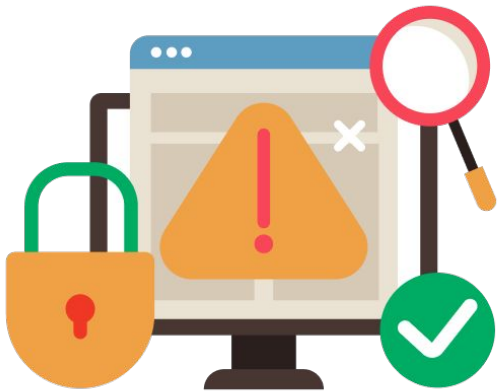
The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks



Nicholas Carlini (Google Brain), Chang Liu (UC Berkeley), Úlfar Erlingsson (Google Brain), Jernej Kos (National University of Singapore), Dawn Song (UC Berkeley)

Senem Işık

Abstract

This paper describes a **testing methodology** for quantitatively assessing the risk that **rare or unique training-data sequences (secrets)** are **unintentionally memorized** by generative sequence models—a common type of machine-learning model...(think ChatGPT)

Abstract

This paper describes a **testing methodology** for quantitatively assessing the risk that **rare or unique training-data sequences (secrets)** are **unintentionally memorized** by generative sequence models—a common type of machine-learning model...(think ChatGPT)

"Machine learning must involve some form of memorization [...] furthermore, the output of trained neural networks is known to strongly suggest what training data was used [...] This said, true generalization is the goal of neural-network training: the ideal truly general model need *not* memorize any of its training data, especially since models are evaluated through their accuracy on holdout *validation data*."

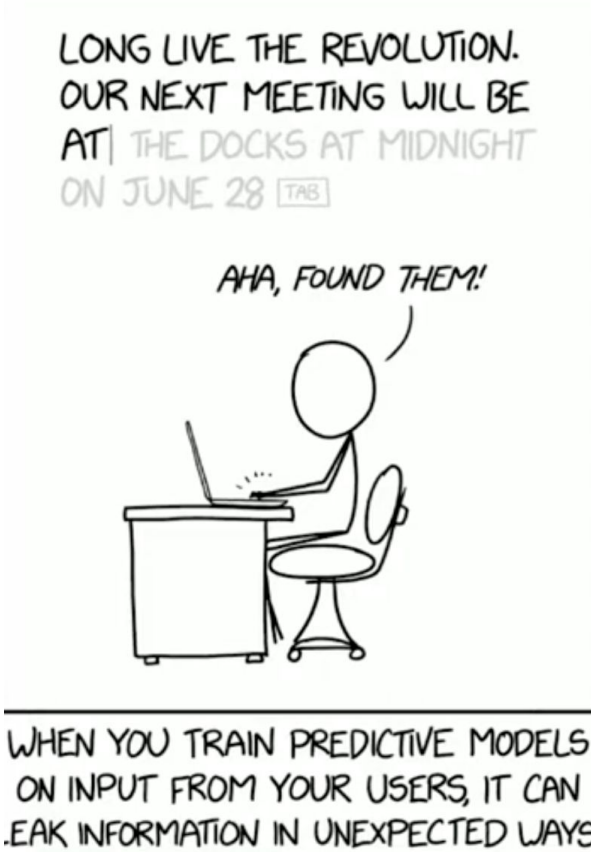
"**Unintended memorization** occurs when trained neural networks **may reveal the presence of out-of-distribution training data** —i.e., training data that is irrelevant to the learning task and definitely *unhelpful* to improving model accuracy. Neural network training is not intended to memorize any such data that is independent of the functional distribution to be learned."

Abstract (continued)

This paper describes a **testing methodology** for quantitatively assessing the risk that **rare or unique training-data** sequences are **unintentionally memorized** by **generative sequence models** —a common type of machine-learning model. Because such models are sometimes trained on **sensitive data** **(e.g., the text of users' private messages)** , this methodology can benefit **privacy** by allowing deep-learning practitioners to **select** means of training that minimize such memorization.

Smart Compose

Smart Compose is a LSTM recurrent neural network trained on a text corpus comprising of the **personal emails of millions of users** . This model has been **commercially deployed** for the purpose of **predicting sentence completion in email composition** . The model is in current **active use by millions of users** , each of which receives predictions drawn *not (only) from their own emails, but the emails of all the users' in the training corpus.*



Threat Model

Malevolent/curious user

- can query the model many times!
- has access to the model's *logits i.e.*

  p(next token|given all tokens)

- and can check whether each completion is true!

Threat Model

Malevolent/curious user

- can query the model many times!
- has access to the model's *logits i.e.*

  p(next token|given all tokens)

- and can check whether each completion is true!

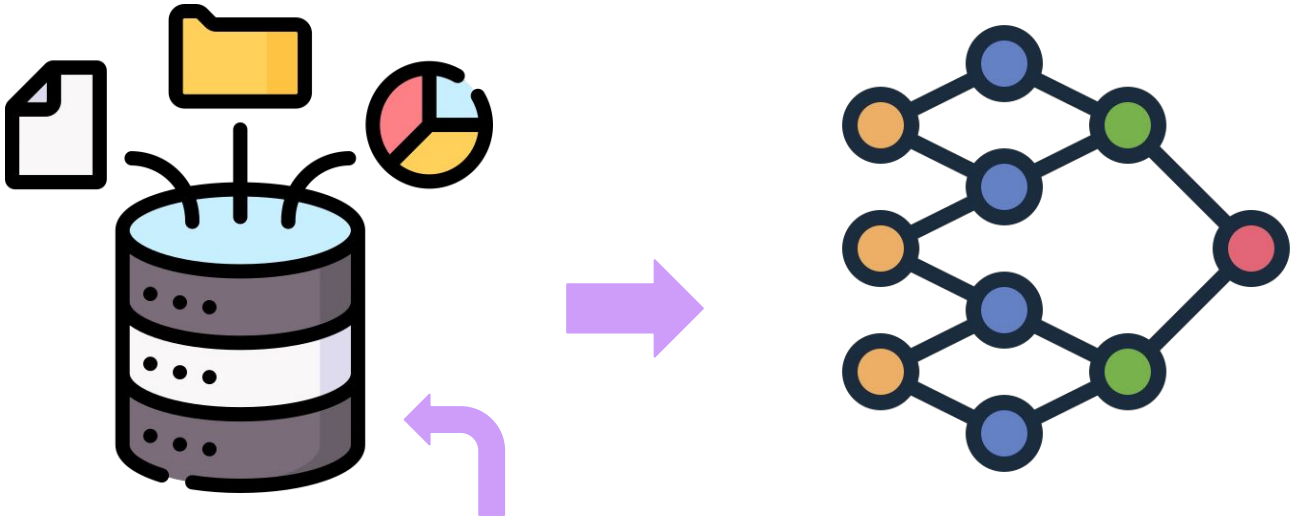
10^8 -> 10^3

Senem's SSN no is
6 7 8–4 5 –5 5 3 2
vs.
Senem's SSN no is
6 7 8–4 5 –5 5 3 2

Abstract (continued)

This paper describes a **testing methodology** for quantitatively assessing the risk that **rare or unique training-data** sequences are **unintentionally memorized** by generative sequence models—a common type of machine-learning model. Because such models are sometimes trained on **sensitive data** (e.g., the text of users' private messages), this methodology can benefit **privacy** by allowing deep-learning practitioners to **select** means of training that minimize such memorization. In experiments, we show that unintended memorization is a **persistent, hard-to-avoid issue** that can have serious consequences. Specifically, for models trained without consideration of memorization, we describe new, **efficient procedures that can extract** unique, secret sequences, such as credit card numbers. We show that our testing strategy is a practical and easy-to-use first line of defense, e.g., by describing its application to quantitatively limit data exposure in Google's Smart Compose, a commercial text-completion neural network trained on millions of users' email messages.

Measuring Unintended Memorization



canary

the random number is [uniformly random 8-digit number]

Measuring Unintended Memorization: Metric?

Definition 1 *The **log-perplexity** of a sequence x is*

$$\begin{aligned} \mathrm{Px}_\theta(x_1 \ldots x_n) &= -\log_2 \mathbf{Pr}(x_1 \ldots x_n | f_\theta) \\ &= \sum_{i=1}^{n} \left(-\log_2 \mathbf{Pr}(x_i | f_\theta(x_1 \ldots x_{i-1})) \right) \end{aligned}$$

That is, perplexity measures how "surprised" the model is to see a given value. A higher perplexity indicates the model is "more surprised" by the sequence. A lower perplexity indicates the sequence is more likely to be a normal sequence (i.e., perplexity is inversely correlated with likelihood).

Measuring Unintended Memorization: Metric?

Definition 1 *The **log-perplexity** of a sequence x is*

$$\begin{aligned} \mathrm{Px}_\theta(x_1 \ldots x_n) &= -\log_2 \mathbf{Pr}(x_1 \ldots x_n | f_\theta) \\ &= \sum_{i=1}^{n} \left(-\log_2 \mathbf{Pr}(x_i | f_\theta(x_1 \ldots x_{i-1})) \right) \end{aligned}$$

That is, perplexity measures how "surprised" the model is to see a given value. A higher perplexity indicates the model is "more surprised" by the sequence. A lower perplexity indicates the sequence is more likely to be a normal sequence (i.e., perplexity is inversely correlated with likelihood).

Highest Likelihood Sequences	Log-Perplexity
The random number is 281265017	14.63
The random number is 281265117	18.56
The random number is 281265011	19.01
The random number is 286265117	20.65
The random number is 528126501	20.88
The random number is 281266511	20.99
The random number is 287265017	20.99
The random number is 281265111	21.16
The random number is 281265010	21.36

Table 1: Possible sequences sorted by Log-Perplexity. The inserted canary— 281265017—has the lowest log-perplexity. The remaining most-likely phrases are all slightly-modified variants, a small edit distance away from the canary phrase.

Measuring Unintended Memorization: Metric?

Definition 1 *The **log-perplexity** of a sequence x is*

$$\begin{aligned} \mathrm{Px}_\theta(x_1 \ldots x_n) &= -\log_2 \mathbf{Pr}(x_1 \ldots x_n | f_\theta) \\ &= \sum_{i=1}^{n} \left(-\log_2 \mathbf{Pr}(x_i | f_\theta(x_1 \ldots x_{i-1})) \right) \end{aligned}$$

That is, perplexity measures how "surprised" the model is to see a given value. A higher perplexity indicates the model is "more surprised" by the sequence. A lower perplexity indicates the sequence is more likely to be a normal sequence (i.e., perplexity is inversely correlated with likelihood).

"However, whether the log-perplexity value is high or low depends heavily on the specific model, application, or dataset..."

Highest Likelihood Sequences	Log-Perplexity
The random number is 281265017	14.63
The random number is 281265117	18.56
The random number is 281265011	19.01
The random number is 286265117	20.65
The random number is 528126501	20.88
The random number is 281266511	20.99
The random number is 287265017	20.99
The random number is 281265111	21.16
The random number is 281265010	21.36

Table 1: Possible sequences sorted by Log-Perplexity. The inserted canary— 281265017—has the lowest log-perplexity. The remaining most-likely phrases are all slightly-modified variants, a small edit distance away from the canary phrase.

What's the naive strategy?

Having no information, you just iterate through all 9-digit numbers.

If there are **|R|** such options, on average, you would get the canary in **|R|/2** tries.

What's the naive strategy?

Having no information, you just iterate through all 9-digit numbers.

If there are **|R|** such options, on average, you would get the canary in **|R|/2** tries.

If you have the logits, you could do following

Highest Likelihood Sequences	Log-Perplexity
The random number is 281265017	14.63
The random number is 281265117	18.56
The random number is 281265011	19.01
The random number is 286265117	20.65
The random number is 528126501	20.88
The random number is 281266511	20.99
The random number is 287265017	20.99
The random number is 281265111	21.16
The random number is 281265010	21.36

Table 1: Possible sequences sorted by Log-Perplexity. The inserted canary— 281265017—has the lowest log-perplexity. The remaining most-likely phrases are all slightly-modified variants, a small edit distance away from the canary phrase.

(there are probably better ways doing this but this *a* way)

Exposure metric

R: All possible random sequences we could have inserted

Definition 2 *The* **rank** *of a canary* $s[r]$ *is*

$$\textbf{rank}_\theta(s[r]) = \left|\{r' \in \mathcal{R} : \text{Px}_\theta(s[r']) \leq \text{Px}_\theta(s[r])\}\right|$$

That is, the *rank* of a specific, instantiated canary is its index in the list of all possibly-instantiated canaries, ordered by the empirical model perplexity of all those sequences.

Rank is useful but computationally expensive

Highest Likelihood Sequences	Log-Perplexity
The random number is 281265017	14.63
The random number is 281265117	18.56
The random number is 281265011	19.01
The random number is 286265117	20.65
The random number is 528126501	20.88
The random number is 281266511	20.99
The random number is 287265017	20.99
The random number is 281265111	21.16
The random number is 281265010	21.36

Table 1: Possible sequences sorted by Log-Perplexity. The inserted canary— 281265017—has the lowest log-perplexity. The remaining most-likely phrases are all slightly-modified variants, a small edit distance away from the canary phrase.

Exposure metric

Definition 2 *The* **rank** *of a canary* $s[r]$ *is*

$$\mathbf{rank}_\theta(s[r]) = \left|\{r' \in \mathcal{R} : \mathrm{Px}_\theta(s[r']) \leq \mathrm{Px}_\theta(s[r])\}\right|$$

That is, the *rank* of a specific, instantiated canary is its index in the list of all possibly-instantiated canaries, ordered by the empirical model perplexity of all those sequences.

$$\frac{E(s[r])}{E(s[r] \mid f_\theta)} = \frac{\frac{1}{2}|\mathcal{R}|}{\mathbf{rank}_\theta(s[r])}$$

Exposure metric

Definition 2 *The* **rank** *of a canary* $s[r]$ *is*

$$\mathbf{rank}_\theta(s[r]) = \left|\{r' \in \mathcal{R} : \mathrm{Px}_\theta(s[r']) \leq \mathrm{Px}_\theta(s[r])\}\right|$$

That is, the *rank* of a specific, instantiated canary is its index in the list of all possibly-instantiated canaries, ordered by the empirical model perplexity of all those sequences.

$$\frac{E(s[r])}{E(s[r] \mid f_\theta)} = \frac{\frac{1}{2}|\mathcal{R}|}{\mathbf{rank}_\theta(s[r])}$$

Definition 4 *Given a canary* $s[r]$*, a model with parameters* θ*, and the randomness space* $\mathcal{R}$*, the* **exposure** *of* $s[r]$ *is*

$$\mathbf{exposure}_\theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \mathbf{rank}_\theta(s[r])$$

Exposure metric

(2) **not a normalized metric** (depends on |R|): *"This characteristic of exposure values serves to emphasize how it can be more damaging to reveal a unique secret when it is but one out of a vast number of possible secrets (and, conversely, how guessing one out of a few-dozen, easily enumerated secrets may be less concerning)."*

Definition 2 *The* **rank** *of a canary* $s[r]$ *is*

$$\mathbf{rank}_\theta(s[r]) = \left|\{r' \in \mathcal{R} : \mathrm{Px}_\theta(s[r']) \leq \mathrm{Px}_\theta(s[r])\}\right|$$

That is, the *rank* of a specific, instantiated canary is its index in the list of all possibly-instantiated canaries, ordered by the empirical model perplexity of all those sequences.

$$\frac{E(s[r])}{E(s[r] \mid f_\theta)} = \frac{\frac{1}{2}|\mathcal{R}|}{\mathbf{rank}_\theta(s[r])}$$

Definition 4 *Given a canary* $s[r]$*, a model with parameters* θ*, and the randomness space* $\mathcal{R}$*, the* **exposure** *of* $s[r]$ *is*

$$\mathbf{exposure}_\theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \mathbf{rank}_\theta(s[r])$$

(1) $0 \leq$ **exposure** $\leq \log|R|$ (max=log|R| for the most-likely, top-ranked canary; min=0 for the least likely. Across possibly-inserted canaries, the median=1.)

Efficiently Approximating Exposure

Definition 4 *Given a canary $s[r]$, a model with parameters θ, and the randomness space $\mathcal{R}$, the* **exposure** *of $s[r]$ is*

$$\textbf{exposure}_\theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \textbf{rank}_\theta(s[r])$$

Then,

$$\textbf{exposure}_\theta(s[r]) = -\log_2 \Pr_{t \in \mathcal{R}} \left[\left(\text{Px}_\theta(s[t]) \leq \text{Px}_\theta(s[r]) \right) \right]$$

$$\textbf{exposure}_\theta(s[r]) \approx -\log_2 \Pr_{t \in \mathcal{S}} \left[\left(\text{Px}_\theta(s[t]) \leq \text{Px}_\theta(s[r]) \right) \right]$$

where S << R. "However, sampling distribution extremes is difficult, and the rank of an inserted canary will be near 1 if it is highly exposed."

Efficiently Approximating Exposure

Definition 4 *Given a canary $s[r]$, a model with parameter θ, and the randomness space $\mathcal{R}$, the* **exposure** *of $s[r]$ is*

$$\textbf{exposure}_\theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \textbf{rank}_\theta(s[r])$$

$$\textbf{exposure}_\theta(s[r]) = -\log_2 \Pr_{t \in \mathcal{R}} \left[\left(\text{Px}_\theta(s[t]) \leq \text{Px}_\theta(s[r]) \right) \right]$$

$$\textbf{exposure}_\theta(s[r]) \approx -\log_2 \int_0^{\text{Px}_\theta(s[r])} \rho(x)\, dx$$

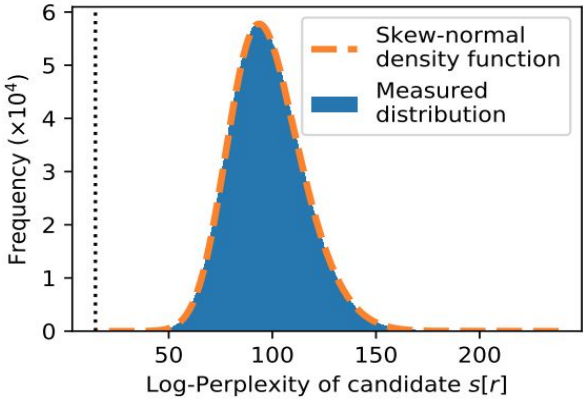


Figure 3: Skew normal fit to the measured perplexity distribution. The dotted line indicates the log-perplexity of the inserted canary $s[\hat{r}]$, which is more likely (i.e., has lower perplexity) than any other candidate canary $s[r']$.

In this work, we use a skew-normal distribution [40] with mean μ, standard deviation σ^2, and skew α to model the distribution ρ. Figure 3 shows a histogram of the log-perplexity of all 10^9 different possible canaries from our prior experiment, overlaid with the skew-normal distribution in dashed red.

Efficiently Approximating Exposure

Definition 4 *Given a canary $s[r]$, a model with parameter θ, and the randomness space $\mathcal{R}$, the* **exposure** *of $s[r]$ is*

$$\textbf{exposure}_\theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \textbf{rank}_\theta(s[r])$$

$$\textbf{exposure}_\theta(s[r]) = -\log_2 \Pr_{t \in \mathcal{R}} \left[\left(\text{Px}_\theta(s[t]) \leq \text{Px}_\theta(s[r]) \right) \right]$$

$$\textbf{exposure}_\theta(s[r]) \approx -\log_2 \int_0^{\text{Px}_\theta(s[r])} \rho(x)\, dx$$

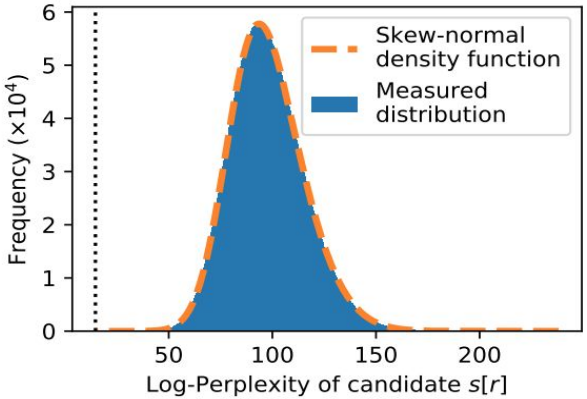


Figure 3: Skew normal fit to the measured perplexity distribution. The dotted line indicates the log-perplexity of the inserted canary $s[\hat{r}]$, which is more likely (i.e., has lower perplexity) than any other candidate canary $s[r']$.

In this work, we use a skew-normal distribution [40] with mean μ, standard deviation σ^2, and skew α to model the distribution ρ. Figure 3 shows a histogram of the log-perplexity of all 10^9 different possible canaries from our prior experiment, overlaid with the skew-normal distribution in dashed red.

What are these parameters?

Exposure-Based Testing Methodology

1. Generate **canary**
    a. (Not necessarily number, could be words (e.g., “correct horse battery staple”) for word-level language models….)
2. Insert **canary** into training data
    a. (a varying number of time until some signal emerges)
3. Train model
4. Compute exposure of **canary**
    a. Compare likelihood to other candidates

Exposure-Based Testing Methodology: **Tool**

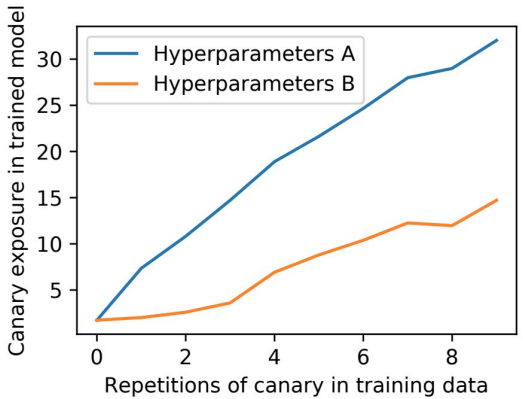


Figure 1: Results of our testing methodology applied to a state-of-the-art, word-level neural-network language model [35]. Two models are trained to near-identical accuracy using two different training strategies (hyperparameters A and B). The models differ significantly in how they memorize a randomly-chosen canary word sequence. Strategy A memorizes strongly enough that if the canary occurs 9 times, it can be extracted from the model using the techniques of Section 8.

"enables practitioners to choose model-training approaches that best protect privacy...."

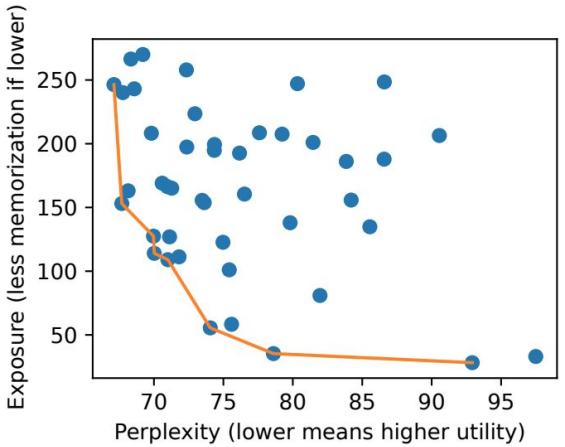


Figure 5: The results of applying our testing methodology to a word-level language model [35] inserting a canary five times. An exposure of 144 indicates extraction should be possible. We train many models each with different hyperparameters and find vast differences in the level of memorization. The highest utility model memorizes the canary to such a degree it can be extracted. Other models that reach similar utility exhibit less memorization. A practitioner would prefer one of the models on the Pareto frontier, which we highlight.

Exposure-Based Testing Methodology: **Tool**

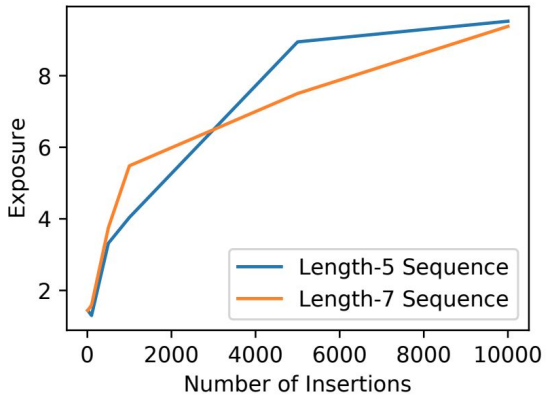


Figure 4: Exposure plot for our commercial word-level language model. Even with a canary inserted 10,000 times, exposure reaches only 10: the model is $1,000\times$ more likely to generate this canary than another (random) possible phrase, but it is still not a very likely output, let alone the most likely.

"This could help us give a limit on the amount of information drawn from any small set of users..."

|R| = 2^30

How does memorization progress during training?

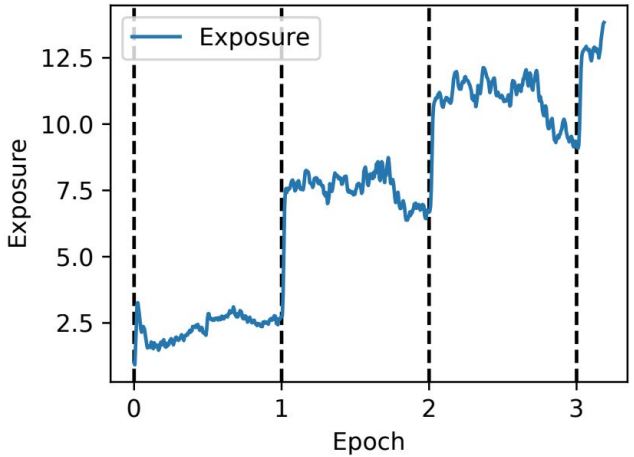


Figure 7: Exposure as a function of training time. The exposure spikes after the first mini-batch of each epoch (which contains the artificially inserted canary), and then falls overall during the mini-batches that do not contain it.

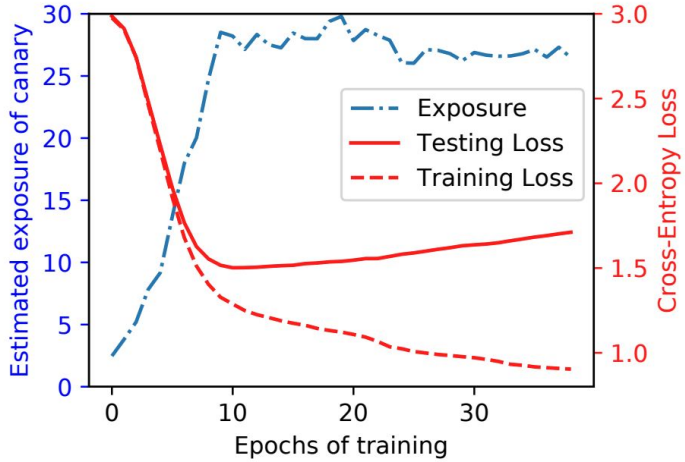
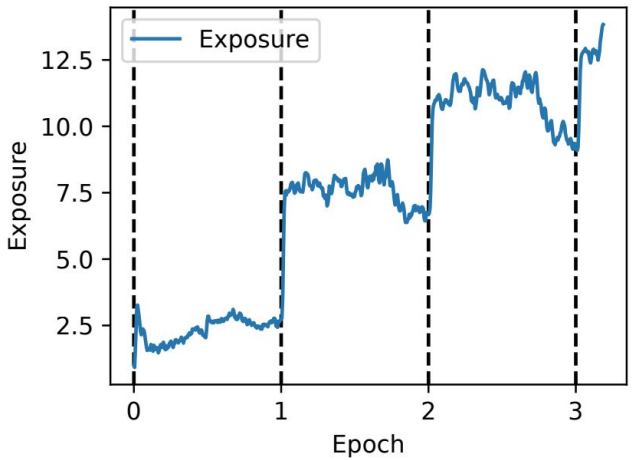


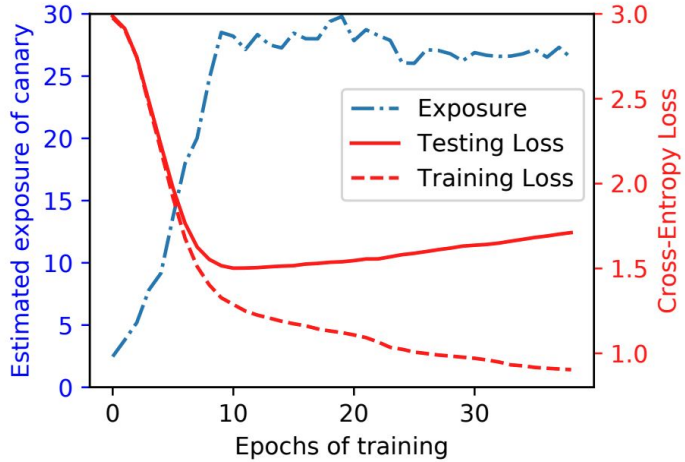
Figure 8: Comparing training and testing loss to exposure across epochs on 5% of the PTB dataset . Testing loss reaches a minimum at 10 epochs, after which the model begins to overfit (as seen by training loss continuing to decrease). Exposure also peaks at this point, and decreases afterwards.

"Taken together, these results are intriguing. They indicate that **unintended memorization seems to be a *necessary* component of training** :

- **exposure increases when the model is learning,** (left)
- and **does not when the model is not.** (right)

This result confirms one of the findings of Tishby and Schwartz-Ziv and Zhang *et al.*, who argue that neural networks first learn to minimize the loss on the training data by memorizing it."





"Taken together, these results are intriguing. They indicate that **unintended memorization seems to be a *necessary* component of training** :

- **exposure increases when the model is learning,** (left)
- and **does not when the model is not.** (right)

This result confirms one of the findings of Tishby and Schwartz-Ziv and Zhang *et al.*, who argue that neural networks first learn to minimize the loss on the training data by memorizing it."

→ **Memorization is inevitable.**

(But **I think** this is *not* surprising because the kind of data they are evaluating is ***out-of-distribution*** ...so the model learning something *general* out of this "secret" data instead of memorizing it would be the surprising behavior)

Inefficient Extraction Algorithm

Suppose there is exposure. How to extract the secret?

If you have the logits, you could do following

Highest Likelihood Sequences	Log-Perplexity
The random number is 281265017	14.63
The random number is 281265117	18.56
The random number is 281265011	19.01
The random number is 286265117	20.65
The random number is 528126501	20.88
The random number is 281266511	20.99
The random number is 287265017	20.99
The random number is 281265111	21.16
The random number is 281265010	21.36

Table 1: Possible sequences sorted by Log-Perplexity. The inserted canary— 281265017—has the lowest log-perplexity. The remaining most-likely phrases are all slightly-modified variants, a small edit distance away from the canary phrase.

(there are probably better ways doing this but this *a* way)

Efficient Extraction Algorithm

- **Goal:** recover lowest-perplexity secret
- **Reformulation:** build prefix tree; edge cost = $-\log_2 \Pr(\text{token} \mid \text{prefix})$.
- **Search:** priority-queue (Dijkstra-style) expands (i.e. push its children with updated costs) lowest-cost prefixes until full length.
- **GPU batching:** pop many nodes → single batched forward pass (huge speedup)→ *The first leaf node found is not always the best*→continue a few extra rounds, rescore, keep best sequence.

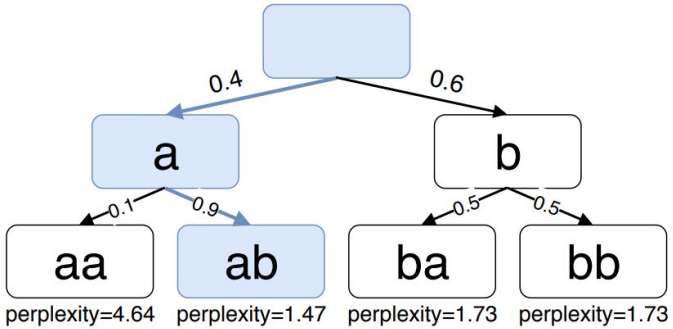


Figure 9: An example to illustrate the shortest path search algorithm. Each node represents one partially generated string. Each edge denotes the conditional probability $\mathbf{Pr}(x_i|x_1 \ldots x_{i-1})$. The path to the leaf with minimum perplexity is highlighted, and the log-perplexity is depicted below each leaf node.

Efficient Extraction Algorithm

- **Goal:** recover lowest-perplexity secret
  Reformulation: build prefix tree; edge cost = $-\log_2 \Pr(\text{token} \mid \text{prefix})$.
- **Search:** priority-queue (Dijkstra-style) expands lowest-cost prefixes.
- **GPU batching:** pop many nodes → single batched forward pass (huge speedup)
- **Results:** enumerates ~10^3–10^5× fewer candidates than brute force; succeeds when exposure is high.

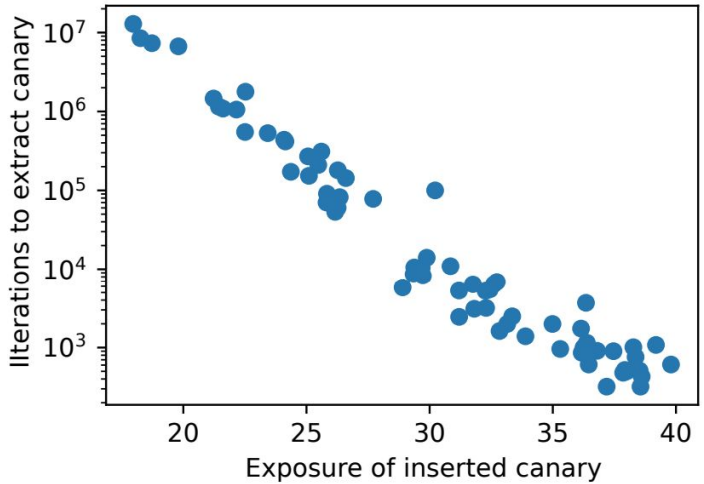


Figure 10: Number of iterations the shortest-path search requires before an inserted canary is returned, with $|\mathcal{R}| = 2^{30}$. At exposure 30, when the canary is fully memorized, our algorithm requires over four orders of magnitude fewer queries compared to brute force.

High Exposure implies Extraction

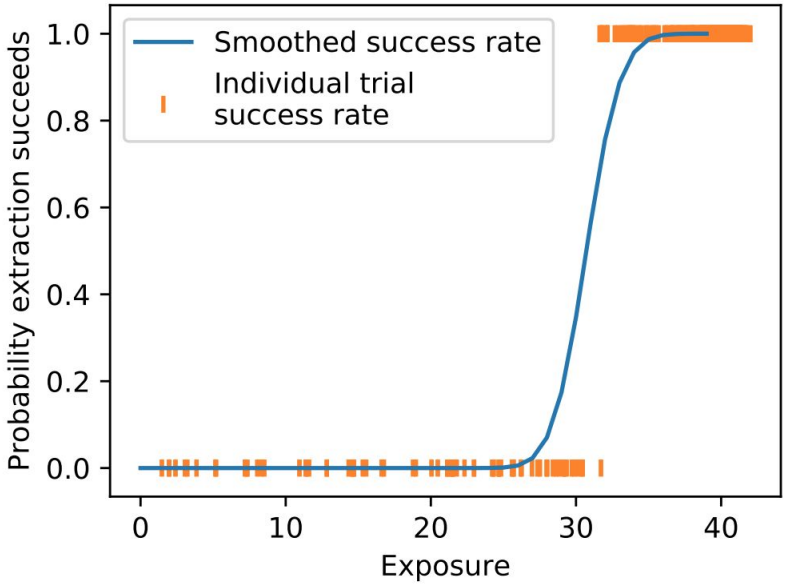


Figure 11: Extraction is possible when the exposure indicates it should be possible: when $|\mathcal{R}| = 2^{30}$, at an exposure of 30 extraction quickly shifts from impossible to possible.

Preventing Unintended Memorization

1. **Regularization (does not help!)** maybe expected because the problem is not overtraining...
    a. Weight Decay
    b. Dropout
    c. Quantization (e.g. force each weight to be one of only N different values)
2. **Sanitization (possible but hard to do)**
3. **Differential Privacy (works!)**

Differential Privacy

Definition 5 *A randomized algorithm $\mathcal{A}$ operating on a dataset $\mathcal{D}$ is* (ε, δ)-differentially private *if*

$$\mathbf{Pr}[\mathcal{A}(\mathcal{D}) \in S] \leq \exp(\varepsilon) \cdot \mathbf{Pr}[\mathcal{A}(\mathcal{D}') \in S] + \delta$$

for any set S of possible outputs of $\mathcal{A}$, and any two data sets $\mathcal{D}, \mathcal{D}'$ that differ in at most one element.

- If dataset D includes one secret x, and D′ = D − {x}, then models trained with **DP-SGD** on D and D′ are nearly identical.
  - DP-SGD has per-example gradient clipping and added Gaussian noise
- **Individual records (like secrets) have minimal influence → model cannot memorize them completely!**
- Training is ~10–100× **slower** than standard + **test loss** within ~10% of non-DP baseline

Discussion

They mentioned:

- **Other ML models** , such as image classifiers?
- What if we have oracle access to **less**? For example, just most likely (arg max) output.
- What if we have oracle access to **more**? (the actual weights and internal activations of the neural network...)

My question(s):

- Is rare, unique, out of distribution data the best way to characterize *secrets*?
- How to prevent from memorizing *in-distribution secrets?*
- How to improve privacy-training time-test loss trade-off?

Thank you!