Measuring memorization in language models via probabilistic extraction

Jamie Hayes[*] Google DeepMind
Marika Swanberg[*] Google
Harsh Chaudhari Google DeepMind & Northeastern
Itay Yona Google DeepMind
Ilia Shumailov Google DeepMind
Milad Nasr Google DeepMind
Christopher A. Choquette-Choo Google DeepMind
Katherine Lee Google DeepMind
A. Feder Cooper[*] Microsoft Research & Stanford

Abstract

Large language models (LLMs) are susceptible to memorizing training data, raising concerns about the potential extraction of sensitive information at generation time. Discoverable extraction is the most common method for measuring this issue: split a training example into a prefix and suffix, then prompt the LLM with the prefix, and deem the example extractable if the LLM generates the matching suffix using greedy sampling. This definition yields a yes-or-no determination of whether extraction was successful with respect to a *single* query. Though efficient to compute, we show that this definition is unreliable because it does not account for non-determinism present in more realistic (non-greedy) sampling schemes, for which LLMs produce a range of outputs for the same prompt. We introduce *probabilistic* discoverable extraction, which, without additional cost, relaxes discoverable extraction by considering *multiple* queries to quantify the probability of extracting a target sequence. We evaluate our probabilistic measure across different models, sampling schemes, and training-data repetitions, and find that this measure provides more nuanced information about extraction risk compared to traditional discoverable extraction.

1 Introduction

Large language models (LLMs) are susceptible to memorizing pieces of their training data, raising concerns about the potential extraction of sensitive information in the training dataset at generation time (Carlini et al., 2021, 2022; Shi et al., 2023; Zhang et al., 2023; Smith et al., 2023; Lee et al., 2023; Biderman et al., 2024; Duan et al., 2024; Cooper and Grimmelmann, 2024; Huang et al., 2024; Bordt et al., 2024).[1] This issue has attracted significant attention, leading researchers to routinely report training-data extraction rates in technical reports introducing new LLMs (Chowdhery et al., 2023; Anil et al., 2023; Gemini Team et al., 2024; Gemma Team et al., 2024a,b; Biderman et al., 2023; Llama Team, 2024). There are numerous ways to measure these rates, but one of the most common is to quantify **discoverable extraction**: split an LLM's training example into a prefix and suffix, prompt the LLM with the prefix, and deem the example extractable if the LLM generates the sequence that matches the suffix (Carlini et al., 2021, 2022; Nasr et al., 2023) (Section 2).

Discoverable extraction is simple and efficient to compute; however, it has drawbacks that may make it an unreliable estimate for a model's true extraction rate. Notably, the definition for discoverable extraction does not account for the non-deterministic nature of LLMs. It yields a yes-or-no determination of whether extraction was successful with respect to a *single* user query, most typically executed with deterministic greedy sampling. But, of course, in realistic production settings, users may query the model multiple times and with non-deterministic sampling schemes, where multiple queries with the same prompt can result in a range of different generations. So, by only generating a single sequence to check for a match with the target, discoverable extraction may miss cases where a match could have been found if more than one sequence had been generated. Further, extracting sensitive training data even once out of multiple queries could be problematic, as an adversary (e.g., a hacker checking credit card numbers) may have external means of verifying

[*]Corresponding author: jamhayes@google.com, marikaswanberg@google.com, afedercooper@gmail.com

[1]This paper covers a very restricted definition of "memorization": whether a generative model can be induced to generate near-facsimiles of some training examples when prompted with appropriate instructions. Models do not "contain" bitwise or code-wise copies of their training data. Rather, If a model can be induced to generate very close copies of certain training examples by supplying such instructions to guide the model's statistical generation process, then that model is said to have "memorized" those examples. This is an area of active ongoing research.

the sensitive information's validity. It is therefore reasonable to quantify the number of sequences that need to be generated before a particular target example becomes extractable.

Following from this motivation, we introduce *probabilistic* discoverable extraction: a relaxation of discoverable extraction that considers *multiple* queries in order to quantify the probability of extracting a particular target sequence. That is, our new definition for **(n, p)-discoverable extraction** quantifies the number of attempts n an adversary would need to make to extract a target sequence at least once with probability p under a given sampling scheme (Section 3). We benchmark (n, p)-discoverable extraction rates for different sampling schemes, settings of n and p, model families and sizes, and repetitions of target data (Section 4). In summary, (n, p)-discoverable extraction

- Provides more meaningful measurements of extraction rates than greedy-sampled discoverable extraction, which, by comparison, underestimates extraction rates even for modest values of n and p (Sections 3.1 & 4.1). The gap between the two rates increases for larger models and more repetitions of the target (Section 4.2).
- Captures the risk that a particular target can be extracted under a given non-deterministic sampling scheme (Sections 3.1 & 4.3).
- Can be approximated with high-fidelity using just one query—i.e., *with no overhead compared to discoverable extraction* (Section 3.2).
- Is easily extended to quantify extraction for non-verbatim matches to the target (Section 3.3).

2 Preliminaries and related work

We begin with prior work on discoverable extraction (Section 2.1) and relevant background on different sampling schemes (Section 2.2).

2.1 Discoverable extraction

There are many different definitions for extraction in the literature (Appendix B), but one of the most popular is **discoverable extraction** (Anil et al., 2023; Gemini Team et al., 2024; Gemma Team et al., 2024a,b; Kudugunta et al., 2024; Llama Team, 2024; Biderman et al., 2023; Kassem et al., 2024). We adapt existing definitions for discoverable extraction (Carlini et al., 2021, 2022; Nasr et al., 2023), described in terms of an arbitrary training example $\boldsymbol{z}$, model f_θ, and sampling scheme g_ϕ.

For a j-length sequence of tokens $\boldsymbol{z} = (z_1, \ldots, z_j)$ and indices $1 \leq h \leq i \leq j$, we use $\boldsymbol{z}_{h:i}$ to denote tokens $z_h, \ldots, z_i$. Let $f_\theta : \mathbb{V}^j \to \mathcal{P}(\mathbb{V})$ be a θ-parameterized LLM that takes a sequence of j tokens from a vocabulary $\mathbb{V}$ and outputs a probability distribution $\mathcal{P}(\mathbb{V})$ over $\mathbb{V}$. Let $g_\phi : \mathcal{P}(\mathbb{V}) \to \mathbb{V}$ be a sampling scheme parameterized by scheme-specific hyperparameters ϕ, which takes as arguments a probability distribution over vocabulary $\mathbb{V}$ and selects a token from $\mathbb{V}$. Finally, for some initial sequence $\boldsymbol{z}$, let $(g_\phi \circ f_\theta)^k(\boldsymbol{z})$ denote the autoregressive process of repeatedly generating a distribution over the token vocabulary, sampling a token from this distribution, and adding the token to the sequence $k > 0$ times, starting from the initial sequence $\boldsymbol{z}$. We will use **query** to denote this entire autoregressive process of generating multiple (up to size k) tokens.

Definition 2.1 (Discoverable extraction). Given a training example $\boldsymbol{z}$ that is split into an a-length prefix $\boldsymbol{z}_{1:a}$ and a k-length suffix $\boldsymbol{z}_{a+1:a+k}$, $\boldsymbol{z}$ is *discoverably extractable* if $(g_\phi \circ f_\theta)^k(\boldsymbol{z}_{1:a}) = \boldsymbol{z}_{1:a+k}$.

In other words, $\boldsymbol{z} = \boldsymbol{z}_{1:a} \parallel \boldsymbol{z}_{a+1:a+k}$; we use the first a tokens of a training example $\boldsymbol{z}$ as the input prompt to the generation process, and then check if the sequence generated under the composition of model f_θ and sampling scheme g_ϕ matches verbatim the remaining k tokens in the example. We refer to this as **one-shot extraction**, given that it returns a binary, yes-or-no determination for a single query. We will relax this when we revisit extraction from a probabilistic perspective in Section 3.

To measure discoverable extraction in practice, prior work has relied on different instantiations of Definition 2.1. This includes varying both the length of prefix prompts and the minimum-length generated suffix that qualifies as extraction. For example, Carlini et al. (2022) test prefix lengths with prompts ranging from 50 to 500 tokens, and consider a training example to be extracted if the model generates the subsequent 50 tokens in the example. Alternatively, Biderman et al. (2023) set the prefix and suffix size to 32 tokens.

Further, the sequence a model f_θ generates is entirely dependent on the choice of sampling scheme g_ϕ that defines how a token is selected from the output distribution over the model's vocabulary $\mathcal{P}(\mathbb{V})$. In most prior work, the common choice for g_ϕ is **greedy sampling**, which generates a sequence by selecting the highest-probability token, conditioned on the previous tokens, at each step. Focusing on

one-shot extraction is justified in this setting, given that the output is deterministic. Work by Carlini et al. (2022) is an exception; while they also focus on greedy sampling, they additionally analyze one-shot discoverable extraction with beam search.

2.2 Alternate sampling schemes

While greedy sampling is the most common choice for g_ϕ in prior work on discoverable extraction, it provides an incomplete picture. Many production language models are deployed with non-greedy schemes, and API users are often free to decide which scheme to use. Other sampling schemes, e.g., Fan et al. (2018); Basu et al. (2020); Vijayakumar et al. (2016), are desirable for increasing output diversity. Further, although greedy sampling locally selects the most likely *next token* it may not select the overall most likely *sequence*, which can decrease generation quality. Below, we briefly discuss popular choices for g_ϕ.

Random sampling with temperature. Given a sequence of $t-1$ tokens $\boldsymbol{z}_{1:t-1}$, we sample

$$\mathcal{P}(z_t \mid \boldsymbol{z}_{1:t-1}) = \frac{\exp\frac{y(z_t)}{T}}{\sum_{v\in\mathbb{V}} \exp\frac{y(v)}{T}}, \qquad (1)$$

where $y(v)$ denotes the logit value of token $v \in \mathbb{V}$, and $T \in \mathbb{R}_{>0}$ is a **temperature** value that controls the flatness (or sharpness) of the probability distribution $\mathcal{P}(\mathbb{V})$. $T = 1$ samples from the base distribution output by the model f_θ, larger T increases diversity, and, as $T \to 0$, Equation (1) converges to deterministic greedy sampling.

Top-k sampling. In $\mathcal{P}(z_t|\boldsymbol{z}_{1:t-1})$, all but the k tokens with the highest probabilities have their probabilities set to 0, the non-zero probabilities are normalized, and the next token is sampled accordingly. (If $k = 1$, this is identical to greedy sampling.)

Carlini et al. (2022) argue that using sampling schemes like these, which have higher degrees of associated randomness compared to deterministic greedy sampling, are "antithetical" to maximizing discoverable extraction; such schemes "maximi[ze] linguistic novelty" and so are less likely to generate outputs that match existing text. Nonetheless, these schemes may be advantageous for more reliably identifying extraction in realistic settings where, to explore different outputs, a user may query the model *multiple* times with the same prompt.

3 Probabilistic discoverable extraction

The more realistic setting of non-deterministic sampling and multiple queries motivates measuring extraction from a probabilistic perspective. Rather than taking a one-shot approach, it is reasonable to quantify the number of sequences that need to be generated before a target example becomes likely to be extracted under a chosen sampling scheme.

Following from this motivation, we introduce **(n, p)-discoverable extraction**, which captures the capabilities of a regular user that can query the model n times to extract verbatim a target sequence with probability p (Section 3.1). Under non-deterministic sampling schemes, many extractable targets are unlikely to be guaranteed to be generated (i.e., $p = 1$) with one query, but many targets may be generated at least once given enough attempts ($n > 1$). Thus, our definition captures a continuous notion of the risk of extracting the target example; it provides a more meaningful estimate of extraction risk than can be gleaned from one-shot, yes-or-no tests (Section 2.1).

We then discuss how n and p are directly related through a simple equation such that, in practice, it is possible to measure a high-fidelity approximation of (n, p)-discoverable extraction with just one query—*with no overhead compared to traditional discoverable extraction* (Section 3.2). We then briefly describe a variant of (n, p)-discoverable extraction that measures the non-verbatim extraction of target sequences (Section 3.3), and draw connections between our contributions and other work (Section 3.4).

3.1 Defining (n, p)-discoverable extraction

We define our probabilistic relaxation of discoverable extraction (Definition 2.1) with two new hyperparameters: number of queries n and probability p:

Definition 3.1 (**(n, p)-discoverable extraction**)**.** Given a training example $\boldsymbol{z}$ that is split into an a-length prefix $\boldsymbol{z}_{1:a}$ and a k-length suffix $\boldsymbol{z}_{a+1:a+k}$, $\boldsymbol{z}$ is (n, p)-*discoverably extractable* if

$$\Pr\left(\cup_{w\in[n]} (g_\phi \circ f_\theta)^k_w(\boldsymbol{z}_{1:a}) = \boldsymbol{z}_{1:a+k}\right) \geq p,$$

where $(g_\phi \circ f_\theta)^k_w(\boldsymbol{z}_{1:a})$ represents the w-th (of n) independent execution of the autoregressive process of generating a distribution over the token vocabulary, sampling a token from this distribution, and adding the token to the sequence $k > 0$ times, starting from the same initial sequence $\boldsymbol{z}_{1:a}$.

That is, in total, we generate n independent sequences by sampling the output distribution of model f_θ using scheme g_ϕ. If the probability of generating $\boldsymbol{z}_{a+1:a+k}$ *at least once* is larger than p,

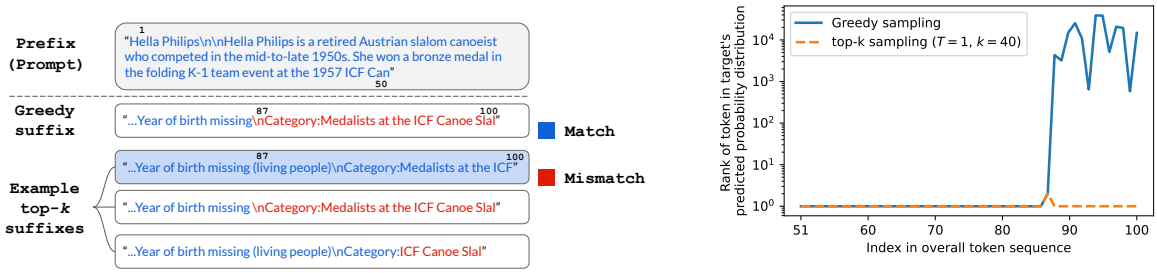


Figure 1: **Left:** The prefix $\boldsymbol{z}^t_{1:50}$, and portions of the greedy-sampled suffix and of example top-k-sampled suffixes for Pythia 12B. Blue indicates a match with the target, red a mismatch. **Right**: For each successive token that is decoded by greedy and top-k ($k = 40$, $T = 1$) sampling, we plot the probability rank with respect to the target suffix token. At index 87, the target token has rank 2; greedy sampling does not select this token, after which the greedy-generated sequence diverges from the target. In contrast, top-k sampling picks the rank-2 token and proceeds to extract the target sequence correctly (with probability 16.2%). Note that, if greedy sampling had selected the rank-2 token at index 87, then it would have generated the target, as the remaining target tokens all have rank-1.

then we say $\boldsymbol{z}$ is (n, p)-discoverably extractable with respect to f_θ and g_ϕ. When the sampling scheme and model are clear from context, we simply say the target is (n, p)-discoverably extractable. One can also view (n, p)-discoverable extraction as the probability that, together, f_θ and g_ϕ provide an n-sized anonymity set: a set of n data points within which the true training example can hide. Note that we can write discoverable extraction in terms of this probabilistic relaxation by setting $n = p = 1$. Also note that n and p are directly related. We will revisit this observation in relation to choosing n and p in practice (Section 3.2).

Next, we describe how (n, p)-discoverable extraction offers several advantages over standard discoverable extraction instantiated with greedy sampling (Definition 2.1). Through the example in Figure 1, we show how greedy sampling can underestimate extraction. As a result, research and model-release reports that rely on greedy-sampled discoverable extraction (e.g., Gemini Team et al., 2024; Llama Team, 2024) could substantially differ from the amount of extraction seen by an end user. We then briefly discuss how multiple queries can be used to capture extraction risk.

Greedy sampling misses instances of extraction. We illustrate this failure mode for greedy-sampled discoverable extraction for target prefix $\boldsymbol{z}^t_{1:50}$ with an example (Figure 1). The target suffix $\boldsymbol{z}^t_{51:100}$ has a higher overall likelihood of being generated, compared to the greedy-sampled suffix $\boldsymbol{z}^g_{51:100}$. In always selecting the locally-most-likely next token (Section 2.2), greedy sampling generates an output that does not match the overall-more-likely target;[2] greedy sampling fails to extract the target.

Figure 1 explores how this happens in more detail. For each token i in the target suffix $\boldsymbol{z}^t_{51:100}$, we compute the probabilities for each possible next token in the vocabulary (i.e., $\mathcal{P}(z^t_i | \boldsymbol{z}^t_{1:i-1})$) and we rank them, with rank 1 reflecting the highest-probability next token. For the first 36 suffix tokens (through overall index 86), the rank-1 token aligns with the actual target token. So, greedy sampling, which always locally selects the rank-1 token, selects these tokens (i.e, $\boldsymbol{z}^g_{51:86} = \boldsymbol{z}^t_{51:86}$). But, for the next token, the target token is the *second-highest probability*, rank-2 token, so greedy sampling does not select it (i.e., $z^g_{87} \neq z^t_{87}$). With this deviation, the rest of the greedy-sampled sequence continues to diverge from the target sequence; in each iteration, the locally highest-probability token differs (often significantly in rank) from the (overall-higher-probability) target token.

In contrast to greedy sampling, using a probabilistic sampling scheme allows for the possibility of selecting the rank-2 token at index 86. At this point, it becomes highly probable that the entire target suffix is extracted. We demonstrate this with top-k sampling ($T = 1$, $k = 40$), where the target has probability of 16.2% of being extracted in one shot. Additional examples are in Appendix A.

Extraction risk. One can view (n, p)-discoverable extraction as capturing the risk of a user extracting a particular target sequence as a function of the number of queries. This setting is reflective of production users: users can (and do) query LLMs

[2]The normalized edit distance between greedy and target outputs is 13.6% in character space and 16% in token space.

many times. In Figure 1, a user would only need 6 queries (in expectation) before generating the target, since it occurs with probability 16.2%. A user who extracts sensitive information after several attempts could be just as successful at exploiting this information, compared to if they had extracted it in one shot. This is because many sensitive sequences (e.g., phone numbers, credit card numbers) can be verified through external means.

3.2 How should one set on n and p?

Our definition introduces two related hyperparameters: number of queries n and probability p. In general, if it is easy to extract a particular target example $\boldsymbol{z}$, the number of queries n to generate $\boldsymbol{z}$ at least once is small (and thus p is large). The reverse holds for targets that are challenging to extract. That is, a specific model, sampling scheme, and example $\boldsymbol{z}$ define a trade-off between n and p such that $\boldsymbol{z}$ is (n, p)-discoverably extractable. Because of this trade-off, we can approximate (n, p)-discoverable extraction with just one query—with no overhead compared to discoverable extraction (Definition 2.1).

To see this, let us say that $p_{\boldsymbol{z}}$ is the probability of generating a suffix $\boldsymbol{z}_{a+1:a+k}$ for prefix $\boldsymbol{z}_{1:a}$, given a model, sampling scheme, and example $\boldsymbol{z} = \boldsymbol{z}_{1:a} \,\|\, \boldsymbol{z}_{a+1:a+k}$. This means that the probability of *not* generating $\boldsymbol{z}_{a+1:a+k}$ in a single draw from the sampling scheme is $1 - p_{\boldsymbol{z}}$, and the probability of not generating $\boldsymbol{z}_{a+1:a+k}$ in n independent draws is $(1 - p_{\boldsymbol{z}})^n$. Therefore, example $\boldsymbol{z}$ is (n, p)-discoverably extractable for n and p that satisfy $1 - (1 - p_{\boldsymbol{z}})^n \geq p$. Equivalently,

$$n \geq \frac{\log(1-p)}{\log(1-p_{\boldsymbol{z}})}. \tag{2}$$

In other words, we can easily find n given a fixed p and the probability of generating a sequence $p_{\boldsymbol{z}}$, and vice versa. In practice, for a given $\boldsymbol{z}$, we can compute $p_{\boldsymbol{z}}$ with just one query (Appendix C.3). This lets us use Equation (2), where we fix a desired minimum extraction probability p and find the corresponding queries n. In expectation, $n = \lceil \frac{1}{p_{\boldsymbol{z}}} \rceil$.

We verify that this one-query procedure for approximating n and p gives a reasonable estimate of (n, p)-discoverable extraction. That is, we confirm that we can use one query to produce $p_{\boldsymbol{z}}$ and Equation (2) for our measurements, rather than the more costly procedure of directly sampling a set of n sequences to estimate $\hat{p}_{\boldsymbol{z}}$ (the fraction of n queries that generate the target suffix) and then computing p (Appendix C.3). In Figure 2, we compare the two procedures. We plot the more costly, n-query-computed p (i.e., empirical p) as a function of the corresponding p computed using Equation (2) (i.e., theoretical p), and indeed the two match. For efficiency, we use Equation (2) in our experiments for verbatim extraction (Section 4).

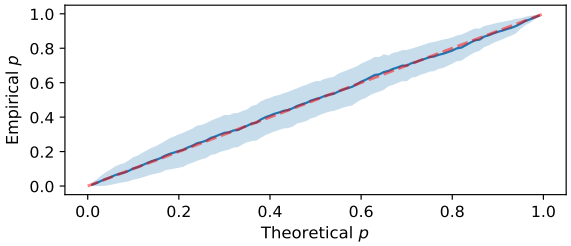


Figure 2: For 250 examples in the Pile (Wikipedia subset) and Pythia 6.9B, we check that generating n=1000 sequences and computing the probability a training example appears at least once in the set (**empirical** p) matches the **theoretical** p using Equation (2).

Choosing an extraction tolerance. The above are general observations about how the math works out; they do not reveal how one ought to set a "reasonable" threshold for the risk of extraction in practice. What values for n and p would connote that a particular target is "reasonably" extractable?

We deliberately do not prescribe specific choices for n and p, as "reasonable" choices depend on the type of information one is trying to extract (Cooper and Grimmelmann, 2024; Cooper et al., 2024). For example, one may be willing to release a model if n is small and p is high for extracting generic phrases, but this may not be tolerable for PII. We view this flexibility of (n, p)-discoverable extraction as a benefit: it allows practitioners to make fine-grained, context-specific decisions according to their respective levels of extraction-risk tolerance.

One possible way to reason about this tolerance is to set a threshold in terms of the expected computation limits of an adversary. If we assume that an adversary has a limited computation budget with which to query the model, we only need to ensure that n is larger than this budget, in order to minimize the risk of extraction. In practice, this n could be enforced through rate limiting.

3.3 Extending to non-verbatim extraction

So far, we have considered probabilistic extraction of sequences that exactly match the target. We now show how to relax (n, p)-discoverable extraction (Definition 3.1) to apply to non-verbatim matches.

Definition 3.2 (**(ϵ, n, p)-discoverable extraction**). For $\epsilon \in \mathbb{R}_{>0}$ and $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{V}^k$, define the set of k-length sequences $\mathbb{S}_\epsilon(\boldsymbol{b}) = \{\boldsymbol{c} \mid \mathsf{dist}(\boldsymbol{b}, \boldsymbol{c}) \leq \epsilon\}$, where $\mathsf{dist}\colon \mathbb{V}^k \times \mathbb{V}^k \to \mathbb{R}_{\geq 0}$ is a function that computes the distance between two k-length token sequences. Given a training example $\boldsymbol{z}$ that is split into an a-length prefix $\boldsymbol{z}_{1:a}$ and a k-length suffix $\boldsymbol{z}_{a+1:a+k}$, $\boldsymbol{z}$ is (ϵ, n, p)-*discoverably extractable* if

$$\Pr\left(\cup_{w\in[n]}\,(g_\phi \circ f_\theta)^k_w(\boldsymbol{z}_{1:a}) \in \mathbb{T}_\epsilon\right) \geq p,$$

where $\mathbb{T}_\epsilon = \{\boldsymbol{z}_{1:a} \parallel \boldsymbol{s} \mid \boldsymbol{s} \in \mathbb{S}_\epsilon(\boldsymbol{z}_{a+1:a+k})\}$.

As in (n, p)-discoverable extraction, we generate n independent sequences by sampling the output distribution of model f_θ using scheme g_ϕ. But here, if the probability of generating *any* suffix in $\mathbb{S}_\epsilon(\boldsymbol{z}_{a+1:a+k})$ *at least once* is larger than p, then we say $\boldsymbol{z}$ is (ϵ, n, p)-discoverably extractable with respect to f_θ and g_ϕ. In practice, this can be quite expensive to compute directly, as even for small ϵ the number of suffixes in $\mathbb{S}_\epsilon(\boldsymbol{z}_{a+1:a+k})$ may be very large. We provide additional discussion, including more efficient approximations, in Appendix D.

3.4 Connections to other definitions

Our definition for (n, p)-discoverable extraction can be related to other work. Notably, Carlini et al. (2019) motivate a measure of **canary memorization** using rank perplexity, where a canary is a unique sequence deliberately inserted into the model's training data. This work considers an adversary that sequentially queries the model to extract the canary, with guess/candidate canaries sorted from lowest to highest perplexity; the rank of the true canary quantifies how many guesses such an adversary would need to make before extracting it. While the secret-sharer attack that Carlini et al. (2019) propose only involves one guess, it is natural to consider how extraction rates scale with multiple attempts—just as we do with n queries (Section 4).

Work by Tiwari and Suh (2025), published shortly after ours, also studies extraction probabilities for different sampling strategies and models. They also consider non-verbatim probabilistic extraction (Section 3.3); we refer to Appendix D and their experimental results on this topic, and focus on verbatim memorization below. In Appendix B, we provide extensive discussion on connections to canary memorization and other related definitions.

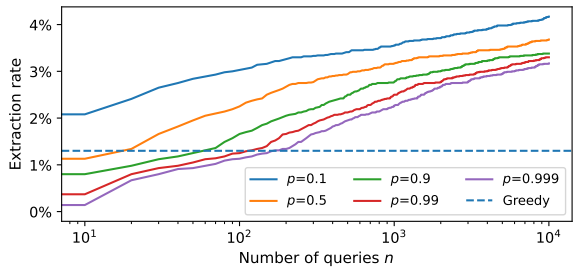
4 Experiments

We now demonstrate the utility of taking a probabilistic perspective on measuring extraction through experiments involving different model families. We show how (n, p)-discoverable extraction rates change as a function of n and p (Section 4.1), and evaluate how these rates increase for larger model sizes and training-data repetitions (Section 4.2). We also verify that our analysis reflects valid estimates of training-data extraction. To do so, we compare the (n, p)-discoverable extraction rate to the corresponding rate of generating test-data examples (Section 4.3). Altogether, our analysis illustrates that (n, p)-discoverable extraction provides more reliable measurements of extraction rates than greedy-sampled discoverable extraction, and also reveals a more nuanced picture of extraction risk in LLMs.

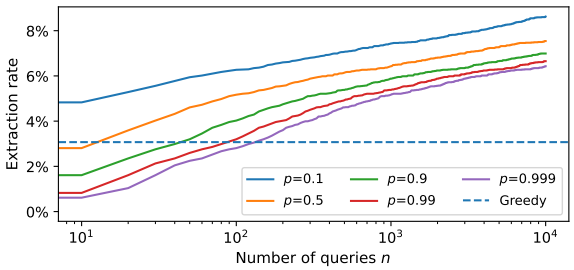
Setup. In this section, we analyze the Pythia model family (Biderman et al., 2023) and GPT-Neo 1.3B (Black et al., 2021). In each experiment, we measure extraction rates with respect to $10,000$ examples drawn from the Enron dataset, which is contained in the Pile (Gao et al., 2020)—the training dataset for both Pythia and GPT-Neo 1.3B. These extraction rates do not necessarily reflect the overall rates that would result for a representative sample of the entire training dataset. Throughout, we use one query and Equation (2) to compute (n, p)-discovable extraction (Definition 3.1). We use top-k sampling ($k = 40$, $T = 1$) and compare to the one-shot, greedy-sampled discoverable extraction rate (Definition 2.1). Following Carlini et al. (2021), in all cases, we use the first 50 tokens of each example as the prefix, and the subsequent 50 tokens as the suffix. In the Appendix, we provide results on the OPT (Zhang et al., 2022) and Llama model families (Touvron et al., 2023), as well as with different sampling schemes and training-data subsets.

4.1 How extraction rates vary with n and p

In Figures 3-5, we plot extraction rates for different models, according to various choices of n and p. The curves therefore represent the overall (n, p)-discoverable extraction rate over the given set of training examples, drawn from the Enron dataset; they capture extraction risk as it varies in terms of n and p (Section 3.1). In contrast, the rate for greedy-sampled discoverable extraction (Definition 2.1) does not convey this type of information; the greedy rate is, of course, constant, because greedy sampling is deterministic. In every experiment, there exist settings of n and p that catch cases of extraction that greedy sampling misses.



(a) Pythia 2.8B



(b) Pythia 12B

Figure 3: For $10,000$ examples from the Enron dataset, we plot variations in (n, p)-discoverable extraction rates for models of different sizes, according to different query budgets n and minimum extraction probability p.

This is clear because, on the same examples, (n, p)-discoverable extraction rates surpass greedy extraction rates—even for relatively modest values of n and p. Results across other model families, model sizes, datasets, and sampling schemes support these observations, and can be found in Appendices E-H.

We organize additional observations into two themes: reasoning about overall extraction risk for a single model (Figures 3 & 4) and comparisons of extraction risk between different models (Figure 5).

Overall model extraction risk. As expected from Equation (2), the (n, p)-discoverable extraction rate appears to have a log-linear relationship with n for all choices of p. As a result, there is a maximum amount of extraction that can be obtained for a particular model f_θ and sampling scheme g_ϕ over a given set of training data, regardless of the choice of p. That is, for each p, there exists an n at which we obtain the worst-case extraction rate—an upper bound on the possible extraction risk. For top-k sampling ($k = 40$, $T = 1$) with Pythia 2.8B on Enron, this worst-case rate is 9.04% (Figure 4), which is nearly $7\times$ the greedy rate (1.3%, Figure 3a).

Beyond worst-case extraction risk, important patterns emerge for different settings of n and p. Recall that low-p settings are the most permissive for deeming a target to be extractable: such settings are appropriate to consider when it may be a problem if there is even a small probability of generating the target at least once (e.g., for PII, see Section 3.2). Figure 3 plots this setting for $p = 0.1$. Even at very small n, the (n, p)-extraction rate is higher than the greedy rate for both Pythia 2.8B (1.3%) and 12B (3.07%). That is, for low p, greedy sampling *underestimates* extraction risk; this underestimate is large even for ($n > 3$), and becomes significantly larger as n increases.

From another perspective—the high-p, low-n setting—greedy-sampled discoverable extraction

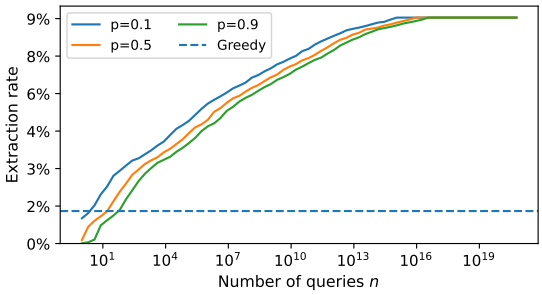


Figure 4: Maximum extraction (Enron, Pythia 2.8B).

can also be viewed to *overestimate* extraction. Consider $p = 0.999$, for which it is almost certain that a target sequence is generated at least once. While for large n, $p = 0.999$ shows that the greedy rate significantly underestimates extraction, for small n the (n, p) rate is *lower* than the greedy rate. For example, for Pythia 2.8B, the (n, p) rate only approaches the greedy rate when $n > 169$. Even for $p = 0.5$, where it is effectively a coin flip that the target is generated at least once, the (n, p) rate only approaches the greedy one when $n > 17$. That is, if we only allow (e.g., rate limit) a relatively small number of queries, the (n, p)-discoverable extraction rate can be kept quite low—indeed, close to 0% for high p and $n < 10$. From this perspective, greedy-sampled discoverable extraction—the metric computed in many model-release reports (e.g., Gemini Team et al., 2024; Llama Team, 2024)—may give an overly pessimistic picture of the amount of extraction experienced by end users in practice.

Extraction risk across models. Measuring (n, p)-discoverable extraction also reveals trends across models that are not apparent for greedy-sampled discoverable extraction. To show this, we compare extraction rates for Pythia 1B with GPT-Neo 1.3B (Black et al., 2021), which is a similar-sized model also trained on the Pile (Figure 5). We observe that Pythia 1B has a larger greedy extraction rate than GPT-Neo 1.3B; however, for the same

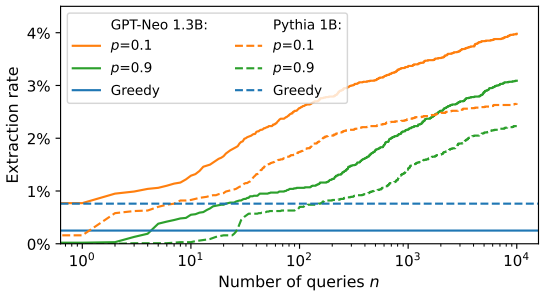


Figure 5: Comparing extraction rates across two different models, GPT-Neo 1.3B and Pythia 1B, using Enron.

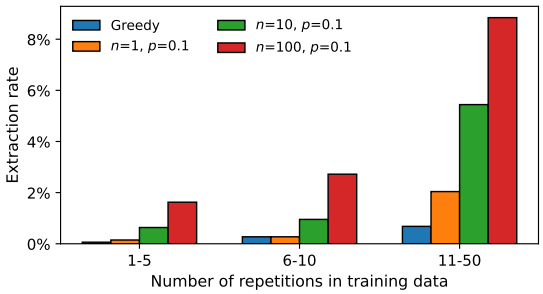


Figure 6: Rates for greedy discoverable and (n, p)-discoverable extraction using top-k sampling ($k = 40$, $T = 1$) on Pythia 2.8B. The (n, p) rates exceed the greedy rate, and the gap widens with more repetitions.

p at *at every* n, the (n, p)-discoverable extraction rate for GPT-Neo 1.3B is larger than for Pythia 1B. A practitioner who only measures discoverable extraction with greedy sampling would falsely conclude that Pythia 1B is at higher risk for extracting training data. In contrast, (n, p)-discoverable extraction implies the opposite. By providing a more reliable estimate of extraction risk in terms of n and p, our probabilistic measure facilitates better comparisons of extraction risk across models.

4.2 Evaluating model size and data repetition

It is well-known in the memorization literature that discoverable extraction rates tend to be higher for larger models (Carlini et al., 2022; Biderman et al., 2024; Lu et al., 2024; Tirumala et al., 2022; Mireshghallah et al., 2022) and that repeated training-data examples are more likely to be extracted (Lee et al., 2021). Our results confirm these trends: the greedy rates for Pythia 1B (Figure 5), 2.8B (Figure 3a), and 12B (Figure 3b) are 0.76%, 1.3%, and 3.07%, respectively; the greedy rate also increases as a function of repetitions (Figure 6). Our experiments with (n, p)-discoverable extraction follow the same patterns and also offer further insights. To the extent that greedy-based extraction attacks may be underestimating extraction at high n and relatively low p, *these underestimates are worse for larger models and for target data that has a higher number of repetitions*.

Model size. Similar to greedy-sampled discoverable extraction, (n, p)-discoverable extraction rates increase with model size for models in the same family. For example, at all n and p, the (n, p)-discoverable extraction rates are higher for Pythia 12B compared to Pythia 2.8B (Figure 3). Further, the gap between the greedy-sampled discoverable extraction rate and (n, p)-discoverable extraction rates increases for larger models. On the Enron training-data subset, for Pythia 2.8B, the gap between the greedy rate (1.3%, see Figure 3a) and maximum extraction rate (9.04%, see Figure 4) is 7.74%. By comparison, the gap for Pythia 12B between the greedy (3.07%, see Figure 3b) and maximum rate (16.07%, see Appendix E) is 13%. In a similar vein, it takes fewer queries for larger models to match the greedy rate. For example, at $p = 0.9$, Pythia 1B needs to generate $n = 150$ sequences to exceed the greedy rate, while Pythia 12B only needs to generate $n = 40$ sequences.

Example repetitions. The (n, p)-discoverable extraction rate also increases as a function of training-example repetitions, and the gap between the greedy and the (n, p) rates is wider for more repetitions. For this experiment, depicted in Figure 6, we find phone numbers that are replicated within the Pile (Pythia's training dataset). We use each phone number as the target suffix, and the preceding text as the prefix to serve as the prompt.

4.3 Validating (n, p)-discoverable extraction

A natural concern for using large n and small p is that this setting is too permissive to provide meaningful measurements of extraction. That is, for sufficiently large n, a model might just happen to output the target suffix with low probability—even if that suffix was not memorized. If this is the case, then our measurements for (n, p)-discoverable extraction might mix together true instances of extracting memorized training data with instances of generating training data by happenstance; our measurements might not be valid estimates for memorization.

We investigate this possibility in Figure 7, and our results support that our measurements for (n, p)-discoverable extraction are indeed valid. For Pythia 2.8B, we compare extraction of training data from Enron to the generation of unseen test data. For the test data, we use $10,000$ emails from the

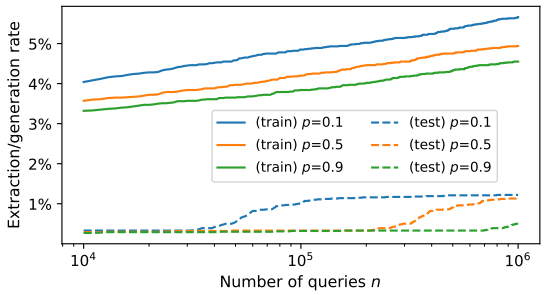


Figure 7: Validating training-data extraction. For Pythia 2.8B, we compare extraction rates of training data (**train**) from Enron to the rates of generating verbatim unseen test data (**test**) from TREC 2007 Spam.

TREC 2007 Spam classification dataset (Bratko et al., 2006). We find that, at all settings for p, the rate of generating test data is very low, and is significantly lower than the corresponding rate of generating training data. Even for $p = 0.1$ and very large n, the test-data generation rate is less than 1%—compared to over 5% for training-data extraction. In fact, the test-data generation rate is effectively 0% for $p = 0.1$ until $n > 50,000$. For $p = 0.9$, it is 0.3% even after $500,000$ queries, compared to 4.4% for the (n, p)-discoverable extraction rate—a difference of over an order of magnitude.

In other words, for all settings of p, the number of queries n needed to generate unseen test data is orders of magnitude larger than for generating training data. We find that it is generally challenging to generate test data—even for low p—and especially in comparison to our measurements for extracting training data. This supports that, in our measurements of (n, p)-discoverable extraction, matches between training-example targets and generated suffixes are almost surely due to memorization.

5 Conclusion

In this paper, we take a probabilistic perspective on measuring training-data extraction in language models. This represents a significant departure from prior work and model-release reports, which tend to measure discoverable extraction: one-shot, yes-or-no determinations of extraction using deterministic greedy sampling. Instead, our measure for (n, p)-discoverable extraction captures a continuous notion of the risk of extracting a target example: we recast discoverable extraction in terms of the number of queries n one would need to make to extract a target at least once with probability p under a chosen sampling scheme. To conclude, we revisit three benefits of our probabilistic approach.

Reliable quantification of extraction. Through extensive experimentation, we show that (n, p)-discoverable extraction provides more reliable measurement of extraction rates and facilitates more valid comparisons of extraction rates across models. Greedy-sampled discoverable extraction—the common approach in prior work—often significantly underestimates the overall rate of possible extraction (Sections 3.1 & 4.1), with the degree of underestimation increasing for larger models (Section 4.2). However, for low query budgets n (and even for relatively low p), the (n, p)-discoverable extraction rate is often lower than the greedy rate. This suggests that prior reported greedy rates may also overestimate the amount of extraction experienced by end users in practice (Section 4.1).

No computational overhead. Importantly, we obtain these results with no additional cost compared to traditional discoverable extraction. This is because, in practice, (n, p)-discoverable extraction can be computed with just one query (Section 3.2). So, with no overhead, our measure yields more nuanced information about extraction risk.

Risks beyond overall extraction. The results we present concern overall extraction rates; however, (n, p)-discoverable extraction can also be leveraged for finer-grained analysis of risks associated with different levels of data sensitivity (Section 3.2). For example, while extracting generic phrases may not pose a significant risk, even rare PII leakages can be problematic. Our measure allows practitioners to adjust n and p based on the extraction-risk tolerance appropriate to specific contexts. Further, future research could adapt our work beyond memorization—to measure the risk of a model outputting any target sequence, such as those that reflect harmful or otherwise undesirable content.

6 Limitations

Extraction is fundamentally challenging to measure; improvements come as researchers and practitioners discover new attacks (e.g., Nasr et al., 2023, 2025; Su et al., 2024). We study a setting with a relatively benign adversary—one with API access only and limited side information—and defer study of more powerful adversaries to other work. Additionally, we make no distinction between extracting different types of target sequences. In practice, some target sequences (e.g., PII) are significantly more sensitive than others (Sections 3.2 & 5), so their extraction rates should be measured separately.

We only cursorily consider extraction of PII (specifically, phone numbers) for the purpose of measuring the effect of repetitions (Section 4.3). We defer to future work to investigate extraction rates of different types of PII and other sensitive targets.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, et al. 2023. PaLM 2 Technical Report. *Preprint*, arXiv:2305.10403.

Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R Varshney. 2020. Mirostat: A neural text decoding algorithm that directly controls perplexity. *arXiv preprint arXiv:2007.14966*.

Stella Biderman, USVSN PRASHANTH, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raff. 2024. Emergent and predictable memorization in large language models. *Advances in Neural Information Processing Systems*, 36.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow.

Sebastian Bordt, Harsha Nori, Vanessa Rodrigues, Besmira Nushi, and Rich Caruana. 2024. Elephants never forget: Memorization and learning of tabular data in large language models. *arXiv preprint arXiv:2404.06209*.

Andrej Bratko, Bogdan Filipič, Gordon V Cormack, Thomas R Lynam, and Blaž Zupan. 2006. Spam filtering using statistical data compression models. *The Journal of Machine Learning Research*, 7:2673–2698.

Gavin Brown, Mark Bun, Vitaly Feldman, Adam Smith, and Kunal Talwar. 2021. When is memorization of irrelevant training data necessary for high-accuracy learning? In *Proceedings of the 53rd annual ACM SIGACT symposium on theory of computing*, pages 123–132.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2022. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX security symposium (USENIX security 19)*, pages 267–284.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113.

A. Feder Cooper, Christopher A. Choquette-Choo, Miranda Bogen, Matthew Jagielski, Katja Filippova, Ken Ziyu Liu, Alexandra Chouldechova, Jamie Hayes, Yangsibo Huang, Niloofar Mireshghallah, Ilia Shumailov, Eleni Triantafillou, Peter Kairouz, Nicole Mitchell, Percy Liang, Daniel E. Ho, Yejin Choi, Sanmi Koyejo, Fernando Delgado, James Grimmelmann, Vitaly Shmatikov, Christopher De Sa, Solon Barocas, Amy Cyphert, Mark Lemley, danah boyd, Jennifer Wortman Vaughan, Miles Brundage, David Bau, Seth Neel, Abigail Z. Jacobs, Andreas Terzis, Hanna Wallach, Nicolas Papernot, and Katherine Lee. 2024. Machine Unlearning Doesn't Do What You Think: Lessons for Generative AI Policy, Research, and Practice. *arXiv preprint arXiv:2412.06966*.

A. Feder Cooper, Jonathan Frankle, and Christopher De Sa. 2022. Non-Determinism and the Lawlessness of Machine Learning Code. In *Proceedings of the 2022 Symposium on Computer Science and Law*, CSLAW '22, page 1–8, New York, NY, USA. Association for Computing Machinery.

A. Feder Cooper and James Grimmelmann. 2024. The Files are in the Computer: Copyright, Memorization, and Generative AI. *arXiv preprint arXiv:2404.12590*.

A. Feder Cooper, Katherine Lee, James Grimmelmann, Daphne Ippolito, Christopher Callison-Burch, Christopher A. Choquette-Choo, Niloofar Mireshghallah, Miles Brundage, David Mimno, Madiha Zahrah Choksi, Jack M. Balkin, Nicholas Carlini, Christopher De Sa, Jonathan Frankle, Deep Ganguli, Bryant Gipson, Andres Guadamuz, Swee Leng Harris, Abigail Z. Jacobs, Elizabeth Joh, Gautam Kamath, Mark Lemley, Cass Matthews, Christine McLeavey, Corynne McSherry, Milad Nasr, Paul Ohm, Adam Roberts, Tom Rubin, Pamela Samuelson, Ludwig Schubert, Kristen Vaccaro, Luis Villa, Felix Wu, and Elana Zeide. 2023. Report of the 1st Workshop on Generative AI and Law. *arXiv preprint arXiv:2311.06477*.

Sunny Duan, Mikail Khona, Abhiram Iyer, Rylan Schaeffer, and Ila R Fiete. 2024. Uncovering Latent Memories: Assessing Data Leakage and Memorization Patterns in Large Language Models. *arXiv preprint arXiv:2406.14549*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Vitaly Feldman and Chiyuan Zhang. 2020. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891.

Giorgio Franceschelli, Claudia Cevenini, and Mirco Musolesi. 2024. Training Foundation Models as Data Compression: On Information, Model Weights and Copyright Law. *Preprint*, arXiv:2407.13493.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*.

Gemini Team, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024a. Gemma: Open Models Based on Gemini Research and Technology. *arXiv preprint arXiv:2403.08295*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024b. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Jing Huang, Diyi Yang, and Christopher Potts. 2024. Demystifying verbatim memorization in large language models. *arXiv preprint arXiv:2407.17817*.

Aly M Kassem, Omar Mahmoud, Niloofar Mireshghallah, Hyunwoo Kim, Yulia Tsvetkov, Yejin Choi, Sherif Saad, and Santu Rana. 2024. Alpaca against Vicuna: Using LLMs to Uncover Memorization of LLMs. *arXiv preprint arXiv:2403.04801*.

Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. 2023. ProPILE: Probing Privacy Leakage in Large Language Models. *Preprint*, arXiv:2307.01881.

Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. 2024. Madlad-400: A multilingual and document-level large audited dataset. *Advances in Neural Information Processing Systems*, 36.

Katherine Lee, A. Feder Cooper, and James Grimmelmann. 2023. Talkin' 'Bout AI Generation: Copyright and the Generative-AI Supply Chain. *arXiv preprint arXiv:2309.08133*.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2021. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*.

Llama Team. 2024. The Llama 3 Herd of Models. *Preprint*, arXiv:2407.21783.

Xingyu Lu, Xiaonan Li, Qinyuan Cheng, Kai Ding, Xuanjing Huang, and Xipeng Qiu. 2024. Scaling laws for fact memorization of large language models. *arXiv preprint arXiv:2406.15720*.

Fatemehsadat Mireshghallah, Archit Uniyal, Tianhao Wang, David K Evans, and Taylor Berg-Kirkpatrick. 2022. An empirical analysis of memorization in fine-tuned autoregressive language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1816–1826.

Krishna Kanth Nakka, Ahmed Frikha, Ricardo Mendes, Xue Jiang, and Xuebing Zhou. 2024. PII-Compass: Guiding LLM training data extraction prompts towards the target PII via grounding. *Preprint*, arXiv:2407.02943.

Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A. Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. 2023. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*.

Milad Nasr, Javier Rando, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A. Feder Cooper, Daphne Ippolito, Christopher A. Choquette-Choo, Florian Tramèr, and Katherine Lee. 2025. Scalable Extraction of Training Data from Aligned, Production Language Models. In *The Thirteenth International Conference on Learning Representations*.

Avi Schwarzschild, Zhili Feng, Pratyush Maini, Zachary C Lipton, and J Zico Kolter. 2024. Rethinking llm memorization through the lens of adversarial compression. *arXiv preprint arXiv:2404.15146*.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. 2023. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789*.

Victoria Smith, Ali Shahin Shamsabadi, Carolyn Ashurst, and Adrian Weller. 2023. Identifying and mitigating privacy risks stemming from language models: A survey. *arXiv preprint arXiv:2310.01424*.

Pierre Stock, Igor Shilov, Ilya Mironov, and Alexandre Sablayrolles. 2022. Defending against Reconstruction Attacks with Rényi Differential Privacy. *Preprint*, arXiv:2202.07623.

Ellen Su, Anu Vellore, Amy Chang, Raffaele Mura, Blaine Nelson, Paul Kassianik, and Amin Karbasi. 2024. Extracting Memorized Training Data via Decomposition. *Preprint*, arXiv:2409.12367.

Kushal Tirumala, Aram Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. 2022. Memorization without overfitting: Analyzing the training dynamics of large language models. *Advances in Neural Information Processing Systems*, 35:38274–38290.

Trishita Tiwari and G. Edward Suh. 2025. Sequence-Level Leakage Risk of Training Data in Large Language Models. *Preprint*, arXiv:2412.11302.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *Preprint*, arXiv:2302.13971.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.

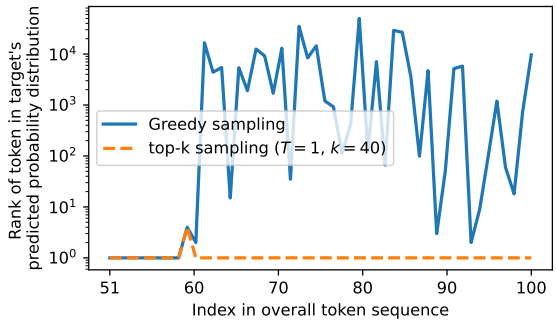
Zhepeng Wang, Runxue Bao, Yawen Wu, Jackson Taylor, Cao Xiao, Feng Zheng, Weiwen Jiang, Shangqian Gao, and Yanfu Zhang. 2024. Unlocking memorization in large language models with dynamic soft prompting. *arXiv preprint arXiv:2409.13853*.

Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023. Counterfactual memorization in neural language models. *Advances in Neural Information Processing Systems*, 36:39321–39362.

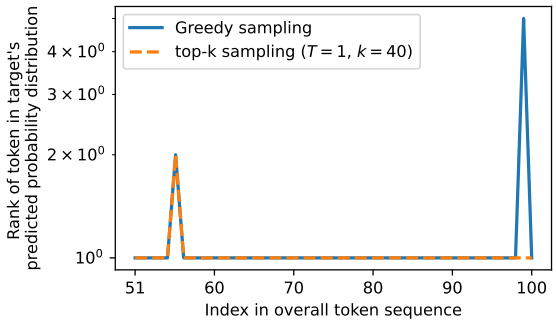
Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *Preprint*, arXiv:2205.01068.

A More examples of how greedy sampling can miss extraction

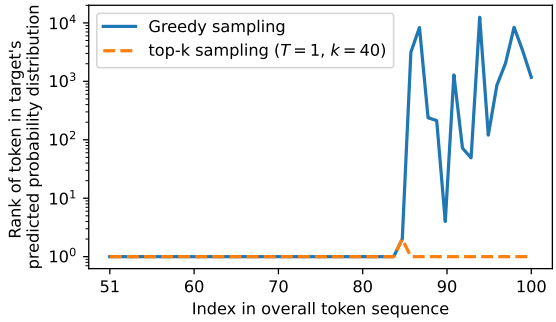
We give four more of examples in Figure 8 of cases where greedy sampling misses extraction (Section 3.1).



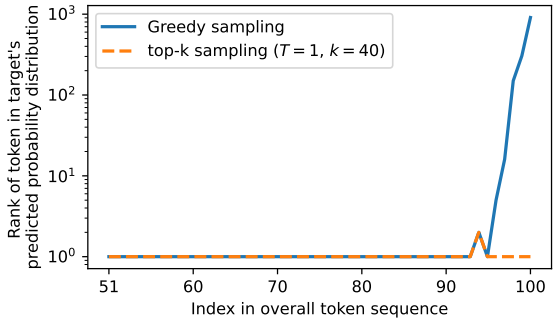
Prefix: *"Wygoda Sierakowska\n\nWygoda Sierakowska () is a village in the administrative district of Gmina Sierakowice, within Kartuzy County, Pomeranian Voivodeship"*
Greedy suffix: *", in northern Poland. It lies approximately south-west of Sierakowice, west of Kartuzy, and west of the regional capital Gdańsk.\n\nFor details of the history of the region,"*
Target suffix: *", in northern Poland. It lies approximately east of Sierakowice, west of Kartuzy, and west of the regional capital Gdańsk.\n\nFor details of the history of the region, see History"*
Edit distance between greedy and target suffix is 4% (tokens) / 0.85% (characters). Top-k sampling outputs target suffix with probability 9.74%.



Prefix: *"Yoshihiro Nikawadori\n\nYoshihiro Nikawadori (荷川取義浩, Nikawadori Yoshihiro, born 4 December 1961) is a Japanese former handball"*
Greedy suffix: *" player who competed in the 1984 Summer Olympics.\n\nReferences\n\nCategory:1961 births\nCategory:Living people\nCategory:Japanese male handball players\nCategory:Olympic handball players of Japan\nCategory:Handball players at the 1984 Summer"*
Target suffix: *" player who competed in the 1988 Summer Olympics.\n\nReferences\n\nCategory:1961 births\nCategory:Living people\nCategory:Japanese male handball players\nCategory:Olympic handball players of Japan\nCategory:Handball players at the 1988 Summer"*
Edit distance between greedy and target suffix is 10% (tokens) / 11.11% (characters). Top-k sampling outputs target suffix with probability 9.09%.



Prefix: *"Greece at the 1984 Summer Paralympics\n\nGreece competed at the 1984 Summer Paralympics in Stoke Mandeville, Great Britain and New York City, United States. 3 competitors from Greece won no medals and so"*
Greedy suffix: *" did not place in the medal table.\n\nSee also \n Greece at the Paralympics\n Greece at the 1984 Summer Olympics\n\nReferences \n\nCategory:Nations at the 1984 Summer Paralympics\n1984\nSummer Paral"*
Target suffix: *" did not place in the medal table.\n\nSee also \n Greece at the Paralympics\n Greece at the 1984 Summer Olympics\n\nReferences \n\nCategory:Greece at the Paralympics\nCategory:1984 in Greek sport"*
Edit distance between greedy and target suffix is 22% (tokens) / 21.81% (characters). Top-k sampling outputs target suffix with probability 14.93%.



Prefix: *"Conus patae\n\nConus patae, common name Pat's cone, is a species of sea snail, a marine gastropod mollusk in the family Conidae, the cone snails and their allies.\n\n""*
Greedy suffix: *"Like all species within the genus Conus, these snails are predatory and venomous. They are capable of "stinging" humans, therefore live ones should be handled carefully or not at all.\n\nDescription\nThe size of an adult"*
Target suffix: *"Like all species within the genus Conus, these snails are predatory and venomous. They are capable of "stinging" humans, therefore live ones should be handled carefully or not at all.\n\nDistribution\nThis species occurs in the"*
Edit distance between greedy and target suffix is 12% (tokens) / 9.82% (characters). Top-k sampling outputs target suffix with probability 7.5%.

Figure 8: Examples from the Pile (Wikipedia subset) of failures of greedy-sampled discoverable extraction from Pythia 12B. We prompt with the first 50 tokens in the target and test extraction with the subsequent 50 tokens. We also plot top-k sampling ($k = 40$, $T = 1$) generating the target suffix. We highlight matches with the target tokens in blue and mismatches in red. Determination of a (mis)match involves comparing tokens at the same index. Characters in the greedy suffix may match characters in the target suffix, but the indices can differ, causing a mismatch.

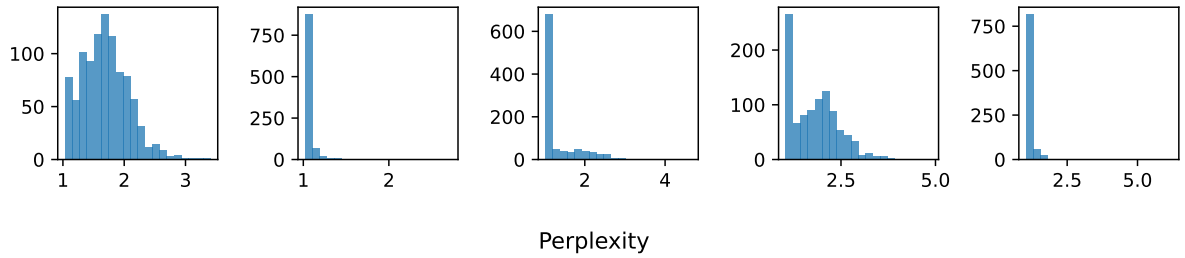


Figure 9: Distributions of perplexity values for 1000 generated sequences, each prompted using one of five training-example prefixes. The assumption in work by Carlini et al. (2019) that the empirical distributions are approximately (skewed) Gaussian, in order to estimate rank perplexity without sampling many times, is not appropriate for this setting. Unlike in their setting for studying canary memorization, here, training examples are not restricted to a bounded domain (e.g., phone numbers or social security numbers).

B Comparison to other extraction, reconstruction, and memorization definitions

In formulating our definition of probabilistic extraction (Section 3), our aim was to define something that 1) can easily be operationalized for production LLMs without needing to retrain multiple models, 2) roughly corresponds to the capabilities of a typical user, and 3) aligns with the risks posed by extraction. Numerous definitions of extraction, reconstruction, and memorization have been proposed, each with their own trade-offs that are not necessarily aligned with ours. We explore some of these definitions in this appendix.

Canary memorization. As discussed in Section 3.4, Carlini et al. (2019) measure the unintended memorization of random strings (called **canaries**) inserted into the training dataset via their **rank perplexities**. That is, an inserted canary has rank perplexity i if the model perplexity on that canary is the i-th highest among all possible canaries that could have been sampled and inserted. (The possible canaries are sorted from lowest to highest perplexity.) This rank naturally corresponds to the number of guesses an adversary would need to make before correctly guessing the true canary, if the adversary guessed canaries in order from most to least likely (i.e., lowest to highest perplexity). This is a fairly natural setup for bounded sets of potential canaries; however, in our setting, it would be intractable to enumerate all possible token sequences of a given length.

Carlini et al. (2019) use a (skewed) Gaussian approximation to model the expected distribution of perplexity values over a number of sequences, such that one can estimate rank perplexity of a target without having to sample a large number of sequences. The assumption of a Gaussian approximation being a good distributional fit is tailored to format-constrained canaries. Even though this seems to align in principle with our n-query setup, in Figure 9, we show that this approach is not a good fit for our setting. We consider five different training examples from the Enron dataset, which is contained in Pythia's training dataset (Gao et al., 2020). For each example, we use its prefix to produce 1000 generations, and we plot the perplexity distribution. Clearly, a (skewed) Gaussian is a relatively poor fit in all cases. Importantly, the empirical distributions vary from example to example; one could not estimate a distribution for one training example and expect this distribution to be a good fit for another example. This makes estimating rank perplexity challenging in our general, non-format-constrained setting.

Information-theoretic definitions. A number of memorization definitions have an information-theoretic flavor that rigorously capture a model's dependence on its training data. Unfortunately, estimating these dependencies often requires training many models, which is infeasible in our LLM setting.

Brown et al. (2021) measure the mutual information between the training dataset and the model output, conditioned on the data-generating distribution. This definition neatly captures the essence of memorization: the amount of information that is contained about the dataset in the model, which is not a property of the underlying data distribution. They present two experiments as a proof-of-concept of their lower bounds. They sample the training data according to the specific learning task that exhibits their information-theoretic lower bounds, and then attack a logistic regression model and a single-hidden-layer feed forward network trained on this data. Importantly, their bounds require knowing the data-generating distribution.

Other definitions similarly consider a counterfactual approach to memorization, which requires estimating the model's performance with and without a particular training example (Feldman and Zhang, 2020; Zhang et al., 2023). While Zhang et al. (2023) use subsampling to reduce the number of models that need to be trained, they still require on the order of hundreds of models to get an estimate of counterfactual memorization of the training data.

Memorization as compression. Some works conceive of memorization as compression (Lee et al., 2023; Cooper et al., 2023; Cooper and Grimmelmann, 2024; Franceschelli et al., 2024; Schwarzschild et al., 2024). As a concrete example, Schwarzschild et al. (2024) study memorization by measuring the compression properties of an LLM with respect to a training example. They measure the length of the smallest prompt that will generate the target example in question as an indication of whether the material was memorized. In our setting, though, we are not necessarily concerned with extraction being proportional to the length of the prompt. One could imagine a very short but unusual prefix that reliably extracts a target, versus a typical but longer prefix doing the same. Clearly, when measuring the risk of extraction by a regular user, the latter prompt is at higher risk of revealing the target even though it is longer, so this definition does not align with our setting.

Prompt engineering to measure extraction. There is another line of work that uses clever prompting strategies to extract information from a target LLM. For example, Kassem et al. (2024) use one LLM to find prompts that elicit extraction in another. Wang et al. (2024) construct prefix-dependent soft prompts to extract a given suffix. Both attacks involve optimizing over multiple prompts for each target sequence, which is more computationally expensive than our approach.

Probabilistic extraction. Several other works study ideas related to probabilistic extraction. Tiwari and Suh (2025) published work shortly after our own, which is most directly related to what we study in this paper. They develop a similar motivation to our own, recognizing that measuring extraction beyond a one-shot approach with greedy sampling is 1) more reflective of realistic settings and 2) likely to show that prior work on discoverable extraction that takes a one-shot approach underestimates the true risk of extraction. Similar to our work, they examine several sampling strategies and models (Llama and OPT). They also discuss a similar strategy for quantifying non-verbatim probabilistic memorization, where the extracted sequence is within a given edit distance of the target. They categorize their findings into six different patterns. In contrast, we show that it is possible to derive a relationship between n and p for (n, p)-discoverable extraction in just one query (Section 3.2). We additionally run experiments to confirm that our extraction rates are capturing valid instances of memorization, as opposed to (potentially low-probability) generation of any target suffixes for large query budgets (Section 4.3).

Stock et al. (2022) study reconstruction attacks, and show that Rényi differential privacy with DP-SGD can protect against reconstructing (i.e., extracting) canaries (which they also refer to as secrets) in GPT-2 fine-tuning experiments. They develop a "lazy sampling" approach to identify target sequences that are likely to be reconstructed. While we use probability p to reason about the confidence that a target is in the set of n generations, their analysis uses expectations.

Kim et al. (2023) develop black-box and white-box probing methods for studying PII-leakage risk in LLMs. They assess PII leakage with several metrics, including quantifying the likelihood of a particular r-length, PII-containing target being generated in response to a q-length prompt. With their particular focus on PII and data subjects, they also introduce a related metric, which quantifies the percentage of data subjects whose PII is leaked within k queries to the model (with probability of leakage being greater than $\frac{1}{k}$).

Nakka et al. (2024) also specifically study PII leakage from LLMs. In a subset of their experiments, they evaluate manual template-based prompts using top-k sampling and 128 repeated queries. For two of the manual templates, they observe an increase in extraction rate (compared to a single query). While this type of analysis is of a similar flavor to ours, it is not the focus of their work and is limited in scope.

Cooper and Grimmelmann (2024) suggest measuring the probability of extraction for particular sequences, rather than relying on one-shot extraction attempts. However, building on their prior work (Cooper et al., 2022), their focus is on why such an approach would be interesting for a legal audience. They do not investigate this idea with tools in machine learning, as we do here.

C Experimental setup

We provide additional details on the models we analyze, the datasets from which we draw prefixes and suffixes to test, and the procedure for using Equation (2) to compute the relationship between n and p for a given training example $\boldsymbol{z}$ using just one query.

C.1 Models

We use GPT-Neo 1.3B (Black et al., 2021), the Pythia model family (Biderman et al., 2023) (1B, 2.8B, 6.9B, and 12B), the Llama 1 model family (7B, 13B) (Touvron et al., 2023), and the OPT model family (350M, 1.3B, 2.7B, 6.7B) for our experiments. Both GPT-Neo 1.3B and the Pythia model family are open-weight and open-data models trained by EleutherAI on the Pile (Gao et al., 2020). The Llama and OPT families are open-weight models released by Meta; while the Llama paper includes information on the training-data mix, full knowledge of the Llama's and OPT's training datasets is not publicly available.

In the main paper and Appendices E-G, we primarily include experiments with the Pythia model family on the Enron training-data subset. We also run an experiment with GPT-Neo 1.3B on Enron, in Figure 5 and run our Equation (2) verification experiment with Pythia 6.9B on Wikipedia data in Figure 2. In Appendix H, we include experiments for all model families on other training-data subsets and proxies.

C.2 Datasets

We use a variety of datasets, from which we draw examples that we divide into prefixes for prompts and suffixes to check against generations. We briefly discuss training datasets and proxy datasets according to their corresponding models.

Pythia and GPT-Neo 1.3B. These open-weight, open-data models were trained on The Pile (Gao et al., 2020). For these models, we test for extraction on different training-data subsets that are included in the Pile: **Enron**, **Wikipedia**, and **GitHub** subsets. For more information on The Pile, we refer the reader to EleutherAI. For the Wikipedia and GitHub subsets, we relied on the linked downloaders. As noted above, in the main paper we primarily run experiments on the Enron subset. In Appendix H, we run experiments on Pythia models for Wikipedia and GitHub. All three datasets have $10,000$ examples.

Llama and OPT. We do not know the exact training datasets for Llama and OPT, so we uses proxies for drawing examples to test for extraction. Since we know that Llama relied on Common Crawl data for training, we use $10,000$ examples drawn from Common Crawl. It is, of course, possible that the examples we use were not contained in the OPT or Llama training datasets. We expect that our estimates for extraction to be lower for these models.

Test dataset. For our experiment for validating (n, p)-discoverable extraction by comparing to generation of test data in Pythia 2.8B, we use the Trek 2007 Spam classification dataset (Bratko et al., 2006).

C.3 Computing (n, p)-discoverable extraction

In the main paper, we compute (n, p)-discoverable extraction using two different procedures. For the most part, we use Equation (2), which provides the relationship between n and p for an example $\boldsymbol{z}$ using only one query to the model to find $p_{\boldsymbol{z}}$. This provides an efficient approximation of actually sampling n queries to compute extraction probabilities. We discuss both procedures in a bit more detail here. We use the method involving just one query for all experiments except for Figure 2, where we verify that this procedure matches empirically with using n queries.

Computations of probabilistic extraction with just one query. To use Equation (2) to compute the relationship between n and p, we use one query to compute $p_{\boldsymbol{z}}$ for a given training example $\boldsymbol{z}$. To do so, we sequentially feed the example $\boldsymbol{z}$ into the model f_θ one token at a time. For each token, the model produces a conditional probability distribution over the vocabulary of the next token given the previous tokens. These conditional probabilities represent the base likelihood of observing a particular token at each position in the sequence. We use the sampling scheme g_ϕ to post-process these conditional probabilities. For example, for top-40 sampling and $T = 1$, we set all token probabilities, except for

the 40 tokens with the highest probabilities, to 0, normalize the 40 remaining non-zero token probabilities, and use Equation (1) with $T = 1$ to re-weight the model f_θ's base conditional probabilities. We then select the next actual token in $\boldsymbol{z}$; we add this token's conditional log probability to a running sum, and repeat the process for the next token, and so on. Altogether, we compute the overall log likelihood of the entire sequence $\boldsymbol{z}$ under both the model f_θ and the sampling scheme g_ϕ by adding the transformed conditional log probabilities for each token in the sequence. We use this overall log likelihood to produce $p_{\boldsymbol{z}}$, which we can then use with Equation (2) to derive n for any fixed p, or vice versa.

Testing different prefix and suffix length with no additional queries. Note that, for the above procedure, we can actually test *any* prefix and suffix length for $\boldsymbol{z}$ using just one query. When we compute $p_{\boldsymbol{z}}$, this depends on adding together the conditional log probability for each token. For a given suffix $\boldsymbol{z}_{a+1:a+k}$, this just involves adding together the conditional log probabilities of the tokens at indices $a+1, \ldots, a+k$ after having already processed the prefix $\boldsymbol{z}_{1:a}$ through the model. If we keep track of the per-index conditional log probabilities for the whole sequence, we can just change which ones we sum together (i.e., change a and k) on the fly; with just one query to the model, we obtain enough information to compute $p_{\boldsymbol{z}}$ for different prefix and suffix lengths, and could examine how these differences impact extraction.

Aside from the other benefits of our approach with respect to surfacing extraction risk, this is another clear benefit in comparison to traditional discoverable extraction (Definition 2.1). Since discoverable extraction has traditionally involved yes-or-no determinations that just compare the generated output with the target suffix, testing for different prefix and suffix lengths entails making different queries to the model using each prefix as a prompt. Here, by examining probabilities, we are able to efficiently compute different measurements of extraction with greater flexibility, and with no additional queries to the model.

n-shot computations of probabilistic extraction. The above procedure works in one query by computing $p_{\boldsymbol{z}}$ directly for the training example $\boldsymbol{z}$. It does not actually sample different generations using a prefix $\boldsymbol{z}^t_{1:a}$ and analyzing output suffixes $\boldsymbol{z}^o_{a+1:a+k}$ to see if they match the target suffix $\boldsymbol{z}^t_{a+1:a+k}$. We can use n queries to directly approximate $p_{\boldsymbol{z}}$: n independent times, we prompt the model f_θ with $\boldsymbol{z}_{1:a}$ and then sample with g_ϕ; for this given n, we compute

$$\hat{p}_{\boldsymbol{z}} = \frac{\sum_{w \in [n]} \mathbf{1}[\boldsymbol{z}^{o,w}_{a+1:a+k} = \boldsymbol{z}^t_{a+1:a+k}]}{n}, \tag{3}$$

which we then use to compute the corresponding p similarly to Equation (2), using $\hat{p}_{\boldsymbol{z}}$ in place of $p_{\boldsymbol{z}}$, i.e.,

$$1 - (1 - \hat{p}_{\boldsymbol{z}})^n \geq p.$$

D Extending to non-verbation extraction of targets

In Section 3.3, we extend the definition of (n, p)-discoverable extraction (Definition 3.1) from capturing only verbatim extraction of targets to also account for non-verbatim extraction of targets. We introduce a general variation of our definition (Definition 3.2) that includes a distance function dist: $\mathbb{V}^k \times \mathbb{V}^k \to \mathbb{R}_{\geq 0}$ and an $\epsilon \in \mathbb{R}_{>0}$ hyperparameter for the maximum allowable distance between the generated text and the target, such that the generated text would still count as a successful extraction.

We deliberately leave this definition general, with the choice of distance function dist and ϵ up to the user. In this appendix, we discuss an instantiation of Definition 3.2 for the Hamming distance (Appendix D.1). In relation to this discussion, we then briefly show how the definition could be modified for other edit-distance metrics (Appendix D.2).

D.1 An example instantiation with the Hamming distance

In our definition for (ϵ, n, p)-discoverable extraction (Definition 3.2), the implemented check for verifying extraction is modified from the check in the verbatim definition. Rather than checking for exact equality between the generated text and the target, we instead check if the generated suffix is a member of the set of all suffixes that are within ϵ distance (where distance is computed with dist) of the target. Here, we discuss a concrete instantiation of this definition with the Hamming distance—i.e., suffixes that are within ϵ token substitutions of the target suffix.

That is, for $\epsilon \geq 1$, we define the set of k-length sequences $\mathbb{S}_\epsilon(\boldsymbol{b}) = \{\boldsymbol{c} \mid \mathsf{HammingDistance}(\boldsymbol{b}, \boldsymbol{c}) \leq \epsilon\}$; these are the possible suffixes. We check the generated text for membership in the set $\mathbb{T}_\epsilon = \{\boldsymbol{z}_{1:a} \parallel \boldsymbol{s} \mid \boldsymbol{s} \in \mathbb{S}_\epsilon(\boldsymbol{z}_{a+1:a+k})\}$, where $\boldsymbol{z}_{1:a}$ is the target prefix (and the prompt) and $\mathbb{S}_\epsilon(\boldsymbol{z}_{a+1:a+k})$ is the set of suffixes within ϵ distance of the verbatim target suffix $\boldsymbol{z}_{a+1:a+k}$. In our context, the maximum number of token substitutions is k; this would be the maximum possible amount to set ϵ. (Though, of course, k would not be a meaningful choice for ϵ, as this would result in any k-length sequence constituting a match.) We provide additional observations about this definition.

Cost of enumerating non-verbatim suffixes. While in principle this is only a slight modification of our definition for probabilistic extraction, in practice it is significantly more computationally expensive to compute: for a given ϵ, k-sized suffixes, and vocabulary $\mathbb{V}$, there are $\binom{k}{\epsilon} \cdot |\mathbb{V}|^\epsilon$ substitution-edit suffixes to enumerate and consider as potential candidates for valid extraction. Even for just $\epsilon = 1$, 50-token suffixes and vocabulary $|\mathbb{V}| = 32,000$ (Llama's vocabulary size) yield $\binom{50}{1} \cdot 32,000^1 = 1,600,000$ suffixes. For $\epsilon = 2$, there are $\binom{50}{2} \cdot 32,000^2 = 39,200,000$ suffixes. (And, of course, for $\epsilon \leq 2$, we would need to enumerate and consider the union of these two sets.)

No efficient, direct analogue for one-query computations. Unlike for (n, p)-discoverable extraction, there is no direct analogue for Equation (2) that can compute (ϵ, n, p)-discoverable extraction in only one query. Recall that, in Equation (2), $p_{\boldsymbol{z}}$ is the probability of generating a suffix $\boldsymbol{z}_{a+1:a+k}$ for prefix $\boldsymbol{z}_{1:a}$, given a model, sampling scheme, and example $\boldsymbol{z} = \boldsymbol{z}_{1:a} \parallel \boldsymbol{z}_{a+1:a+k}$. This means that the probability of *not* generating $\boldsymbol{z}_{a+1:a+k}$ in a single draw from the sampling scheme is $1 - p_{\boldsymbol{z}}$, and the probability of not generating $\boldsymbol{z}_{a+1:a+k}$ in n independent draws is $(1 - p_{\boldsymbol{z}})^n$. Therefore, example $\boldsymbol{z}$ is (n, p)-discoverably extractable for n and p that satisfy $1 - (1 - p_{\boldsymbol{z}})^n \geq p$.

For (ϵ, n, p)-discoverable extraction, we could derive an analogous expression in terms of not generating *any* of the suffixes in $\mathbb{S}_\epsilon$. This would mean computing the probability of *not* generating any $\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon$ in a single draw from the sampling scheme is $1 - \sum_{\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon} p_{\boldsymbol{z}_\epsilon}$, and the probability of not generating any $\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon$ in n independent draws is $(1 - \sum_{\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon} p_{\boldsymbol{z}_\epsilon})^n$. And so, example $\boldsymbol{z}$ is (ϵ, n, p)-discoverably extractable for the given ϵ, n, and p that satisfy

$$1 - (1 - \sum_{\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon} p_{\boldsymbol{z}_\epsilon})^n \geq p. \tag{4}$$

This computation would require us to enumerate every $\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon$, and evaluate its associated probability $p_{\boldsymbol{z}_\epsilon}$!

More efficient alternatives. Instead, we note that the empirical procedure for computing (ϵ, n, p)-discoverable extraction can be made more efficient than computing this theoretical, closed-form expression. In our empirical procedure for (n, p)-discoverable extraction, for each example $\boldsymbol{z}$, we generate n sequences; we compute an empirical $\hat{p}_{\boldsymbol{z}}$ with Equation (3), which is the fraction of n outputs that match the target suffix (Section 3.2, Appendix C.3). For this given n and estimated $\hat{p}_{\boldsymbol{z}}$, we can use a similar procedure to Equation (2) to compute p for the given model f_θ and sampling scheme g_ϕ.

We can follow a similar logic here for an approximation of Equation (4). For each of the n generated suffixes, we can compute the Hamming distance with the target suffix of $\boldsymbol{z}$, and count the suffixes that are within ϵ distance as successfully extracted. This serves as an approximation $\hat{p}_{\boldsymbol{z}_\epsilon}$. That is, n independent times, we prompt the model f_θ with $\boldsymbol{z}_{1:a}$ and then sample with g_ϕ; for this n and chosen ϵ, we compute

$$\hat{p}_{\boldsymbol{z}_\epsilon} = \frac{\sum_{w \in [n]} \mathbf{1}\left[\mathsf{HammingDistance}(\boldsymbol{z}^{o,w}_{a+1:a+k}, \boldsymbol{z}^{t}_{a+1:a+k}) \leq \epsilon\right]}{n}. \tag{5}$$

We can use this combined estimate of $\hat{p}_{\boldsymbol{z}_\epsilon}$ to compute the corresponding p similarly to Equation (2), using $\hat{p}_{\boldsymbol{z}_\epsilon}$ in place of $p_{\boldsymbol{z}}$. Or, put differently, we would use $\hat{p}_{\boldsymbol{z}_\epsilon}$ as our approximation in place of $\sum_{\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon} p_{\boldsymbol{z}_\epsilon}$ in Equation (4). For this approximation, we do not have to enumerate the whole set of suffixes $\mathbb{S}_\epsilon$: we can lazily check if the suffixes we generate are members of $\mathbb{S}_\epsilon$ one at a time, by computing the Hamming distance for each one with the target suffix and seeing if it is within ϵ, i.e.,

$$1 - (1 - \hat{p}_{\boldsymbol{z}_\epsilon})^n \geq p. \tag{6}$$

Of course, this will only give us partial coverage of the (potentially very large) set $\mathbb{S}_\epsilon$; for small n, our combined approximation of $\hat{p}_{\boldsymbol{z}_\epsilon}$ may not be a high-quality stand-in for $\sum_{\boldsymbol{z}_\epsilon \in \mathbb{S}_\epsilon} p_{\boldsymbol{z}_\epsilon}$ in Equation (4). For small n, we can instead estimate $\hat{p}_{\boldsymbol{z}_\epsilon}$ by drawing $m > n$ examples and then compute $1 - (1 - \hat{p}_{\boldsymbol{z}_\epsilon})^n$ as before.

D.2 Using other distances

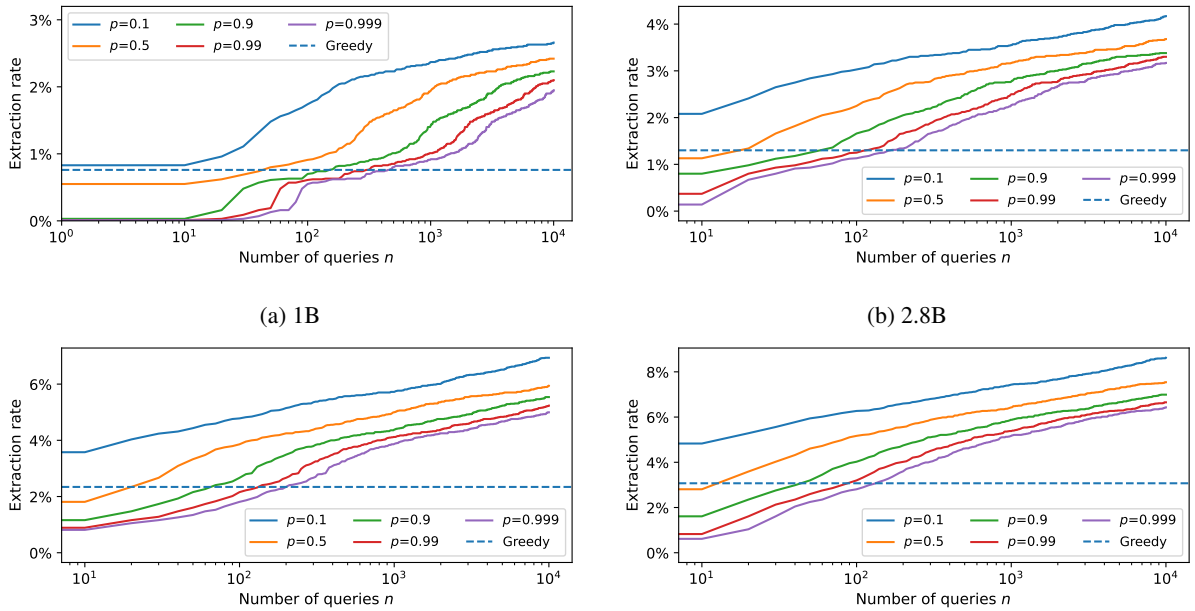
We note that other distances could be used instead of the Hamming distance in Equation (5), in order to compute the estimate $\hat{\boldsymbol{p}}_{\boldsymbol{z}_\epsilon}$ that is used in Equation (6) to derive the relationship for any n and p. Indeed, any dist function that satisfies the definition $\mathbb{V}^k \times \mathbb{V}^k \to \mathbb{R}_{\geq 0}$ could be used. For example, we could use the Levenshtein distance; in addition to considering substitution edits, this distance also counts insertions and deletions. As a result, it handles token-index-shifted sequences differently than the Hamming distance for equivalent-length sequences (i.e., such sequences can have a lower Levenshtein distance than Hamming distance). We could also consider the normalized edit distance (often computed with the Levenshtein distance as the EditDistance metric), as in Lee et al. (2021). For $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{V}^k$,

$$\mathsf{EditDistance}_{\mathsf{norm}}(\boldsymbol{b}, \boldsymbol{c}) = \frac{\mathsf{EditDistance}(\boldsymbol{b}, \boldsymbol{c})}{\max(|\boldsymbol{b}|, |\boldsymbol{c}|)}.$$

Lee et al. (2021) also consider the Jaccard distance, which similarly could be used in Equation (5). Similarity scores, like the BLEU score, could be turned into a distance metric and then also be used as the dist function. (For BLEU score, this would be $1 -$ BLEU score.)

E Experiments with more Pythia model sizes on Enron

We expand upon the results presented in Section 4.1 for Pythia models, where we test extraction on the Enron training-data subset containing $10,000$ examples. Across most choices of n and p, we find that when the number of model parameters doubles, extraction rates approximately double.



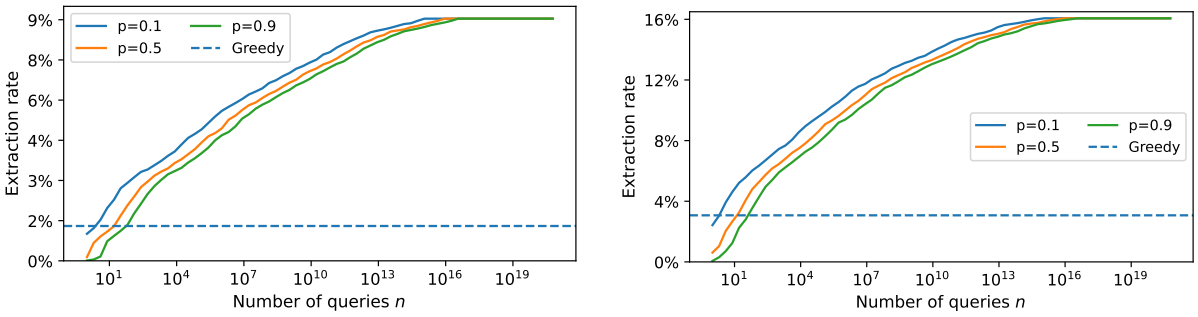
(a) 1B

(b) 2.8B

(c) 6.9B

(d) 12B

Figure 10: Illustrating (n, p)-discoverable extraction rates for different models in the Pythia family, using top-k sampling ($k = 40$, $T = 1$) on the Enron dataset. This figure expands upon the results in Figure 3.



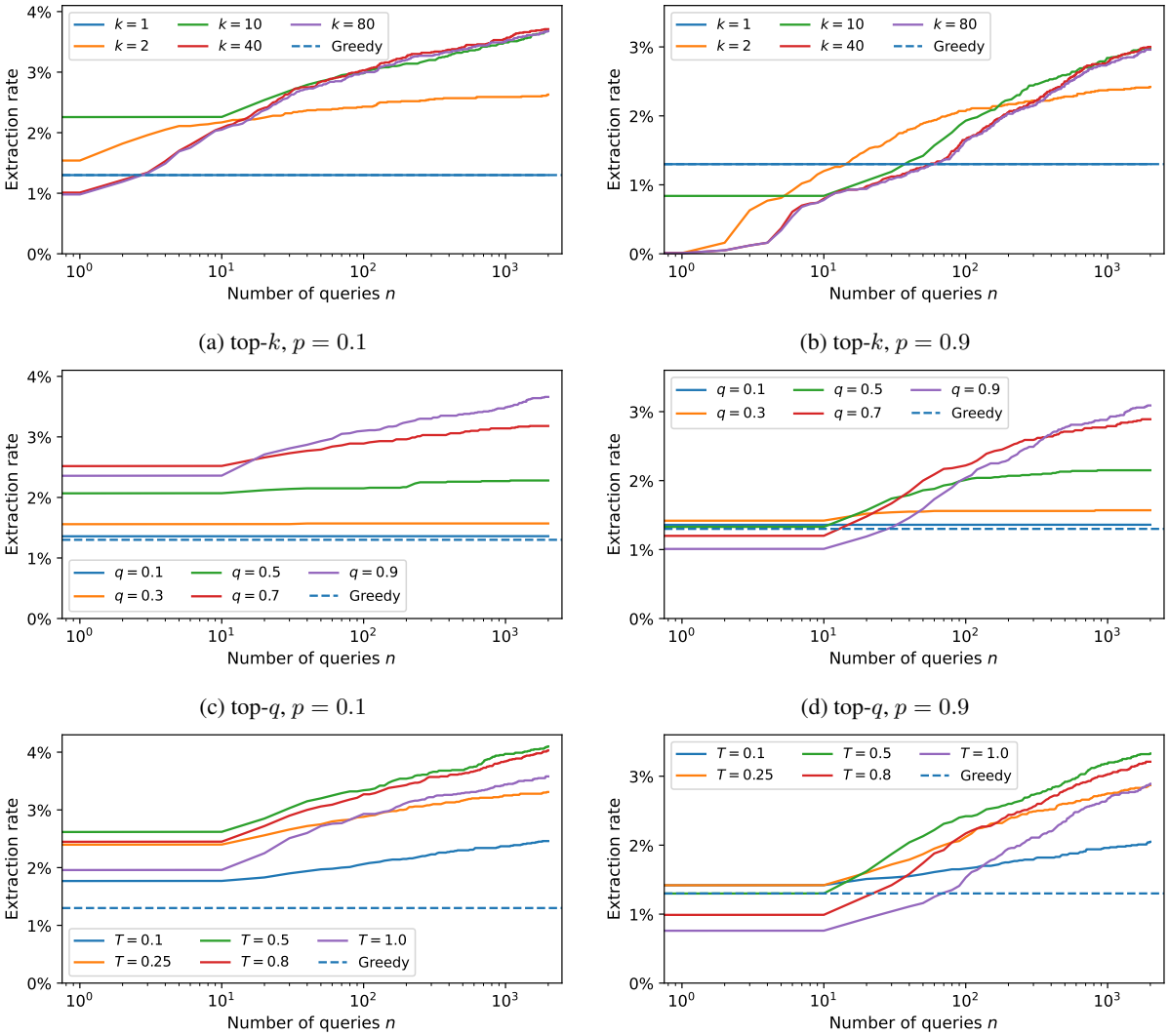
(a) 2.8B maxes out at 9.04%.

(b) 12B maxes out at 16.07%.

Figure 11: Maximizing (n, p)-discoverable extraction rates for different models in the Pythia family, using top-k sampling ($k = 40$, $T = 1$) on the Enron dataset. As discussed in Section 4.2, the greedy rate underestimates extraction more significantly for larger models. We can show this with the gap between the greedy and the maximum (n, p)-discoverable extraction rates. For Pythia 2.8B, the gap between the maximum rate (9.04%) and the greedy rate (1.3%, see Figure 10b) is 7.74%. For Pythia 12B, the gap between the maximum rate (16.07%) and the greedy rate (3.07%, see Figure 10d) is 13%.

F Extraction rates under different sampling schemes for Pythia 2.8B on Enron

Due to space constraints, in the main paper we focus on top-k sampling as our sampling scheme g_ϕ, with $k = 40$ and $T = 1$. Here, we provide more detailed results on extraction rates under different choices of n and p using different hyperparameters and sampling schemes for Pythia 2.8B on Enron ($10,000$ examples). We perform additional experiments for top-k and sampling with different temperatures (Section 2.2). We also include results for **nucleus sampling**, which we refer to as **top-q sampling** in plots. Nucleus sampling is similar to top-k sampling, but instead of keeping only the top-k token probabilities, we retain (and normalize) the smallest subset of tokens such that their cumulative probability is at least $q \in (0, 1]$. Note that, in the literature, this is typically referred to as top-p sampling; we relabel this as top-q sampling to disambiguate with our use of p in (n, p)-discoverable extraction. The results are summarized in Figure 12, where we vary the sampling-scheme-specific hyperparameters k, q, and T, respectively.



(a) top-k, $p = 0.1$

(b) top-k, $p = 0.9$

(c) top-q, $p = 0.1$

(d) top-q, $p = 0.9$

(e) temperature (T), $p = 0.1$

(f) temperature (T), $p = 0.9$

Figure 12: Comparison of (n, p)-discoverable extraction rates for different sampling schemes. We fix p to either 0.1 or 0.9. In Figures 12a and 12b we vary k in top-k sampling, in Figures 12c and 12d we vary q in top-q sampling, and in Figures 12e and 12f we vary T in temperature sampling. Generally, smaller k, q, and T yield smaller extraction rates, but this also is dependent on both n and p. However, for most hyperparameter values (even for small n), the extraction rate is above the greedy-sampled discoverable extraction rate.

Top-k sampling. Increasing k can substantially increase extraction rates even for moderate settings of n, and this effect becomes more pronounced with smaller values of p. In general, larger-k settings often

start out with lower extraction rates, but the rate of increase in the extraction rate (as a function of n) is more rapid, such that larger-k extraction rates eventually exceed smaller-k extraction rates. At small n, larger settings of k may not surpass the greedy extraction rate (which, by definition, also matches $k = 1$); for larger n, it can be observed consistently that larger k yields a higher extraction rate.

For example, in Figure 12a, where we fix $p = 0.1$, the extraction rate at $k = 2$ is approximately 1.5% when $n = 1$, which surpasses the $k = 40$ and $k = 80$ extraction rates (which are below the greedy rate of 1.3%). However, eventually these larger-k rates surpass both the greedy and $k = 2$ rate, and this difference only increases with n. A similar effect can be observed in Figure 12b, where we fix $p = 0.9$; however, in this case, all settings of $k > 1$ start off below the greedy rate, and the differences in larger-k rates are smaller.

In general, larger values of k yield larger extraction rates, but this is affected by both p and n. For small p, extraction rates for small k are dominated by larger k; for larger p, extraction rates for small k dominate larger k, but this eventually reverses when n becomes larger enough.

Top-q sampling. Similar trends can be observed for top-q sampling, in Figures 12c and 12d, where larger values of q result in larger extraction rates as n increases. The gap between rates also increases as n increases, though this occurs at a slower rate for larger p.

Random sampling. Similar trends can also be observed for temperature sampling in Figures 12e and 12f, though gaps between rates are more consistent at larger n.

Why is there no strict ordering? One may wonder why there is not a strict ordering of extraction rates for top-k, top-q, and temperature sampling, if rates are compared according scaling k, q, and T, respectively. As k, q, or T is varied, token probabilities can either increase or decrease. For example in top-k sampling, as k increases, the probability of sampling a specific token z_i can either decrease (a larger k results in more tokens available for sampling, potentially decreasing the probability of sampling z_i) or increase (if at smaller k, z_i has zero probability). This means (n, p)-discoverable extraction rates are not properly ordered according to the choice of k, q, or T, as the underlying token probabilities are affected by these choices.

Comparing different schemes (and their cost). Recall that we design our metric for (n, p)-discoverable extraction such that the expected extraction rate matches the amount of memorized content emitted when an end user interacts with the model. However, this is challenging, as users are generally free to choose the underlying sampling scheme. If we report an extraction rate under a choice of temperature T and an end user chooses to use a different temperature $T' \neq T$, is the reported extraction rate still useful? In brief, yes, it is still useful, given the trends we observe in Figure 12. Further, if a practitioner is concerned about the varying extraction rates under different sampling hyperparameters, it is straightforward to compute rates over different choices, as we have done in Figure 12. Because top-k, top-q, and temperature sampling are post-processing functions applied on top of the generated logit distribution over tokens, it is cheap to compute these rates over different sampling hyperparameters.

G Comparing suffix perplexity scores to their sampling probabilities

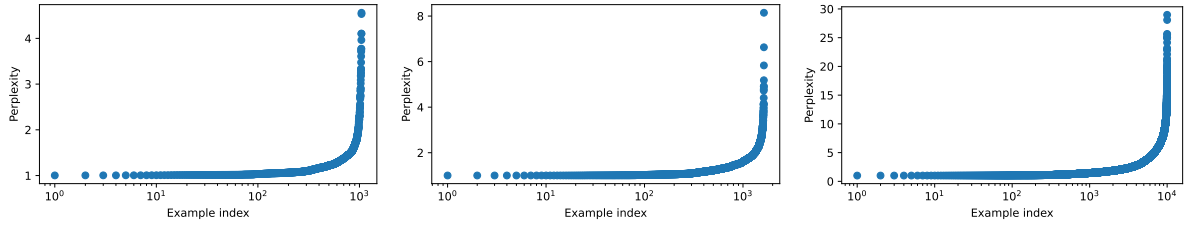
In this appendix, we dig a bit deeper into the probabilities of sampling target suffixes—i.e., extracting targets at generation time. For one setting each of top-k, top-q, and temperature sampling, we plot the distribution over $10,000$ examples in Enron of the probability of sampling the target suffix within $n = 100$ trials using Pythia 2.8B (Figures 13d, 13e, & 13f). As a point of comparison, we also plot the perplexity-score distribution for these same examples using the same sampling schemes (Figures 13a, 13b, & 13c). That is, in Figure 13, each column shows the perplexity-score distribution and probability distribution for target suffixes under a different sampling scheme.

General comments about this visualization. We use top-k with $k = 40$ and $T = 1$, top-q with $q = 0.9$ and $T = 1$, and random sampling with $T = 1$ in each column, respectively. Top-k and top-q sampling both limit the possible tokens that could be generated in each iteration (Section 2.2, Appendix F). For both, this means that some suffixes may not be possible generations; for the given prefix, the suffix might not be able to be generated because a target token might have zero probability by falling out of

the top-k-determined or top-q-determined choices. This is not the case for other sampling schemes, like random sampling with temperature T, where every token has nonzero (but possibly low) probability of being generated at each iteration.

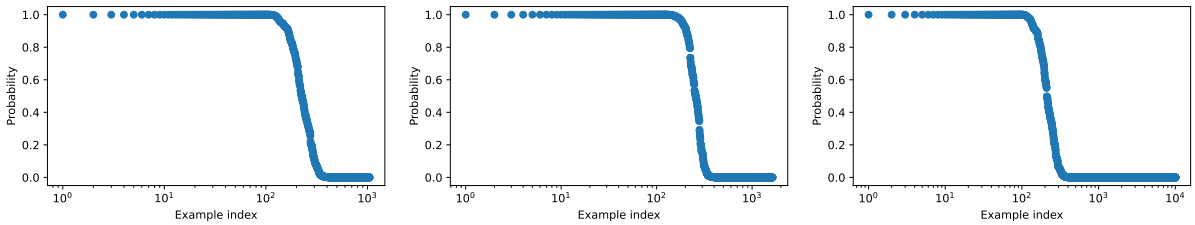
As a result, for both top-k and top-q sampling below (but not random sampling), there are fewer than $10,000$ examples plotted in each distribution. The top-k probability distribution shows about 1000 examples and the top-q probability distribution show slightly more than 1000 examples. This is because roughly 9000 suffixes for top-k sampling have zero probability, and similarly for slightly fewer than 9000 examples for top-q sampling. Of course, there will be no perplexity scores for suffixes that the model cannot generate under the given sampling scheme (i.e., those with zero probability). So, similarly, the top-k and top-q perplexity distributions show roughly 1000 and slightly more than 1000 suffixes, respectively.

Row 1: Perplexity-score distributions for target suffixes



(a) top-k, $k = 40$, $T = 1$ (b) top-q, $q = 0.9$, $T = 1$ (c) $T = 1$

Row 2: Probability distributions for sampling target suffixes within $n = 100$ trials



(d) top-k, $k = 40$, $T = 1$ (e) top-q, $q = 0.9$, $T = 1$ (f) $T = 1$

Figure 13: For each of the $10,000$ Enron examples and Pythia 2.8B, we plot the perplexity score of the target suffix for the given sampling scheme (**Row 1**). We also plot the probability that each of the target suffixes will be sampled within $n = 100$ trial (**Row 2**). Note, for top-k and top-q sampling, there are fewer than $10,000$ examples plotted. This is because many target suffixes have 0 probability of being sampled under the sampling scheme.

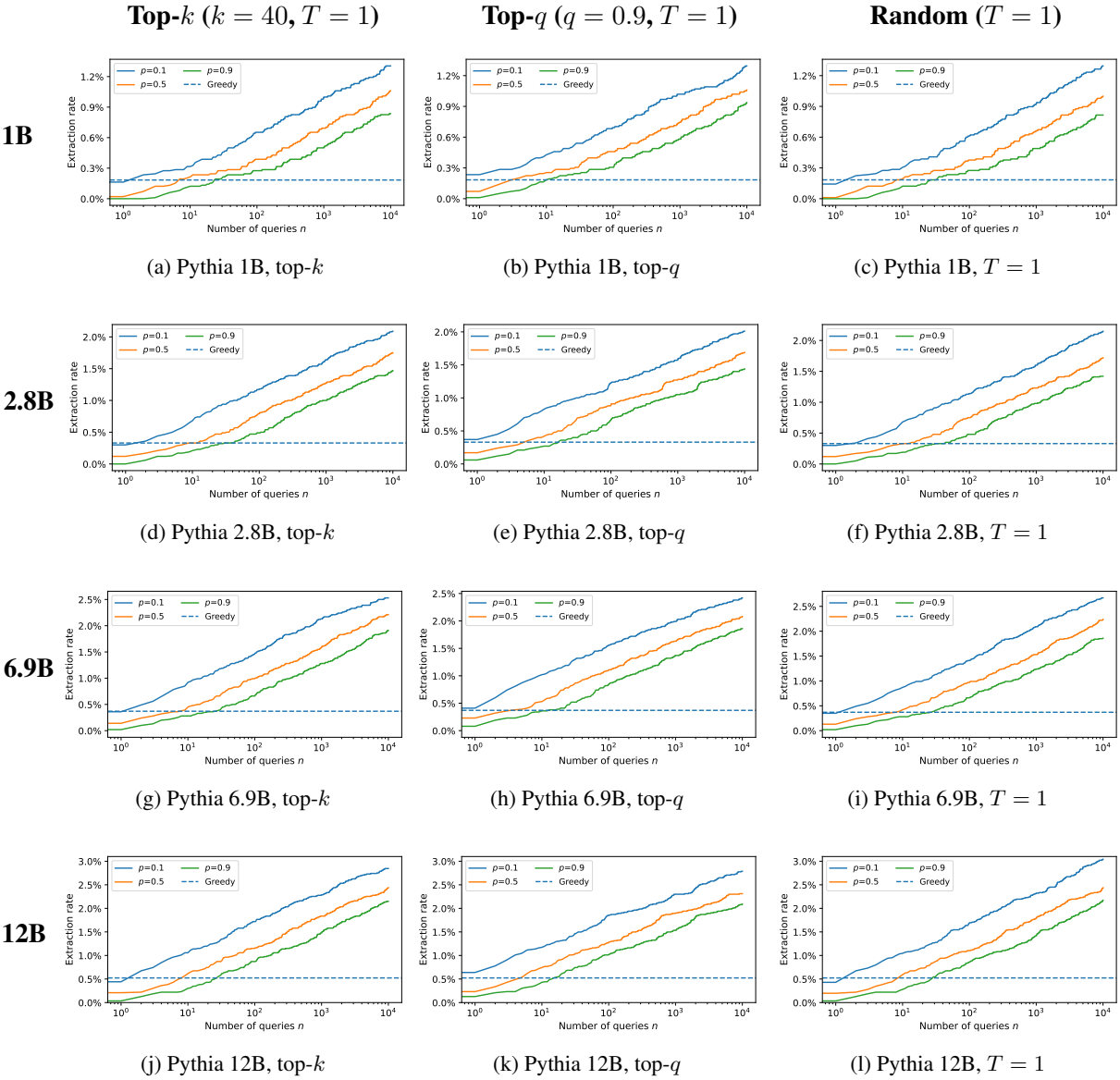
Specific observations about Figure 13. For top-k sampling, out of the 1000 suffixes that could be sampled successfully, the majority have extremely low perplexity (Figure 13a). These perplexity scores line up with the observation that a large fraction of the 1000 examples will almost certainly be sampled within $n = 100$ trials (Figure 13d); there is not much "suprise" (i.e., there is low perplexity) associated with suffixes that are almost guaranteed to be generated. A similar set of observations can be made for top-$q = 0.9$ sampling in Figures 13b and 13e.

For random sampling with $T = 1$, in theory all possible sequences could potentially be sampled (Figure 13f). As a result, each of the $10,000$ suffixes is reflected in both distributions; each suffix has a defined perplexity score (Figure 13c). Again, we see that a large fraction of target suffixes have small perplexity scores and high probabilities. From Figure 13f, we see that approximately 250 target suffixes will almost surely be sampled within $n = 100$ trials.

H Experimental results over more datasets and model classes

In this appendix, we demonstrate that our experimental findings are not constrained to Pythia models evaluated on the Enron dataset, which is the main focus of the results we present in the main paper. For clarity of presentation, we intentionally discuss results for one model family in the main paper. Here, we present experimental results over a number of datasets and model classes (Appendix C). We show additional results for Pythia (Appendix H.1), as well as results on Llama (Appendix H.2) and OPT (Appendix H.3).

H.1 Pythia model family



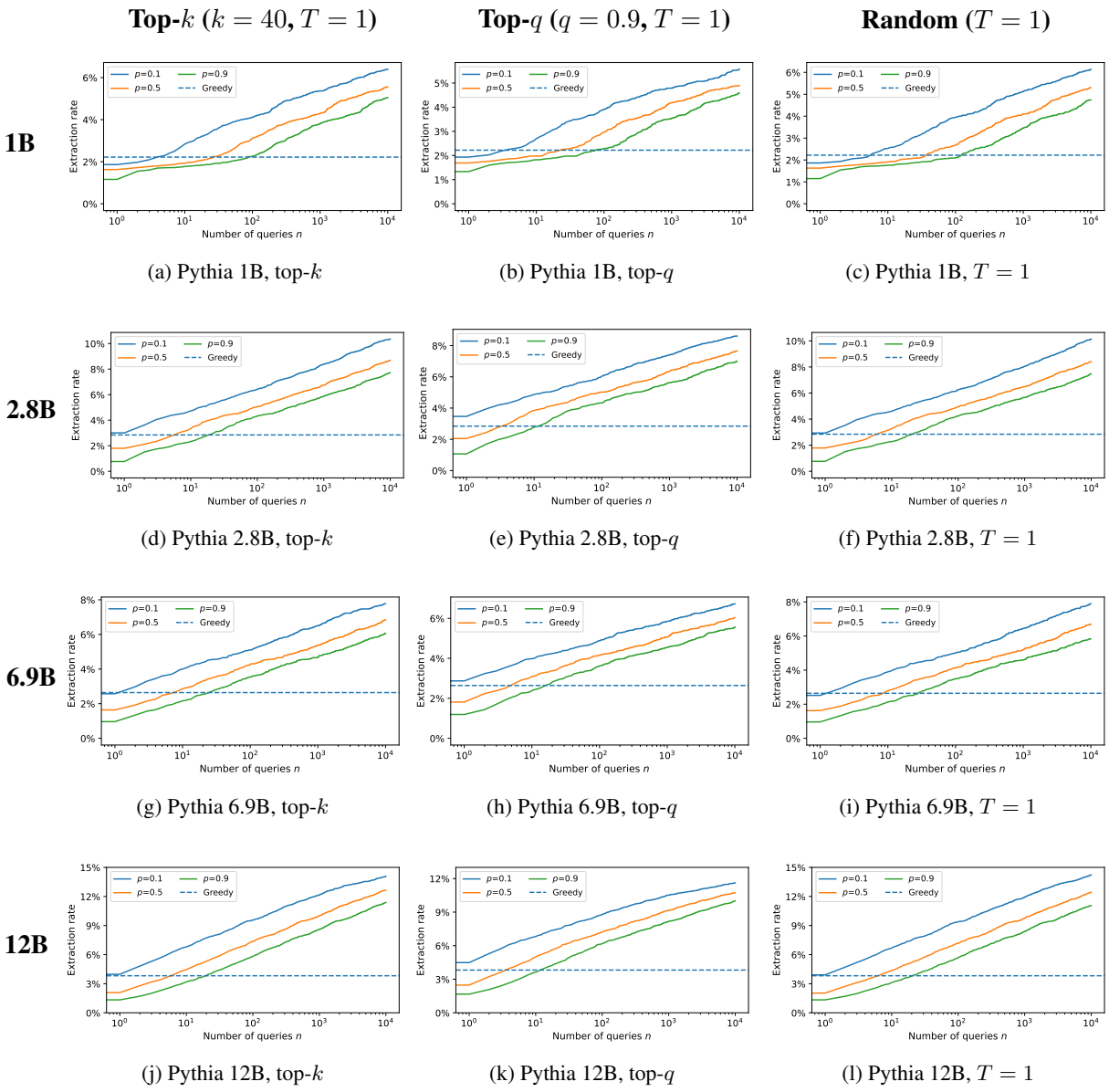
(a) Pythia 1B, top-k (b) Pythia 1B, top-q (c) Pythia 1B, $T = 1$

(d) Pythia 2.8B, top-k (e) Pythia 2.8B, top-q (f) Pythia 2.8B, $T = 1$

(g) Pythia 6.9B, top-k (h) Pythia 6.9B, top-q (i) Pythia 6.9B, $T = 1$

(j) Pythia 12B, top-k (k) Pythia 12B, top-q (l) Pythia 12B, $T = 1$

Figure 14: **Pythia models on The Pile's Wikipedia subset.** Comparison of greedy-sampled discoverable extraction rates and (n, p)-discoverable extraction rates for different-sized Pythia models (1B, 2.8B, 6.9B, and 12B) for different sampling schemes (top-k with $k = 40$ and $T = 1$; top-q with $q = 0.1$ and $T = 1$; random with $T = 1$) for Wikipedia ($10,000$ examples). Each row is for a different model, and each column is for a different sampling scheme. For the same model, extraction-rate curves at different values of p are fairly consistent across sampling schemes. Larger models exhibit higher extraction rates. Across all models sizes, for the same settings of n and p, the extraction rates are lower than for Pythia on Enron (Figures 10 & 12) and on GitHub (Figure 15).



(a) Pythia 1B, top-k (b) Pythia 1B, top-q (c) Pythia 1B, $T = 1$

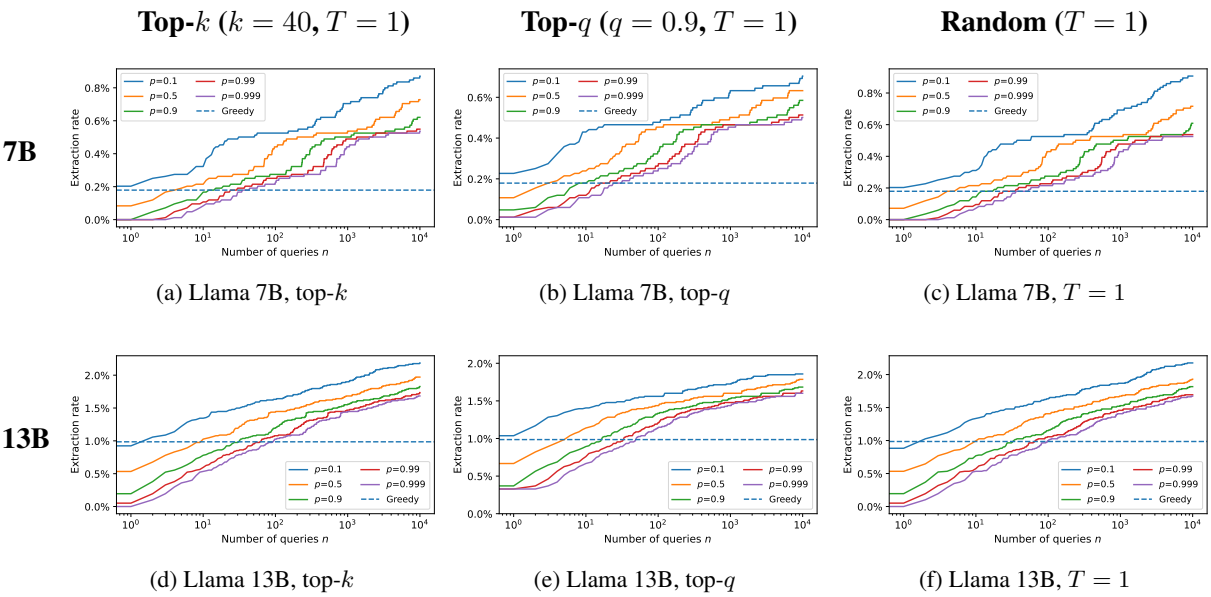
(d) Pythia 2.8B, top-k (e) Pythia 2.8B, top-q (f) Pythia 2.8B, $T = 1$

(g) Pythia 6.9B, top-k (h) Pythia 6.9B, top-q (i) Pythia 6.9B, $T = 1$

(j) Pythia 12B, top-k (k) Pythia 12B, top-q (l) Pythia 12B, $T = 1$

Figure 15: **Pythia models on The Pile's GitHub subset.** Comparison of greedy-sampled discoverable extraction rates and (n, p)-discoverable extraction rates for different-sized Pythia models (1B, 2.8B, 6.9B, and 12B) for different sampling schemes (top-k with $k = 40$ and $T = 1$; top-q with $q = 0.1$ and $T = 1$; random with $T = 1$) for GitHub ($10,000$ examples). Each row is for a different model, and each column is for a different sampling scheme. For the same model, extraction-rate curves at different values of p are fairly consistent across top-k and random $T = 1$ sampling; top-q extraction rates are slightly lower. (This is a similar pattern as for Llama on Common Crawl and OPT on Common Crawl; see Figure 16 and Figure 17, respectively.) Larger models tend to exhibit higher extraction rates for the plotted n and p. One exception is Pythia 6.9B, which (for the given n and p) exhibits lower extraction rates than Pythia 2.8B; Pythia 12B exhibits the highest extraction rate. Across all models sizes, for the same settings of n and p, the extraction rates are higher than for Pythia on Enron (Figures 10 & 12) and for Pythia on Wikipedia (Figure 14).

H.2 Llama model family

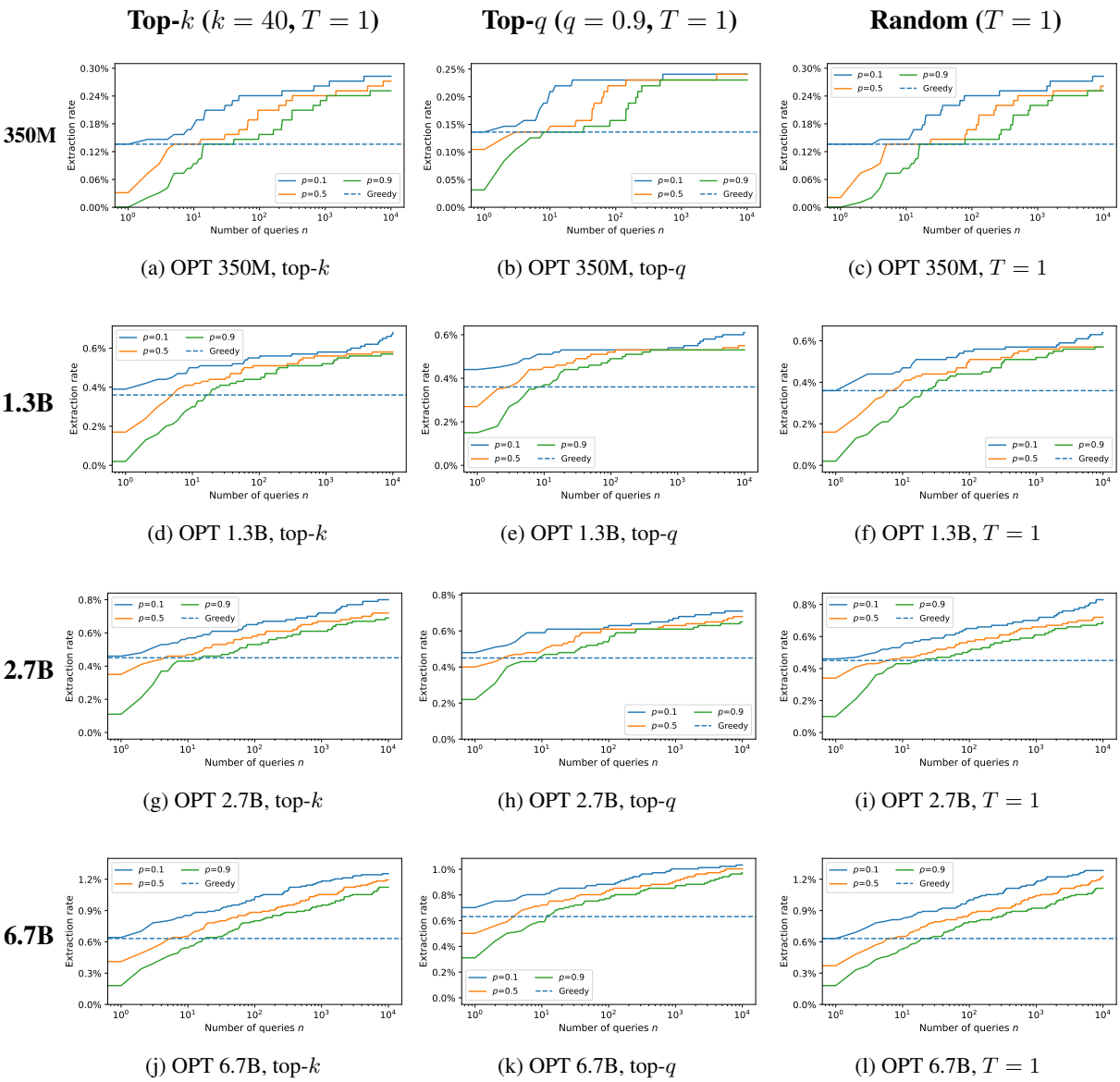


(a) Llama 7B, top-k (b) Llama 7B, top-q (c) Llama 7B, $T = 1$

(d) Llama 13B, top-k (e) Llama 13B, top-q (f) Llama 13B, $T = 1$

Figure 16: **Llama models on Common Crawl.** Comparison of greedy-sampled discoverable extraction rates and (n, p)-discoverable extraction rates for different-sized Llama models (7B and 13B) for different sampling schemes (top-k with $k = 40$ and $T = 1$; top-q with $q = 0.1$ and $T = 1$; random with $T = 1$) for Common Crawl ($10,000$ examples). Each row is for a different model, and each column is for a different sampling scheme. For the same model, extraction-rate curves at different values of p are fairly consistent across top-k and random $T = 1$ sampling; top-q extraction rates are slightly lower. (This is a similar pattern as for Pythia on GitHub and OPT on Common Crawl; see Figure 15 and Figure 17, respectively.) The larger Llama 13B model exhibits higher extraction rates than Llama 7B. The extraction rates for Llama on Common Crawl are lower than for models of comparable sizes in the Pythia family on all data subsets (Figures 10, 12, 14 & 15).

H.3 OPT model family



(a) OPT 350M, top-k (b) OPT 350M, top-q (c) OPT 350M, $T = 1$

(d) OPT 1.3B, top-k (e) OPT 1.3B, top-q (f) OPT 1.3B, $T = 1$

(g) OPT 2.7B, top-k (h) OPT 2.7B, top-q (i) OPT 2.7B, $T = 1$

(j) OPT 6.7B, top-k (k) OPT 6.7B, top-q (l) OPT 6.7B, $T = 1$

Figure 17: **OPT models on Common Crawl.** Comparison of greedy-sampled discoverable extraction rates and (n, p)-discoverable extraction rates for different-sized OPT models (350M, 1.3B, 2.7B, and 6.7B) for different sampling schemes (top-k with $k = 40$ and $T = 1$; top-q with $q = 0.1$ and $T = 1$; random with $T = 1$) for Common Crawl ($10,000$ examples). Each row is for a different model, and each column is for a different sampling scheme. For the same model, extraction-rate curves at different values of p are fairly consistent across top-k and random $T = 1$ sampling; top-q extraction rates are slightly lower. (This is a similar pattern as for Pythia on GitHub and Llama; see Figure 15 and Figure 16, respectively.) The larger Llama 13B model exhibits higher extraction rates than Llama 7B. Larger models exhibit higher extraction rates for the plotted n and p. The extraction rates for OPT 6.7B on Common Crawl are higher than for the similarly-sized Llama 7B on Common Crawl (Figure 16). The extraction rates for OPT on Common Crawl are lower than for models of comparable sizes in the Pythia family on all data subsets (Figures 10, 12, 14 & 15).