix of uppercase and lowercase letters:** PePpY123 3. **Make it longer and use unrelated words (passphrase):** Peppy123!BlueHorse

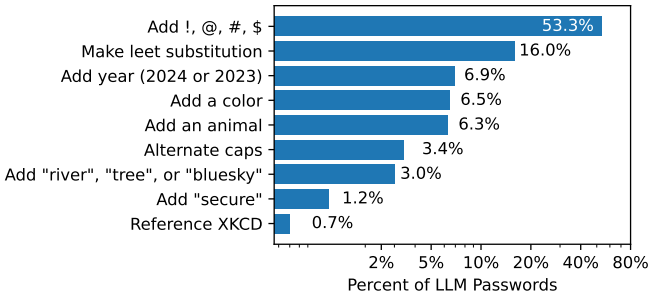

Figure 7: Edits LLMs made to make passwords "stronger."

LLM suffered a small performance degradation on the CMU-Basic and Bookcrossing sets ($\Delta = 0.01, -0.02$ respectively), yet performed better on the remaining test sets, particularly LongComplex passwords ($\Delta = 0.20$). This implies that even smaller, older models like GPT-2 Base may be viable for password modeling. The random prompt was beneficial ($\Delta \geq 0$) for all test sets. We caution that reducing sample efficiency, data quantity, and training epochs at the same time may result in the LLM failing to fit.

7. LLM Prompting for Password Feedback

Whereas some existing (non-LLM-based) password meters give users feedback on improving their password [3], [7], [56], these approaches are fairly basic and brittle. LLMs plausibly could provide richer and more semantically meaningful feedback. To our knowledge, no prior work has tested using LLMs to provide password feedback. In our final set of experiments, we prompted LLMs to give abstract **recommendations** (e.g., "describe how to improve the password `letmein1`") and make concrete **modifications** (e.g., "give three examples of stronger versions of `letmein1`").

7.1. Prompt Variants

For abstract recommendations, our base prompt was, "*In three bullet points, describe how the password on the next line could be modified to make it stronger. If the password is already strong, reply N/A.*" For concrete modifications, it was, "*Give three examples of stronger versions of the password on the next line, where each example illustrates a way to make the password stronger. If the password is already strong, reply N/A.*"

We also explored eight additional prompts (see Appendix A) that varied from the base in terminology ("*better*" instead of "*stronger*"), aim (requesting "*memorable*" passwords), diversity (e.g., "*using a different approach for each modification*"), and justification (e.g., "*explain why each example is stronger*"). We found that despite the different base prompts, the LLMs responded very similarly for different passwords (Table 10 and Table 11). Similarly, while there were slight variations in the frequency of some *types* of predictable suggestions (e.g., the tendency to add colors), the precise suggestions made (e.g., "blue" being the color that was added) did not often change across prompts. Accordingly, we present aggregate results across all prompts.

Table 11 shows a representative example of GPT-4.1's concrete modifications to the password `peppy123`. While the feedback provided is rich, personalized, and (in isolation) seems like good advice, *in aggregate* the suggestions were often predictable and repetitive, substantially limiting any potential security gains.

7.2. Results

We collected and analyzed data on LLMs suggesting 365,027 concrete modifications of passwords.

7.2.1. Modifications Made. Figure 7 summarizes common modifications. All percentages refer to the rate that a behavior was observed over these 365,027 passwords.

LLMs mimicked some human strategies for making "strong" passwords, such as using keyboard patterns. LLMs' most common edit was to add a symbol (62.1%), but they were almost always one of '!', '@', '#', or '$' (53.3%), which correspond to the symbols for the first four numbers on a qwerty keyboard. They often appeared in groups that further evoked keyboard patterns (11.8%), such as "!@."

Like humans, LLMs favored the '!' symbol and used symbols in predictable places. LLMs added '!' most frequently of all symbols (33.2%) and tended to add '!' to the end of a password, whereas most other suggested symbols occurred somewhere in the middle. Some introduced symbols were due to leet replacements (16.0% of passwords), such as changing 'a' to '@.'[1] Symbol placements sometimes mirrored how humans often delimit words in their passwords. The "-" symbol was frequently added (17.9%), but, from manual examination, mostly appeared on word boundaries (e.g., `Sunny-Skies-Cloud-04`).

Other LLM behaviors parodied humans, rather than mimicked: years were added—but were almost always 2023

1. We used Levenshtein edit distance to calculate this rate. Unlike for other suggestions, there may be noise associated with this statistic due to multi-character replacements. We defined a leet substitution to include relevant single-character replacements (e.g., 'i→!") as well as replacements where a single character was replaced with a group and the group includes a leet substitution (e.g., 'a→@pple").

or 2024. LLMs *alternated* capitalization over a password or introduced unusual words. LLMs added a year to 8.4% of passwords, typically to the end of passwords (5.4%). While humans often use their birth year, LLM years were almost always one of 2023 or 2024 (6.9%). While humans often capitalize the first letter of a password, LLMs *alternated* capitalization for an entire password 3.4% of the time. While humans often employ profanity or themes of love, LLMs employed nature themes, suggesting the words "river," "tree," and "bluesky" roughly 1% of the time each. LLMs sometimes simply added "secure" (1.2%) to the password. LLMs also referenced an XKCD comic [85] about secure password creation 0.7% of the time.

LLMs frequently added colors (6.5% of passwords), but only six of the 45 colors we observed were suggested often. When colors were suggested, they were typically blue (63.5%), purple (14.1%), green (4.9%), red (4.8%), orange (3.7%), or yellow (2.1%). Gemini, GPT, and Claude had similar rates in suggesting colors (8.62–9.24% of passwords), but Llama almost never suggested them ($<$1%). When colors were suggested, GPT added blue most frequently (91.1%), almost double the rate of Gemini (48.0%) and Claude (55.8%). For these models, purple was the second most frequent color, with Claude suggesting it (28.8%) at over triple the rate of Gemini (10.7%) and GPT (3.3%) when a color was suggested.

LLMs often added animals (6.3% of passwords). While we observed 224 unique animals being added to passwords, LLMs only suggested five of them frequently: tiger (21.9%), giraffe (16.6%), horse (13.5%), dog (7.8%), and cat (5.8%). Gemini and GPT suggested animals at similar rates ($\sim$9% of all passwords), while Claude and Llama did so at about half that rate ($\sim$3-4%). One might think that all occurrences of an animal could be due to a single model. However, it was more likely that three LLMs would produce an animal with similar frequency while a single model of the four behaved as an outlier. When LLMs suggested animals, they added "Giraffe" frequently (55.9% for Llama and 12.9-12.5% for two other models). Claude, though, almost never added it (1.1%). Tiger was similarly added frequently: 51.8% by GPT and 6.6–19.7% by the other models when an animal was suggested. However, Gemini added tiger 1.3% of the time (when an animal was a suggested.) Finally, horse was often suggested by Gemini (23.0%) and GPT (12.9%), but less than 4% of the time by the other two LLMs when an animal was suggested.

7.2.2. The Relative Strength of Suggested Passwords. We randomly sampled 40 suggested passwords for each combination of model, prompt, and test set. When LLMs gave multiple suggestions, we randomly picked one. Although it was very rare (5/240 combinations), we were sometimes unable to sample 40 suggestions. We drew the maximum number possible when this occurred. We did not observe any useful broader insights regarding LLMs' suggestion frequency or quality across test sets, models, or prompts. We sampled 9,522 passwords in total and obtained PGS++ guess numbers. We use the term ***significantly*** stronger to denote a difference in guess number ≥ 6 orders of magnitude. We caution readers that strength scores are based on a pre-LLM state of password creation.

Suggested passwords were usually stronger, but LLMs sometimes suggested much weaker passwords. 81.6% of feedback passwords were stronger, 7.8% were weaker, and the remaining 10.6% had the exact same guess number. 11.5% of passwords made weaker were significantly weaker. From manual inspection, virtually all of these passwords were shortened, sometimes substantially (`Thisisreallydumb.` $\rightarrow$ `T1RD!`). Perhaps most egregiously, the LLM sometimes disregarded a user's reasonably strong password to suggest an unrelated, weak password (`mturkPASSw0rd!` $\rightarrow$ `password`). Suggestions that were only slightly weaker had minor changes (`333linkedin` $\rightarrow$ `linkedin333`).

Unfortunately, prompts yielding significantly weaker passwords seem closer to how a human might solicit feedback. Our "modify directly" base prompt, "modify, but explain" and "modify (phrased like an end-user)" variants were most likely to result in a significantly weaker passwords: only 9.5% of significantly weaker passwords had a different prompt. Claude and Gemini were more likely to suggest a significantly stronger or weaker password. They produced 34.3% and 33.3% of significantly stronger or weaker suggestions, as opposed to GPT 4.1 (23.4%) or Llama 4 (9.0%).

Fortunately, when LLMs' suggested passwords had significantly different strength, strength improved to a greater degree than it decreased. Stronger passwords improved by a median of 3.4 orders of magnitude, while passwords weakened by a median of 2.4 orders of magnitude. 22.0% of strong suggestions were significantly stronger. Manual inspection suggests significant improvements were mostly due to length and/or the inclusion of capitalization (e.g., `housni25` $\rightarrow$ `housni25RiverSky`). However, the strength of some transformations may be overestimated by PGS++. `pa44w0rd` $\rightarrow$ `P@44w0Rd!2024` resulted in an improvement of 6.3 orders of magnitude, and `Gjq2l80M$` $\rightarrow$ `CorrectHorseBatteryStaple` increased by 6.0. The latter example shows LLMs ignoring the input password. This occurred mostly when the input was randomly generated or a known weak password like `princess17`.

8. Discussion

In this section, we discuss our experiments' implications.

The Role of LLMs vs. Existing Tools: LLMs exhibited unique strengths and weaknesses compared to previous password-strength models. They correctly rated as weak passwords containing modern cultural references and non-Western names, whereas all prior approaches failed to do so since such passwords were not frequent in their training data. In contrast, LLMs struggled to assess the strength of particularly long or complex passwords. Further, LLMs have far higher computational costs than previous models.

As a result, we do not envision LLMs completely replacing existing models, but instead being used in conjunction

with existing models. Most directly, LLMs may be suitable for defense-in-depth approaches for high-value accounts. Lightweight models could be run first. If all models report a password as strong, the LLM could be run as a final check.

However, since LLMs were able to flag as weak out-of-distribution passwords containing recent (but presumably common) cultural references, we envision the true power of LLMs will be in solving two complementary challenges. First, LLMs can fill gaps in existing models. In addition to passwords containing recent or complementary cultural references, LLMs could fill a unicode-shaped gap in the space of password models. ASCII-only password models are the norm; neither PGS nor zxcvbn handle strength estimates for passwords containing unicode [7], [67]. This shortcoming potentially forces users to use Latin characters, as opposed to their native script. Second, one can imagine adapting LLMs to password distributions for which little (or no) training data is available. A potential use case could involve an organization (e.g., a K-pop site) for which no similar organization has yet suffered a public data breach or that has passwords that are likely different from the breaches typically studied. An LLM could perhaps model password strength directly or instead generate passwords unique to that organization to create a blocklist. For all of the aforementioned uses, we expect that fine-tuning smaller LLMs would be sufficient, and these models are less computationally heavyweight than the current state-of-the-art LLMs. In fact, in many of our experiments, fine-tuning smaller, older LLMs worked better than prompting newer, larger LLMs, suggesting that these smaller LLMs contain the key semantic content needed to model passwords.

Guessing Passwords With LLMs: Our paper's focus is on defense: estimating the strength of a user's password to help the user improve their password. Practically speaking, we do not believe that adapting either our prompt-based or fine-tuning-based approaches would be especially useful for offensive security (i.e., guessing passwords). In a real guessing attack, an attacker must produce an ordered list of guesses. While "generation" is a core function of LLMs, this process does not enumerate strings (passwords) from the most to least probable, a key requirement for conducting a guessing attack. That said, generating (weighted random) passwords is less computationally intensive than probability assignment. A clever attacker might use an LLM to generate passwords underrepresented in datasets and a cheaper model to order guesses. Even so, this approach is far more expensive than existing rule-mangling approaches [27].

Pruning: Although we developed our pruning methods for rating the strength of long passwords, these algorithms work for any string. We imagine these pruning methods could be broadly useful for LLM safety or security audits. For example, one could use a pruning strategy to assess gender bias by checking the probability of typically male names being produced, as opposed to using repeated sampling or per-token probability lookups. However, pruning incurs error. Deeper analyses of this error and its bounds—potentially calculated with the Confident Monte-Carlo approach [19]—are exciting avenues for future work.

LLM-Driven Feedback: We also found that LLMs were able to provide semantically rich password feedback that often looked appropriate in isolation. However, recurring patterns in this feedback, such as frequently suggesting that "blue" be added, would be predictable to a clever attacker. Some of this poor quality feedback may be due to a tool mismatch: LLMs are trained to predict likely text, while modifications should be suggested randomly. We speculate that giving a tool-calling LLM access to a pseudorandom number generator may provide the best of both worlds: rich, individualized feedback *and* strong passwords. That said, our analyses all centered on prompts we created. Future work could collect and investigate the analogous prompts a range of end users choose, the effect of prompt length, and the impact of tool-calling to obtain better randomness.

Limitations: There is no "ground truth" for password strength. While PGS++ is an ensemble of prior models, this is an imperfect proxy for a password's actual strength. There is also error associated with guess numbers due to pruning and Monte Carlo estimation. While pruning error for our best strategy seemed small (Section 5.1.2), we cannot guarantee negligible error for all possible passwords.

While we took care to ensure that LLMs were tested on passwords that were never publicly released, some may still exist in LLMs' pretraining (e.g., dictionary words). Further, it is likely that all pretrained LLMs were exposed to the Linkedin breach due to its wide circulation (see Section 3.2). Finally, a key observed strength of LLMs was their ability to recognize cultural references. However, LLMs' knowledge cutoffs may prevent them from understanding emerging pop-culture references. We leave an investigation of how to best handle knowledge cutoffs to future work.

Our experiments require giving one's plaintext password to an LLM. If an LLM is running locally, this may be secure. However, these information flows require careful thought, especially if a proprietary model is used. Regardless of whether it is secure to do so, end users may provide their plaintext passwords to LLMs for feedback as part of their general reliance on LLMs. We caution enterprise security engineers to consider this avenue of credential leakage.

Finally, our fine-tuning experiments showed that LLMs struggled to model the strength of passwords made under complex composition policies. We speculate that could be due to LLMs' training data (natural language documents) and/or the inherent architectural structure of LLMs. LLM tokenizers and attention mechanisms were optimized to model natural language. They may inherently struggle to model the more unusual transformations that emerge under complex password-composition policies. If this is indeed the case, more training data alone may not suffice.

Reproducibility: To facilitate the reproduction of our results, we release our code for LLM fine-tuning and pruning, our fine-tuned models, and all prompting data collected for public passwords in the following Open Science Framework repository: https://osf.io/mzqkb/

While the public release includes all data we collected for the two sets of passwords that have previously been released publicly, it cannot include the passwords originally collected by CMU researchers; they never requested permission from their human subjects or IRB for public release.

Acknowledgments

Some results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation [86]. We thank David Cash and Ari Holtzman for their feedback on our manuscript and technical guidance, as well as Maximillian Golla and Alexandra Nisenoff for their insights and help sourcing passwords. Finally, we thank our anonymous reviewers for their helpful suggestions.

References

[1] D. Florêncio, C. Herley, and P. C. Van Oorschot, "Pushing On String: The 'Don't Care' Region of Password Strength," *CACM*, vol. 59, no. 11, 2016.

[2] ——, "An Administrator's Guide to Internet Password Research," in *Proc. LISA*, 2014.

[3] M. Golla and M. Dürmuth, "On the Accuracy of Password Strength Meters," in *Proc. CCS*, 2018.

[4] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, "Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks," in *Proc. USENIX Security*, 2016.

[5] A. Narayanan and V. Shmatikov, "Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff," in *Proc. CCS*, 2005.

[6] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek, "Password Cracking Using Probabilistic Context-Free Grammars," in *Proc. IEEE S&P*, 2009.

[7] D. L. Wheeler, "zxcvbn: Low-Budget Password Strength Estimation," in *Proc. USENIX Security*, 2016.

[8] T. Hunt, C. Hunt, and S. J. Sigurdarson, "Have I Been Pwned?" https://haveibeenpwned.com, 2024.

[9] J. Ma, W. Yang, M. Luo, and N. Li, "A Study of Probabilistic Password Models," in *Proc. IEEE S&P*, 2014.

[10] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, "Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms," in *Proc. IEEE S&P*, 2012.

[11] M. Islam, M. S. Bohuk, P. Chung, T. Ristenpart, and R. Chatterjee, "Araña: Discovering and Characterizing Password Guessing Attacks in Practice," in *Proc. USENIX Security*, 2023.

[12] J. Bonneau, "The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords," in *Proc. IEEE S&P*, 2012.

[13] J. Bonneau, M. Just, and G. Matthews, ""What's in a Name?"," in *Proc. FC*, 2010.

[14] D. Pasquini, M. Cianfriglia, G. Ateniese, and M. Bernaschi, "Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries," in *Proc. USENIX Security*, 2021.

[15] D. Goodin, "Why Passwords Have Never Been Weaker–and Crackers Have Never Been Stronger," https://arstechnica.com/information-technology/2012/08/passwords-under-assault/, 2012.

[16] J. Blocki, B. Harsha, and S. Zhou, "On the Economics of Offline Password Cracking," in *Proc. IEEE S&P*, 2018.

[17] M. Dell'Amico and M. Filippone, "Monte Carlo Strength Evaluation: Fast and Reliable Password Checking," in *Proc. CCS*, 2015.

[18] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring Real-World Accuracies and Biases in Modeling Password Guessability," in *Proc. USENIX Security*, 2015.

[19] P. Liu, J. Blocki, and W. Bai, "Confident Monte Carlo: Rigorous Analysis of Guessing Curves for Probabilistic Password Models," in *Proc. IEEE S&P*, 2023.

[20] J. Tan, L. Bauer, N. Christin, and L. F. Cranor, "Practical Recommendations for Stronger, More Usable Passwords Combining Minimum-strength, Minimum-length, and Blocklist Requirements," in *Proc. CCS*, 2020.

[21] A. Nisenoff, M. Golla, M. Wei, J. Hainline, H. Szymanek, A. Braun, A. Hildebrandt, B. Christensen, D. Langenberg, and B. Ur, "A Two-Decade Retrospective Analysis of a University's Vulnerability to Attacks Exploiting Reused Passwords," in *Proc. USENIX Security*, 2023.

[22] B. Pal, M. Islam, M. S. Bohuk, N. Sullivan, L. Valenta, T. Whalen, C. Wood, T. Ristenpart, and R. Chatterjee, "Might I Get Pwned: A Second Generation Compromised Credential Checking Service," in *Proc. USENIX Security*, 2022.

[23] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond Credential Stuffing: Password Similarity Models Using Neural Networks," in *Proc. IEEE S&P*, 2019.

[24] S. Houshmand and S. Aggarwal, "Using Personal Information in Targeted Grammar-based Probabilistic Password Attacks," in *Proc. ADF*, 2017.

[25] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted Online Password Guessing: An Underestimated Threat," in *Proc. CCS*, 2016.

[26] Hashcat, "Advanced Password Recovery," https://hashcat.net/, 2026.

[27] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur, "Reasoning Analytically about Password-Cracking Software," in *Proc. IEEE S&P*, 2019.

[28] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "PassGAN: A Deep Learning Approach for Password Guessing," in *Proc. ACNS*, 2019.

[29] D. Pasquini, G. Ateniese, and C. Troncoso, "Universal Neural-Cracking-Machines: Self-Configurable Password Models from Auxiliary Data," in *Proc. IEEE S&P*, 2024.

[30] B. Dorsey, "PassGAN," https://github.com/brannondorsey/PassGAN, 2018.

[31] X. He, H. Cheng, J. Xie, P. Wang, and K. Liang, "Passtrans: An Improved Password Reuse Model Based on Transformer," in *Proc. ICASSP*, 2022.

[32] D. Biesner, K. Cvejoski, and R. Sifa, "Combining Variational Autoencoders and Transformer Language Models for Improved Password Generation," in *Proc. ARES*, 2022.

[33] J. Rando, F. Perez-Cruz, and B. Hitaj, "PassGPT: Password Modeling and (Guided) Generation with Large Language Models," in *Proc. ESORICS*, 2023.

[34] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han, "Improving Real-world Password Guessing Attacks via Bi-directional Transformers," in *Proc. USENIX Security*, 2023.

[35] J. Yang, W. Li, H. Cheng, and P. Wang, "Username-Password Models Beyond Traditional Password Guessability Assessment," in *Proc. ICASSP*, 2025.

[36] M. Atzori, E. Calò, L. Caruccio, S. Cirillo, G. Polese, and G. Solimando, "Evaluating Password Strength Based on Information Spread on Social Networks: A Combined Approach Relying on Data Reconstruction and Generative Models," *Proc. OSNEM*, 2024.

[37] X. Su, X. Zhu, Y. Li, Y. Li, C. Chen, and P. Esteves-Veríssimo, "PagPassGPT: Pattern Guided Password Guessing via Generative Pretrained Transformer," in *Proc. DSN*, 2024.

[38] Y. Zou, M. An, and D. Wang, "Password Guessing Using Large Language Models," in *Proc. USENIX Security*, 2025.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," arXiv:1706.03762, 2017.

[40] Meta, "Llama 4," https://www.llama.com/models/llama-4/, 2025.

[41] N. Lukas, A. Salem, R. Sim, S. Tople, L. Wutschitz, and S. Zanella-Béguelin, "Analyzing Leakage of Personally Identifiable Information in Language Models," in *Proc. IEEE S&P*, 2023.

[42] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting Training Data from Large Language Models," in *Proc. USENIX Security*, 2021.

[43] N. Carlini *et al.*, "Stealing Part of a Production Language Model," in *Proc. ICML*, 2024.

[44] Y. Wen, L. Marchyok, S. Hong, J. Geiping, T. Goldstein, and N. Carlini, "Privacy Backdoors: Enhancing Membership Inference Through Poisoning Pre-trained Models," in *Proc. NeurIPS*, 2024.

[45] T. B. Brown *et al.*, "Language Models Are Few-shot Learners," arXiv:2005.14165, 2020.

[46] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv:2307.09288, 2023.

[47] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," in *Proc. ICLR*, 2020.

[48] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models Are Zero-shot Reasoners," in *Proc. NIPS*, 2022.

[49] R. Veras, C. Collins, and J. Thorpe, "A Large-Scale Analysis of the Semantic Password Model and Linguistic Patterns in Passwords," *ACM TOPS*, vol. 24, no. 3, 2021.

[50] M. Wei, M. Golla, and B. Ur, "The Password Doesn't Fall Far: How Service Influences Password Choice," in *Proc. WAY*, 2018.

[51] M. Burnett, "Today I Am Releasing Ten Million Passwords," https://medium.com/xato-security/today-i-am-releasing-ten-million-passwords-b6278bbe7495, 2015.

[52] K. Lee, S. Sjöberg, and A. Narayanan, "Password Policies of Most Top Websites Fail to Follow Best Practices," in *Proc. SOUPS*, 2022.

[53] X. de Carné de Carnavalet and M. Mannan, "From Very Weak to Very Strong: Analyzing Password-Strength Meters," in *Proc. NDSS*, 2014.

[54] N. Kandpal, E. Wallace, and C. Raffel, "Deduplicating Training Data Mitigates Privacy Risks in Language Models," in *Proc. ICLR*, 2022.

[55] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, "Deduplicating Training Data Makes Language Models Better," in *Proc. ACL*, 2022.

[56] B. Ur, F. Alfieri, M. Aung, L. Bauer, N. Christin, J. Colnago, L. F. Cranor, H. Dixon, P. Emami Naeini, H. Habib, N. Johnson, and W. Melicher, "Design and Evaluation of a Data-Driven Password Meter," in *Proc. CHI*, 2017.

[57] B. Ur, J. Bees, S. M. Segreti, L. Bauer, N. Christin, and L. F. Cranor, "Do Users' Perceptions of Password Security Match Reality?" in *Proc. CHI*, 2016.

[58] Twingate Team, "What Happened in the Bookcrossing Data Breach?" https://www.twingate.com/blog/tips/bookcrossing-data-breach, 2024.

[59] BookCrossing, "Pre 2013 Data Breach," https://www.bookcrossing.com/forum/9/584194, 2023.

[60] T. Hunt, "Observations and Thoughts on the LinkedIn Data Breach," https://www.troyhunt.com/observations-and-thoughts-on-the-linkedin-data-breach/, 2016.

[61] A. Addington, "Knowledge Cutoff Dates For ChatGPT, Meta AI, Copilot, Gemini, Claude," https://computercity.com/artificial-intelligence/knowledge-cutoff-dates-llms, 2024.

[62] Meta, "Llama 2 Model Card," https://github.com/meta-llama/llama/blob/main/MODEL_CARD.md, 2023.

[63] Anthropic, "Release Notes," https://support.anthropic.com/en/articles/8114494-how-up-to-date-is-claude-s-training-data, 2025.

[64] OpenAI, "Introducing GPT-4.1 in the API," https://openai.com/index/gpt-4-1/, 2025.

[65] Oracle, "Meta Llama 4 Maverick (New)," https://docs.oracle.com/en-us/iaas/Content/generative-ai/meta-llama-4-maverick.htm, 2025.

[66] Google Cloud, "Gemini 2.5 Pro," https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro, 2025.

[67] Carnegie Mellon University, "Password Guessability Service," https://pgs.ece.cmu.edu/, 2015.

[68] JohnTheRipper, "John The Ripper Password Cracker," https://www.openwall.com/john/, 2025.

[69] D. Hamilton, "Yahoo's Password Leak: What You Need to Know (FAQ)," https://www.cnet.com/news/privacy/yahoos-password-leak-what-you-need-to-know-faq/, 2006.

[70] J. Blocki and A. Datta, "CASH: A Cost Asymmetric Secure Hash Algorithm for Optimal Password Protection," in *Proc. CSF*, 2016.

[71] W. Bai and J. Blocki, "DAHash: Distribution Aware Tuning of Password Hashing Costs," in *Proc. FC*, 2021.

[72] K. Scarfone and M. Souppaya, "Guide to Enterprise Password Management," https://csrc.nist.gov/pubs/sp/800/118/ipd, 2009.

[73] T. Le Scao and A. Rush, "How Many Data Points Is a Prompt Worth?" in *Proc. ACL*, 2021.

[74] A. Webson and E. Pavlick, "Do Prompt-Based Models Really Understand the Meaning of Their Prompts?" in *Proc. ACL*, 2022.

[75] A. Vance, "If Your Password Is 123456, Just Make It Hackme," https://www.nytimes.com/2010/01/21/technology/21password.html, 2010.

[76] T. Hunt, "Breaches, Traders, Plain Text Passwords, Ethical Disclosure and 000webhost," https://www.troyhunt.com/breaches-traders-plain-text-passwords/, 2015.

[77] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling Laws for Neural Language Models," arXiv:2001.08361, 2020.

[78] Y. Razeghi, R. L. Logan IV, M. Gardner, and S. Singh, "Impact of Pretraining Term Frequencies on Few-shot Numerical Reasoning," in *Proc. EMNLP*, 2022.

[79] A. Radford and K. Narasimhan, "Improving Language Understanding by Generative Pre-Training," https://api.semanticscholar.org/CorpusID:49313245, 2018.

[80] N. Muennighoff, A. M. Rush, B. Barak, T. L. Scao, A. Piktus, N. Tazi, S. Pyysalo, T. Wolf, and C. Raffel, "Scaling Data-Constrained Language Models," *JMLR*, vol. 26, no. 53, 2025.

[81] D. Yogatama, C. de Masson d'Autume, J. Connor, T. Kocisky, M. Chrzanowski, L. Kong, A. Lazaridou, W. Ling, L. Yu, C. Dyer, and P. Blunsom, "Learning and Evaluating General Linguistic Intelligence," arXiv:1901.11373, 2019.

[82] H. W. Chung *et al.*, "Scaling Instruction-Finetuned Language Models," *JMLR*, vol. 25, no. 1, 2024.

[83] T. Gao, A. Fisch, and D. Chen, "Making Pre-trained Language Models Better Few-shot Learners," in *Proc. ACL*, 2021.

[84] N. Carlini, D. Ippolito, M. Jagielski, K. Lee, F. Tramer, and C. Zhang, "Quantifying Memorization Across Neural Language Models," in *Proc. ICLR*, 2023.

[85] R. Munroe, "xkcd: Password Strength," https://xkcd.com/936/?correct=horse&battery=staple, 2025.

[86] K. Keahey *et al.*, "Lessons Learned from the Chameleon Testbed," in *Proc. USENIX ATC*, 2020.

[87] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the Carbon Emissions of Machine Learning," *arXiv:1910.09700*, 2019.

[88] Hugging Face, "Padding and Truncation," https://huggingface.co/docs/transformers/pad_truncation, 2025.

Ethics Considerations

Our use of evaluation sets containing real passwords that were originally stolen in data breaches and subsequently leaked publicly raises important ethical considerations. While the data subjects did not consent to having their data released publicly, these sets are already publicly available. In our experiments, we intentionally did not use, and in fact deleted, all data other than the passwords themselves. Notably, we deleted the usernames and email addresses associated with these credentials, minimizing the likelihood that these passwords represent identifiable information. We expect the benefits of our research (contributing to better understanding how to use LLMs to model password strength defensively) outweigh the unlikely possibility that our use of these evaluation sets would cause additional harm.

We took many additional precautions in terms of submitting data to LLMs. Namely, we checked each model provider's policies and terms of service for data-retention policies and future data use regarding LLM training. We ensured that, according to these stated policies based on the (paid) APIs we used, none of our experiments should result in the data being retained by the model developers.

Additionally, we used four sets of passwords previously collected by different investigators for a different research study among our evaluation sets. We worked with those investigators to ensure that only de-identified data would be transferred to us and that no metadata (e.g., pseudonyms, survey responses) would be included. We submitted a formal protocol to our IRB for secondary analysis of that data; our institution's IRB reached a formal determination that our secondary analysis was not considered human-subjects research. Regardless, while fulfilling our commitment to scientific reproducibility in releasing the data we collected for the Bookcrossing and Linkedin datasets, both of which are already available publicly, we do not do the same for these passwords collected in past studies since they have never been publicly released.

While most prior work on applying LLMs to passwords (see Section 2.2) is primarily offensive in nature (e.g., proposing new guessing attacks that use LLMs), our work is defensive in nature. Specifically, we aim to use LLMs to provide accurate password-strength estimates. Notably, our work does not propose any methods for efficiently generating probability-ordered lists of guesses from even our fine-tuned models. In fact, doing so is very likely to be computationally challenging.

A final ethical consideration relates to the environmental impacts of LLMs. Compared to the legacy password models represented in PGS++, LLMs have a much larger environmental impact for either fine-tuning or querying LLMs (locally or via API prompts). In other words, LLMs are much less computationally efficient mechanism than legacy password models. We expect that the benefit of using LLMs instead will come from their potential to adapt to previously unseen password distributions in the future (see Section 8).

LLM Usage Considerations

While LLMs were the object of study of our work, we did not use LLMs to assist in the preparation of our paper. All writing and copyediting was performed by humans.

We chose LLMs as our main topic of study as we hypothesized that the unique characteristics of natural language learned during pretraining could overcome longstanding deficiencies in defensive techniques for modeling password strength. Our work involved a series of experiments on different LLMs, as well as on the same LLMs in differing configurations, designed to establish best practices for using LLMs to model password strength and provide password feedback. The goal of these experiments was to establish how other researchers and practitioners ought to consider using LLMs in this domain, meaning that others would hopefully not need to repeat these experiments. We hope that our experiments and transparent reporting (including our data release) obviate the need for others to perform their own experiments, thus saving future computation.

Password data can be sensitive and have privacy implications for the humans who created those passwords. As such, we carefully chose LLMs (and their accompanying access methods) to minimize these privacy implications. As much as possible, we used open-weight models (e.g., GPT-2, Llama 2) that we ran ourselves locally, preventing companies from knowing anything about our queries. For closed-weight LLMs, we carefully reviewed data-usage agreements to ensure that none of our queries would be used for companies' training purposes. We paid for those companies' premium APIs, which promise not to use the data provided for model training accordingly.

We carefully scaled down experiments to minimize environmental impacts while preserving scientific rigor. For instance, we downsampled password sets to 500 passwords to prevent an overly large number of queries from being issued. Similarly, we used the minimum number of queries necessary to prompt the LLMs (1 query/prompt) while controlling for potential confounds. While our full-factorial design ultimately does result in more prompt combinations being tried, we believe this is scientifically necessary to control for confounds appropriately.

We estimate [87] that approximately 72.58 kg of carbon was emitted due to the experiments presented. This amount of emissions is substantially less than the emissions from one of our research team members flying to the conference. However, many of our ablation experiments highlight the success of using smaller models and less training. These findings may help prevent unnecessary carbon emissions from other researchers moving forward.

At every point we used LLMs for data parsing, we manually validated the LLM's performance. While some results obtained from prompted LLMs may not be reproducible due to model deprecation, we partially mitigate this concern by also studying open-source LLMs and by releasing the data collected from our experiments when possible.

Appendix A.
Additional Details for Prompting Methods

TABLE 12: Prompt variants for soliciting abstract password suggestions. Bold text highlights changes across prompts; the actual prompts did not include bold text.

Variant	Describe How
Base	In three bullet points, describe how the password on the next line could be modified to make it **stronger**. If the password is already **strong**, reply N/A.
End-User	In three bullet points, describe how the password on the next line could be modified to make it **better**. If the password is already **good**, reply N/A.
Memorable	In three bullet points, describe how the password on the next line could be modified to make it stronger **while keeping it memorable**. If the password is already strong, reply N/A.
Diversity	In three bullet points, describe how the password on the next line could be modified to make it stronger, **using a different approach for each modification**. If the password is already strong, reply N/A.
Explain	In three bullet points, **explain** how the password on the next line could be modified to make it stronger. If the password is already strong, reply N/A.

TABLE 13: Prompt variants for LLM direct improvements.

Variant	Directly Improve
Base	Give three examples of **stronger** versions of the password on the next line, where each example illustrates a way to make the password **stronger**. If the password is already **strong**, reply N/A.
End-User	Give three examples of **better** versions of the password on the next line, where each example illustrates a way to make the password **better**. If the password is already **good**, reply N/A.
Memorable	Give three examples of stronger **but still memorable** versions of the password on the next line, where each example illustrates a way to make the password stronger. If the password is already strong, reply N/A.
Diversity	Give three examples of stronger versions of the password on the next line, where each example illustrates a **different** way to make the password stronger. If the password is already strong, reply N/A.
Explain	Give three examples of stronger versions of the password on the next line **and explain why each example is stronger**, where each example illustrates a way to make the password stronger. If the password is already strong, reply N/A.

A.1. Prompt Fine-Tuning Tokens

Our natural language prompt was: "[BOS]A password is: <password>[EOS]," where [BOS] and [EOS] denote each model's beginning and end of sentence tokens, respectively.

For the randomly chosen prompt, we randomly and uniformly sample five tokens from the LLMs' vocabulary, then prefix and suffix passwords with them such that they appear as "[random tokens]<password>[random tokens]." The tokens for GPT-2's random prompt were: [43326, 45260, 41877, 24763, 6564]. The tokens for Llama 2's random prompt were: [2200, 18372, 31624, 30889, 20752].

A.2. Explicit CoT Prompt Heuristics

A password is stronger the more it is a truly random mixture of letters, numbers, and symbols. However, humans often make passwords in predicable ways, using certain patterns to do so. Some of the common patterns humans use to make passwords are as follows:
- tokens, for example: "logitech", "l0giT3CH", "ain't", "parliamentarian", "1232323q" [. . .]
- reversing, for example: "DrowssaP"
- sequences, for example: "123", "23456", "jklm", "ywsuq"
- repeating, for example: "zzz", "ababab", "l0giT3CHl0giT3CH"
- keyboard patterns, for example: "qwertyuio", "qAzxcde3", "diueoa"
- dates, for example: "7/8/1947", "8.7.47", "781947", "4778", "7-21-2011", "72111", "11.7.21"

Appendix B.
Additional Technical Details

B.1. Training and Generation

For all models, we set the per-device training batch size to 64 and the gradient accumulation steps to 16. We do not quantize or use other PEFT approaches. We use one NVIDIA RTX 6000 for GPT-2 LLMs and one or more NVIDIA A100s for Llama LLMs. We use a single-node setup for all experiments. We pad input sequences to a multiple of 256 during training so inputs can be batched [88]. However, this also expands the final output layer of the model ("logits"). Occasionally, models assigned non-zero probability to the expanded logits even though they do not map to a token in the pretrained model's vocabulary. We discarded these tokens. For more details, please see our OSF.

B.2. XATO Properties

XATO contains 10 million passwords, less than 1% of which fit Complex, LongBasic, or LongComplex password-composition policies. Among XATO passwords, 48% are Basic. Our de-duplicated sets are 39% (100k) and 33% (10k) Basic. Of the XATO passwords, 50% are short (containing $<$8 characters). Where "**xcy**" denotes a password of length $\geq$y+ with x different character classes, 23% are 1c8, 21% are 2c8, 5% are 3c8, and the remaining $<$1% are 4c8, 1c16, or 3c12. Our 10k and 100k sets do not differ meaningfully from this distribution. Our 100k de-duplicated set contains 61% short passwords, 32% 1c8, 7% 2c8, and less than 1% for the rest. Our 10k de-duplicated set contains 67% short passwords, 30% 1c8, 3% 2c8, and less than 1% for the rest.

B.3. Data Extraction

LLMs sometimes ignored instructions. We prompted LLMs, "*Using your previous response, output only the rating as a number (e.g., 'one' should be 1) and no other text.*" We replaced "rating" with "guess number" as appropriate. Models sometimes output scientific notation, which we parsed with regular expressions. Despite our efforts, we failed to parse (and discarded) some of each LLM's 42,000 outputs: 354 (Llama 4), 0 (Gemini 2.5 Pro), 6 (GPT 4.1), and 297 (Claude Opus 4). For specific feedback, we used Mistral:7b to parse examples and modifications. We manually verified 720 responses; it correctly identified 96.2%.

Appendix C.
Additional Tables and Figures

TABLE 14: Codebook for *agreements* between strength estimates produced by an LLM and PGS++.

Code	Description
Random	The password looks randomly generated
Strong	Password is clearly strong, but not random
Weak	Password is clearly weak; may be a known common password
Other	None of the above apply

TABLE 15: Codebook for *disagreements* between strength estimates produced by an LLM and PGS++.

Code	Description
Culture	Includes cultural phenomenon (e.g., location, phrase)
Pop Culture	Includes a pop culture reference (e.g., sports, music)
Platform	Includes a platform far less established in 2010
Western Name	Includes a name common in the US (e.g., John)
Non-Western Name	Includes a non-English or non-Western name
Email	Includes an email address
Foreign Language	Password is in a language other than English
Phrase	Includes a (potentially uncommon) phrase
Long	The password is very long
Leet	The password contains leet substitutions (e.g., 1–>!)
Random	The password looks randomly generated
Other	None of the above apply

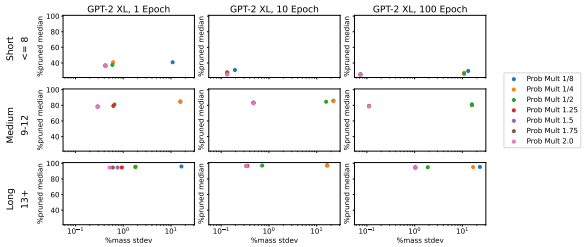

Figure 8: Canonical Probability Pruning (GPT-2 XL).

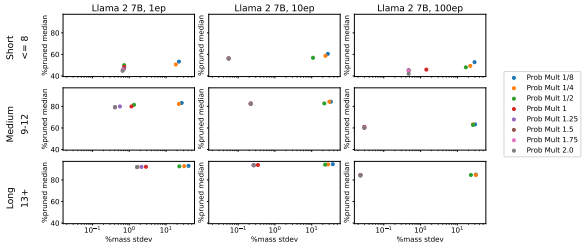

Figure 9: Canonical Probability Pruning (Llama 2 7B).

TABLE 16: The performance of each pruning strategy for GPT-2 XL on long passwords (13+ characters in length).

Approach	Epochs	Median % Pruned	σ in % Mass
Canon Prob (2.0)	1	94.62	0.52
	10	96.53	0.34
	100	94.53	1.04
Canon Len (1.25)	1	63.53	0.79
	10	63.53	0.03
	100	63.53	0.02
Thresh (1e-35)	1	25.99	0.03
	10	75.65	0.01
	100	85.92	16.65
Float Error	1	9.44	0.00
	10	25.44	0.00
	100	23.28	16.66
Depth (10)	1	15.15	1.67
	10	15.15	0.15
	100	15.15	2.18

TABLE 17: The performance of each pruning strategy for Llama 2 7B on long passwords (13+ characters in length).

Approach	Epochs	Median % Pruned	σ in % Mass
Canon Prob (2.0)	1	91.92	1.64
	10	93.63	0.26
	100	84.13	0.02
Thresh (1e-35)	1	48.27	0.00
	10	75.66	0.00
	100	85.55	44.97
Float Error	1	13.14	0.00
	10	47.05	0.00
	100	72.30	34.98
Canon Len (1.25)	1	27.91	0.01
	10	27.91	0.00
	100	27.91	2.97
Depth (15)	1	0.01	16.66
	10	0.01	16.66
	100	0.01	16.66

TABLE 18: Spearman's Rho correlation coefficient between our GPT-2 LLMs and PGS++.

Evaluation Set	Baseline	One Epoch	Five Epochs	No Pretraining	Less Data	Non-Deduplicated	Random Prompt	Smaller Model
Linkedin	0.84	0.93	0.87	0.78	0.92	0.88	0.85	0.90
Bookcrossing	0.96	0.97	0.96	0.87	0.97	0.83	0.96	0.95
CMU-Basic	0.89	0.91	0.91	0.82	0.90	0.87	0.89	0.87
CMU-LongBasic	0.73	0.79	0.77	0.57	0.73	0.82	0.77	0.79
CMU-Complex	0.53	0.64	0.60	0.62	0.46	0.55	0.60	0.54
CMU-LongComplex	0.46	0.64	0.53	0.49	0.47	0.58	0.54	0.66

TABLE 19: Spearman's Rho correlation coefficient between our Llama 2 LLMs and PGS++.

Evaluation Set	Baseline	One Epoch	Five Epochs	No Pretraining	Less Data	Non-Deduplicated	Random Prompt
Linkedin	0.49	0.75	0.68	0.97	0.77	0.81	0.86
Bookcrossing	0.95	0.98	0.97	0.97	0.97	0.57	0.95
CMU-Basic	0.90	0.90	0.91	0.85	0.89	0.90	0.91
CMU-LongBasic	0.73	0.80	0.74	0.47	0.80	0.77	0.76
CMU-Complex	0.51	0.59	0.56	0.23	0.46	0.53	0.55
CMU-LongComplex	0.42	0.61	0.54	0.12	0.57	0.61	0.51

Appendix D.
Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

D.1. Summary

This paper investigates the effectiveness of Large Language Models (LLMs) in password security, specifically focusing on their ability to estimate password strength and provide feedback to improve password security. The authors evaluate direct prompting of state-of-the-art models and a fine-tuning approach for smaller models, demonstrating that LLMs can identify semantically weak passwords (e.g., cultural references) that traditional probabilistic models often miss.

D.2. Scientific Contributions

- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

D.3. Reasons for Acceptance

1) Creates a new tool to enable future science: The authors have open-sourced their fine-tuning code and released prompting data.
2) Provides a valuable step forward in an established field: The paper contributes novel insights by showing how LLMs can augment existing probabilistic password cracking models, particularly by identifying weak passwords containing names, quotes, or cultural references.

D.4. Noteworthy Concerns

1) The paper evaluates LLM-based password strength estimation via correlation with PGS++, which is a reasonable and practical approach in the absence of ground truth. Accordingly, correlation with PGS++ should not be interpreted as a definitive measure of accuracy. The paper appropriately notes these limitations.
2) The pruning strategies used to speed up calculations could introduce errors in the guessing number estimates. The paper presents promising empirical evidence that optimal pruning techniques do not significantly impact the accuracy of password guessing number or guessing curve estimates in the evaluated settings. However, a more extensive evaluation would be needed to conclusively establish accuracy across a broader range of settings.