Auditing Data Provenance in Text-Generation Models

Congzheng Song
Cornell University
cs2296@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

ABSTRACT

To help enforce data-protection regulations such as GDPR and detect unauthorized uses of personal data, we develop a new *model auditing* technique that helps users check if their data was used to train a machine learning model. We focus on auditing deep-learning models that generate natural-language text, including word prediction and dialog generation. These models are at the core of popular online services and are often trained on personal data such as users' messages, searches, chats, and comments.

We design and evaluate a black-box auditing method that can detect, with very few queries to a model, if a particular user's texts were used to train it (among thousands of other users). We empirically show that our method can successfully audit well-generalized models that are not overfitted to the training data. We also analyze how text-generation models memorize word sequences and explain why this memorization makes them amenable to auditing.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Software and application security**.

KEYWORDS

machine learning, text generation, auditing, membership inference

ACM Reference Format:
Congzheng Song and Vitaly Shmatikov. 2019. Auditing Data Provenance in Text-Generation Models . In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330885

1 INTRODUCTION

Data-protection policies and regulations such as the European Union's General Data Protection Regulation (GDPR) [9] give users the right to know how their data is processed. As machine learning (ML) becomes a core component of data processing in many offline and online services, and incidents such as DeepMind's unauthorized use of NHS patients' data to train ML models [3] illustrate the resulting privacy risks, it is essential to be able to *audit* the provenance of personal data used for model training.

In this paper, we design and evaluate a technology that can **help users audit ML models to determine if their data was used to train these models**. We focus specifically on auditing models that generate natural-language text. Text-generation models for tasks such as next-word prediction (the basis of query autocompletion and predictive virtual keyboards) and dialog generation (the basis of chatbots and automated customer service) are extensively trained on personal data, including users' messages, documents, chats, comments, and search queries. Our technology can help users audit a publicly available text-generation model and see if their words were used, perhaps without their permission, to create this model. Furthermore, our work sheds new light on **how deep learning-based, text-generation models memorize their training data**—a topic that has important implications for both data privacy and natural language processing.

The problem of auditing is closely related to the problem of membership inference (see Section 7), but auditing text-generation models requires new technical machinery vs. membership inference in image-classification and categorical models.

First, we assume a very restrictive auditing scenario, which we believe matches how an individual user may audit a deployed ML-based service in practice. The auditor has only black-box access to the model and can query it only on a limited number of inputs. We assume that the model's output does not include numeric probabilities or confidence values (deployed models rarely release these values). Furthermore, we consider scenarios where the model's output is restricted to a relatively small list of words or even a single word. This precludes the application of most previously proposed membership inference methods.

Second, we work with text-generation models that are trained on the data of hundreds or thousands of users and are well-generalized, i.e., their accuracy on test inputs is not substantially different from their accuracy on training inputs. This precludes the application of membership inference methods that exploit the test-train accuracy gap exhibited by overfitted models.

Third, state-of-the-art text-generation models are based on recurrent neural networks (RNNs). We investigate how these models overfit to their training data, what signal this overfitting creates in their outputs, and how to exploit this signal for effective auditing. We show that overfitting in text-generation models appears to manifest primarily via shifted probability distributions over the models' output space. Specifically, we show that these models tend to assign significantly higher rank to relatively rare words when they appear in a familiar context (e.g., in a sentence seen during training). This does not affect the top-ranked, likeliest word generated by the model and therefore—in contrast to "conventional" overfitting—does not manifest in reduced test accuracy.

Fourth, we show how to use auxiliary public datasets and cross-domain training when the auditor does not know the distribution from which the training data for the target model was drawn.

Fifth, we focus on user-level auditing (vs. inferring membership of individual inputs in the training dataset) and measure how many queries are needed to determine if the user's data was

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330885

used—possibly in combination with the data from thousands of other users—to train the model. We quantitatively show that sequences that include relatively rare words are more effective for auditing than word sequences randomly selected from the user's data. We also measure the robustness of our auditing methodology to noise and errors in the test inputs used for auditing. This is important because the user may not know exactly which of his chats or online comments were used, or when the model creator may have started training on the user's data.

Our black-box auditing methodology is very effective. In our experiments on the Reddit, SATED, and Dialogs tasks for, respectively, word prediction, translation, and dialog generation, it performs perfectly (i.e., its AUC score is 1) when the models are trained on the data of hundreds of users and the models' outputs cover the entire vocabulary. Furthermore, it requires surprisingly few queries. If the auditor selects query sequences that include relatively rare words, a *single query* achieves AUC between 0.8 and 0.9 depending on the task, and 8 queries achieve almost perfect AUC.

If the word-prediction and dialog-generation models are restricted to generate and rank only the 500 likeliest words, AUC score of our auditor remains above 0.9. If the translation model generates a *single word* (as opposed to a ranked list of words), the auditor can still infer with a much-better-than-random probability if the model was trained on the word sequences of a particular user. For the Reddit word-prediction model, the auditor's AUC score remains close to 0.9 even if the model was trained on the data of over 4,000 users. Furthermore, we empirically show that our auditing is robust to a significant amount of noise and errors in the audit queries. These results demonstrate that auditing modern text-generation models is feasible in realistic scenarios.

Finally, to explain why auditing works, we provide new insights into memorization in different types of text-generation models. For example, we demonstrate that deep learning-based translation models are more prone than the word-prediction models to memorize training sequences in their inner units.

2 BACKGROUND

2.1 Deep learning

A deep learning model is a function $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ parameterized by θ, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space. Supervised training of a model f_θ aims to find the best set of parameters θ using a labeled training dataset $D = \{(x_i, y_i)\}_{i=1}^n$ and a loss function L.

For ML tasks where the input space is discrete and sparse (e.g., text or location data), the standard approach is to transform discrete inputs into a lower-dimensional continuous vector representation. For a text corpus with vocabulary V, an *embedding* is a function $E : V \mapsto \mathbb{R}^{d_{\text{emb}}}$ where d_{emb}, the dimension of the embedding, is a hyper-parameter. In many NLP tasks, the input is a variable-length sequence of tokens $x = [x^1, \ldots, x^l]$ in the embedding space. The output y can be either a class label (e.g., for sentiment analysis), or a token (e.g., for next-word prediction), or a sequence of tokens (e.g., for machine translation).

Recurrent neural networks (RNNs) are a common architecture for text-generation tasks such as next-word prediction. An RNN maps the input sequence to a sequence of hidden representations $a = [a^1, \ldots, a^l]$, where the computation of a^j is recursively dependent on the previous hidden representation a^{j-1} and the current input token x^j, and feeds these hidden representations to a classifier.

Sequence-to-sequence models are a common architecture for text-generation tasks where both the input $x = [x^1, \ldots, x^l]$ and the output $y = [y^1, \ldots, y^t]$ are sequences of tokens. A typical sequence-to-sequence model consists of an encoder RNN and a decoder RNN. The encoder learns the representation for the input texts, then passes this representation as the initial state for the decoder, which makes word predictions one at a time. Translation models are similar: the decoder predicts words in the target language by feeding its hidden representations to a classifier.

2.2 Text-generation models

Next-word prediction is used in many natural-language applications, including predictive virtual keyboards and query autocompletion. Given an input sequence $x = [x^1, \ldots, x^l]$, the task is to predict the next token x^j from the context $[x^1, \ldots, x^{j-1}]$. RNNs are commonly used for this task. RNN feeds the last hidden representation a^{j-1} in the context sequence to a $|V|$-way classifier to predict the next token, where V is the vocabulary.

Neural machine translation (NMT) models based on RNNs reach near-human performance on many language pairs [34]. The input to these models is a sequence of tokens from the source language, the output is a sequence of tokens from the target language. NMT models use the sequence-to-sequence framework. The input text is encoded as a hidden representation, and the decoder RNN predicts translated tokens based on this representation.

Dialog generation aims to generate replies in a conversation. It is a common component of chatbots and question-answering services. The input is a sentence, the output is the next sentence in the same conversation. Dialog-generation models can also employ a sequence-to-sequence architecture [18, 33]. Similar to NMT, the model encodes the input sentence to a hidden representation, then generates the reply by passing this representation to the decoder.

Loss functions. For the next-word prediction task, given an input sequence $x = [x^1, \ldots, x^l]$, the RNN models the conditional probability $\mathbf{Pr}(x^j | x^1, \ldots, x^{j-1}) = f(x^1, \ldots, x^{j-1})$ and aims to maximize the probability for the sequence $\mathbf{Pr}(x) = \prod_{j=1}^{l} \mathbf{Pr}(x^j | x^1, \ldots, x^{j-1})$. The loss function used when training the model is thus the negative log likelihood: $L(f(x), x) = -\sum_{j=1}^{l} \log f(x^1, \ldots, x^{j-1})$. For the machine translation and dialog-generation tasks where the input is x and the target is $y = [y^1, \ldots, y^t]$, the sequence-to-sequence model computes the probability $\mathbf{Pr}(y^j | y^1, \ldots, y^{j-1}; x)$ as $f(y^1, \ldots, y^{j-1}; x)$. Similar to the next-word prediction task, the loss function is the negative log probability on the target sequence.

3 AUDITING TEXT-GENERATION MODELS

Consider a training dataset $\mathcal{D}_{\text{train}}$ where each row is associated with an individual user, and let $\mathcal{U}_{\text{train}}$ be the set of all users in $\mathcal{D}_{\text{train}}$. The target model f is trained on $\mathcal{D}_{\text{train}}$ using a training protocol $\mathcal{T}_{\text{target}}$, which includes the learning algorithm and the hyper-parameters that govern the training regime. As described in Section 2.2, a text-generation model f takes as input a sequence of tokens x and outputs a prediction $f(x)$ for a single token (if the

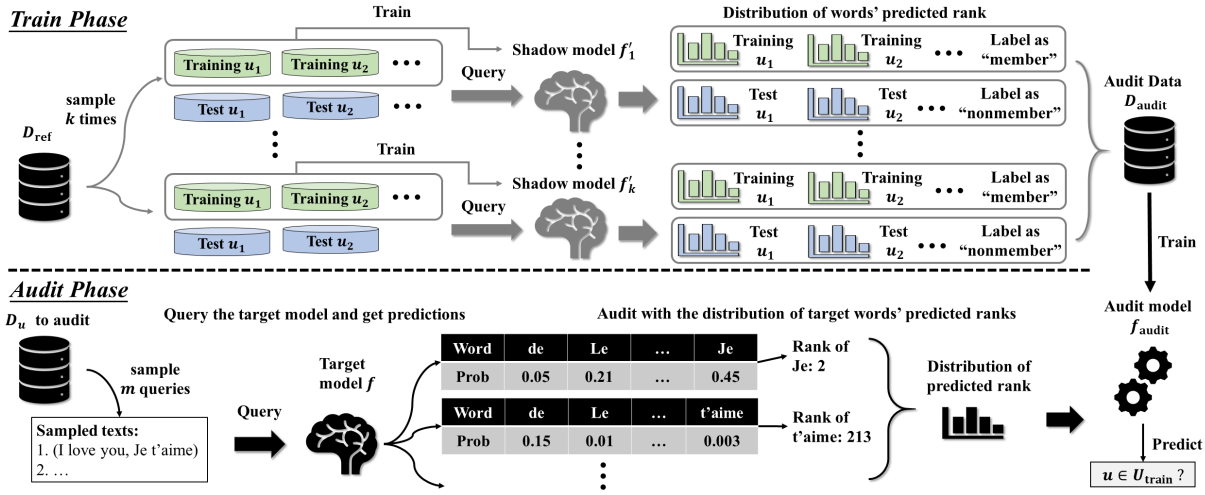

Figure 1: Overview of the auditing process. In the *Train* phase, the auditor trains an audit model; in the *Audit* phase, he applies the audit model to infer if the user's data is part of the target's training dataset.

task is next-word prediction) or a sequence of tokens (if the task is machine translation or dialog generation). The prediction $f(x)$ is a probability distribution or a sequence of distributions over the training vocabulary V or a subset of V. We assume that the tokens in the model's output space are ranked (i.e., the output distribution imposes an order on all possible tokens) but do not assume that the numeric probabilities from which the ranks are computed are available as part of the model's output.

The goal of **auditing** is to infer user-level membership against the target model f, i.e., to decide whether a user $u \in \mathcal{U}_{\text{train}}$ or not.

We assume that the auditor has black-box access to f: given an input query x, the auditor can observe $f(x)$. In realistic deployments of text-generation models, the auditor may not be able to observe the entire vector of ranked words $f(x)$ but only several top-ranked predictions. In our experiments in Section 4.3, we vary the size of the model's output and show how it affects the accuracy of auditing.

We assume that the auditor knows the learning algorithm used to create f but he may or may not know the training hyper-parameters (see Section 4.3). The auditor also needs an auxiliary dataset $\mathcal{D}_{\text{ref}}$ to train shadow models that perform the same task as f.

Fig 1 outlines the auditing process. Similar to standard membership inference [28], the auditor's goal is to learn to distinguish the outputs produced by the target model on sequences that it trained on and its outputs on sequences that it did not see during training. For this purpose, the auditor builds a binary user-level membership classifier f_{audit} that takes as input a (processed) list of predictions obtained by querying f with a subset of the user's dataset $\mathcal{D}_u$ and outputs a decision on $u \in \mathcal{U}_{\text{train}}$. In Section 4.3, we show that a small subset of $\mathcal{D}_u$ is sufficient for this purpose.

Training shadow models. To collect the data for training f_{audit}, the auditor first trains k shadow models $f'_1 \dots f'_k$ (that "simulate" f) using the same protocol $\mathcal{T}_{\text{target}}$ as f with the same hyper-parameters (if known) or varying the hyper-parameters as in Section 4.3.

The training data for each shadow is a random user subset $\mathcal{U}_{\text{ref}}^{\text{train}} \in \mathcal{U}_{\text{ref}}$ of the auxiliary dataset $\mathcal{D}_{\text{ref}}$. Our shadow training technique is inspired by [28], but one essential distinction is that in our case the shadow-training data does not need to be drawn from the same distribution as the training data of the target model In Section 4.3, we show that public sources can be used for $\mathcal{D}_{\text{ref}}$ and the loss in audit accuracy is negligible when $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{ref}}$ are drawn from different domains. This is important for real-world auditing because in practice the auditor may not know the entire distribution of the target model's training data, and API limits may prevent the auditor from querying the target model repeatedly to extract sufficient data for training shadow models as in [28].

The auditor then queries the shadow models with $\mathcal{D}_{\text{ref},u}$ for each u in $\mathcal{U}_{\text{ref}}$ and labels the resulting outputs as "member" if u was part of the shadow's training data, "non-member" otherwise. The next step is to use these labeled predictions to train a binary membership classifier.

Training the audit model. Record-level membership inference typically uses the output probability distribution directly as the feature to distinguish between members and non-members. User-level membership inference in text-generation models calls for a different approach. Each user is associated with multiple sequences, each of which has multiple words. Therefore, the auditor can obtain a collection of output predictions. On the negative side, the actual probabilities associated with each prediction may not be available.

As mentioned before, the output prediction $f(x)$ for a input x is a probability distribution across the entire training vocabulary V, i.e., a $|V|$-dimensional probability vector. $|V|$ is generally large and the probability values are noisy. Instead of the raw probability values, we use the *ranks* of the target words in the output distributions as signals for inferring user-level membership. As we will show in Section 5, even for a well-generalized model (i.e., whose test-train accuracy gap is small), there is a substantial gap in the predicted rank of the same word when it appears in a training text and a test text. Specifically, the model ranks relatively rare words much

higher when it sees them during testing in the same context as it saw them during training.

Given a user u's data $\mathcal{D}_{\text{ref},u}$, the auditor queries the shadow model on each data point $(x, y) \in \mathcal{D}_{\text{ref},u}$ and collects the ranks of y in $f(x)$ into a rank set R_u. Taking English-to-French machine translation task as an example where (x, y) = (I love you, Je t'aime), $f(x) = [f(x)^1, f(x)^2]$ is a sequence of two probability vectors for tokens "Je" and "t'aime." The auditor collects the rank of the probability of "Je" in $f(x)^1$ (e.g., 2) and the rank of the probability of "t'aime" in $f(x)^2$ (e.g., 213), and adds {2, 213} to the rank set R_u. Rank 2 means that the word is the second likeliest prediction in the entire vocabulary. After collecting the ranks for all $(x, y) \in \mathcal{D}_{\text{ref},u}$, the auditor builds a histogram for R_u with a fixed number of bins d. The final feature vector h_u is a d-way count vector where each entry is the count of the ranks in that bin.

The auditor extracts features h_u and labels them as 1 if $u \in \mathcal{U}_{\text{ref}}^{\text{train}}$ and 0 otherwise. The auditor repeats this procedure for each user in each shadow model and obtains a collection of labeled feature vectors $\mathcal{D}_{\text{audit}}$. Finally, the auditor trains a binary membership classifier f_{audit} on $\mathcal{D}_{\text{audit}}$. We refer to f_{audit} as the **audit model**.

Auditing membership in the training data. At inference (i.e., audit) time, the auditor queries the target model f with the user's data $\mathcal{D}_u$. If the number of queries to f is limited, only a sample from $\mathcal{D}_u$ is used. It can be random, but we show in Section 4.3 that it is more effective to select test inputs that have the smallest frequency counts in their labels y, i.e., sequences with relatively rare words are more useful for auditing.

After querying f, the auditor processes the corresponding outputs and obtains a feature vector h_u that describes the distribution of the predicted ranks for each word in $\mathcal{D}_u$. Finally, the auditor feeds h_u to f_{audit}, which decides whether $u \in \mathcal{U}_{\text{train}}$ or not.

4 EXPERIMENTS

4.1 Datasets

The **Reddit comments dataset** (Reddit) is a randomly chosen month (November 2017) from the public Reddit dataset.[1] We filtered it to retain only the users with at least 150 but no more than 500 posts, for a total of 83,293 users with 247 posts each on average. We use the resulting dataset for the next-word prediction task.

The **speaker annotated TED talks dataset** (SATED) consists of transcripts from TED talks,[2] totaling 2,324 talks with roughly 271K sentences in each language [24]. The dataset contains English-French (en-fr), English-German (en-de) and English-Spanish (en-es) language pairs and speaker annotation. We use the data from the en-fr pair for the machine translation task.

The **Cornell movie dialogs corpus** (Dialogs) is a collection of fictional conversations extracted from movie scripts [7]. There are a total of 220,579 exchanges between pairs of characters engaging in at least 5 exchanges, involving 9,035 characters from 617 movies. We use this dataset for the dialogue-generation task.

Cross-domain reference datasets. The auditor may not know the distribution on which the target model was trained and thus needs a reference dataset to train its shadow models. In our experiments, we use public datasets for this purpose. As the cross-domain reference dataset for word prediction, we use the Wikitext-103 corpus[3] obtained by a Wikipedia crawl. For translation, we use the English-French pair in the Europarl dataset [15], a parallel language corpus extracted from the proceedings of the European Parliament. For dialog generation, we use the Ubuntu dialogs dataset [20], which contains two-person technical support chat logs.

These datasets are not labeled with individual users, thus we split them into random n_u subsets, each corresponding to an artificial "user." Our experiments show that we can produce effective audit models even with this artificial separation into users and even though the topics of the reference datasets are very different from the target models' training datasets (e.g., technical support chats vs. conversations between movie characters).

Table 1: Performance of target models. Acc is word prediction accuracy, perp is perplexity.

Dataset	Model	Train Acc	Test Acc	Train Perp	Test Perp
Reddit	1-layer LSTM [12]	0.184	0.206	102.22	113.14
SATED	Seq2Seq w/ attn [24]	0.587	0.535	6.36	10.28
Dialogs	Seq2Seq w/o attn	0.283	0.264	45.57	61.11

4.2 Performance of target models

We use standard architectures and hyper-parameters to train target models (see Section B.1) and evaluate their performance using *word prediction accuracy* $= \frac{1}{M}\sum_{i=1}^{n}\sum_{j=1}^{l_i}\mathbb{I}(\arg\max f(x_i)^j = y_i^j)$ and *perplexity* $= 2^{-\frac{1}{M}\sum_{i=1}^{n}\sum_{j=1}^{l_i}\log f(x_i)^j[y_i^j]}$, where n is the number of data points, $M = \sum_i l_i$ the sum of the number of tokens in all labels, $\mathbb{I}$ is the indicator function that outputs 1 if the predicted token $\arg\max f(x_i)^j$ equals the label token y_i^j and 0 otherwise, and $f(x_i)^j[y_i^j]$ is the probability of predicting y_i^j in $f(x_i)^j$. Perplexity is measured as 2 to the power of the entropy of the label predictions. The lower the perplexity, the better the model fits the data.

Table 1 shows the results for models trained on 300 users, with the test data sampled from 300 disjoint users from the training set. These results match the literature. On Reddit, test accuracy of word prediction is 20%, similar to [23]. On SATED, test perplexity is 10, close to [21]. Low test perplexity shows that the models are learning a meaningful language-generation process. Test-train accuracy gaps are below 5%, indicating that the models are not overfitted. Perplexity gaps are within 15, which is relatively small.

4.3 Performance of auditing

To train shadow models, we sample a set of "shadow users" disjoint from both the training and test users. The number of shadow users is twice the number of training users. We use one half of the shadow users to train shadow models and the other half to collect the shadow models' outputs on the non-members of their training datasets (see Section 3). We train 10 shadow models for all tasks and use a linear SVM as the audit classifier.

Our metrics are precision (the percentage of users classified by the audit model as "members" who are indeed members), recall (the

[1] 1https://bigquery.cloud.google.com/dataset/fh-bigquery:redditcomments
[2] https://www.ted.com/talks

[3] https://einstein.ai/research/blog/the%2Dwikitext%2Dlong%2Dterm%2Ddependency%2Dlanguage%2Dmodeling%2Ddataset

Table 2: Effect of training shadow models with different hyper-parameters than the target model.

Dataset	Accuracy	AUC	Precision	Recall
Reddit	0.990	0.993	0.983	0.996
SATED	0.965	0.981	0.937	0.996
Dialogs	0.978	0.998	0.958	1.000

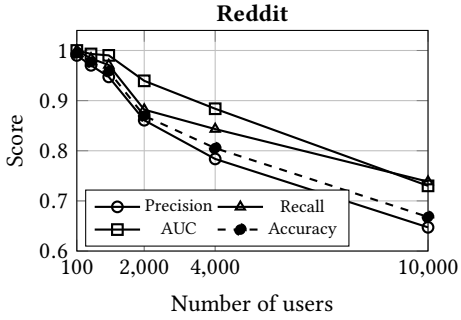

Figure 2: Effect of the number of Reddit users used to train a word-prediction model.

percentage of members who are classified as "members"), accuracy (the percentage of all users who are classified correctly), and AUC, the area under the ROC curve that shows the gap between the scores (i.e., distances to the decision hyperplane of SVM) given by the audit model to members and non-members. We use 300 members and non-members. Therefore, the baseline for all metrics is 0.5, corresponding to random guessing.

Our audit model achieves the perfect score (i.e., 1) on all metrics for all datasets and models when there is no restriction on the output size of the target models (i.e., they produce predictions over the entire vocabulary) and the auditor can query the target models any number of times.

Effect of different hyper-parameters. To demonstrate that knowledge of the target model's hyper-parameters is not essential for successful auditing, we train 10 shadow models for each task with different training configurations (detailed in Appendix B.2). Table 2 shows the results. Auditing scores are still above 0.95 on nearly all metrics for all tasks and models.

Effect of the number of users. To evaluate how the number of users in the training dataset affects the auditor's ability to infer the presence of a single user, we train word-prediction models on 100, 500, 1,000, 2000, 4,000, and 10,000 users from the Reddit dataset. Test users and shadow users are disjoint samples of the same size.

Fig. 2 shows the results. When the number of users is under 1,000, all metrics are at least 0.95. With 4,000 users, precision drops below 0.8 while AUC is still around 0.9. Audit performance drops more significantly when the number of users is 10,000.

Effect of the number and selection of audit queries. To measure the performance of auditing when the auditor is restricted to only a few queries, we vary the number of audit queries between 1, 2, 4, 8, 16, and 32 word sequences.

Fig 3 shows the results. With 32 queries, audit performance exceeds 0.9 on all metrics for all datasets. If query selection is random,

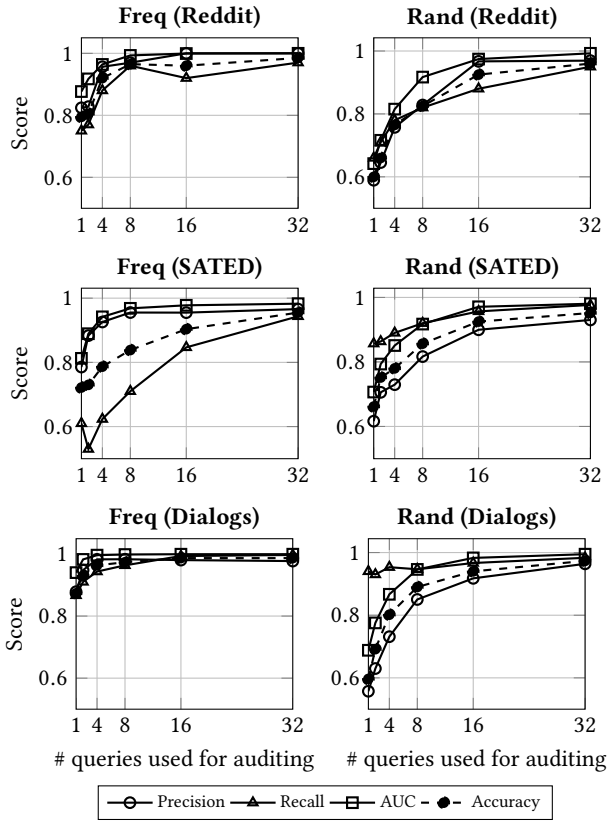

Figure 3: Effect of the number of queries and sampling strategy. Plots on the left show the results when the auditor samples the user's data for queries in the ascending order of frequency counts of tokens in the label; plots on the right show the results with randomly sampled data.

audit performance is low with fewer than 8 queries. If the auditor queries the target with the user's word sequences whose summary word-frequency counts are the lowest, **even with a single query, the auditor can accurately determine if the user's data was used to train the model** on the Reddit or Dialogs dataset. This remarkable result demonstrates the extent to which text-generation models memorize word sequences they were trained on, especially those that contain relatively rare words.

Effect of the size of the model's output. In a realistic deployment of a text-generation model, its output may be limited to a few top-ranked words rather than the entire ranked vocabulary. We constrain the model's output to the top-ranked 1, 5, 50, 500, and 1000 words, while the other hyper-parameters remain as in Section B.1. When building the histogram feature vector for training the audit model (see Section 3), we add an additional feature that counts how many times the ground-truth words are not among the top predictions output by the model.

Table 3 shows the results. On Reddit and Dialogs, the auditor's performance is close to random guessing when the model's outputs are limited to the top 50 or fewer words, increasing to above 0.9

Table 3: Effect of the model's output size. $|f(x)|$ is the number of words ranked by f.

Reddit	Same domain				Cross domain			
$\|f(x)\|$	Acc	AUC	Pre	Rec	Acc	AUC	Pre	Rec
1	0.545	0.549	0.574	0.350	0.505	0.589	0.667	0.020
5	0.550	0.572	0.553	0.520	0.490	0.525	0.495	0.920
10	0.580	0.602	0.582	0.570	0.500	0.552	0.500	0.950
50	0.605	0.648	0.606	0.600	0.505	0.659	0.503	0.980
100	0.725	0.788	0.765	0.650	0.585	0.714	0.549	0.950
500	0.970	0.998	0.970	0.970	0.905	0.992	0.988	0.820
1000	0.985	0.999	0.971	1.000	0.910	0.999	1.000	0.820
SATED								
$\|f(x)\|$	Acc	AUC	Pre	Rec	Acc	AUC	Pre	Rec
1	0.723	0.785	0.770	0.637	0.723	0.785	0.712	0.750
5	0.748	0.838	0.767	0.713	0.767	0.834	0.755	0.790
10	0.800	0.880	0.783	0.830	0.805	0.878	0.814	0.790
50	0.928	0.973	0.908	0.953	0.925	0.979	0.947	0.900
100	0.948	0.981	0.944	0.953	0.942	0.978	0.965	0.917
500	0.972	0.988	0.958	0.987	0.970	0.988	0.983	0.957
1000	0.960	0.984	0.939	0.983	0.967	0.985	0.973	0.960
Dialogs								
$\|f(x)\|$	Acc	AUC	Pre	Rec	Acc	AUC	Pre	Rec
1	0.577	0.618	0.582	0.547	0.538	0.618	0.520	0.977
5	0.575	0.642	0.582	0.530	0.552	0.643	0.528	0.970
10	0.583	0.645	0.591	0.543	0.543	0.638	0.523	0.977
50	0.605	0.660	0.611	0.580	0.537	0.610	0.520	0.963
100	0.647	0.714	0.643	0.660	0.570	0.669	0.541	0.920
500	0.935	0.975	0.917	0.957	0.925	0.969	0.895	0.963
1000	0.972	0.995	0.955	0.990	0.962	0.992	0.948	0.977

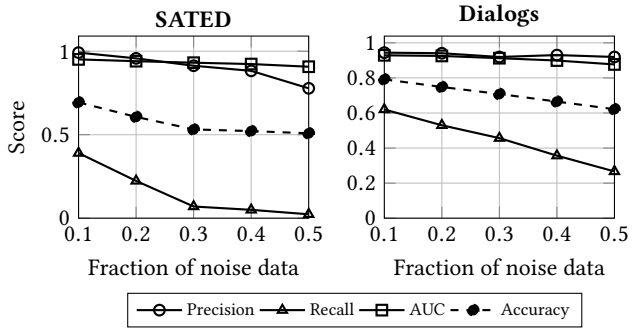

Figure 4: Effect of noise and errors.

when the output size is the top 500 words (only 10% of the entire vocabulary)—regardless of whether the shadow models are trained on the same domain as the target model or a different domain.

For the translation task, **audit performance is much higher than random guessing even if the model outputs just one top-ranked word** and exceeds 0.9 when the model outputs 50 top-ranked words (1% of the vocabulary). These results demonstrate the remarkable extent to which translation models memorize specific word sequences encountered in training.

Effect of noise and errors in the queries. $\mathcal{D}_u$ may be noisy or partially erroneous (e.g., if not all of $\mathcal{D}_u$ was used to train the target model f). To evaluate how this affects auditing, for each training user, we use part of his data to train f and hold out the remaining fraction to represent noise during auditing. We vary this fraction between 0.1, 0.2, …, 0.5.

Table 4: Examples of texts obfuscated using Google translation API and Yandex translation API.

No obfuscation: i see so many adults that could benefit from this going around having themselves a big fat sugar snack or soda pop as a treat it 's so sad
Google: i saw so many adults who can benefit from cherishing big fat sugar snacks and soda pop and going around, it is very sad
Yandex: i think a lot of adults have benefited over your big fat candy and and handling of grief

Table 5: Audit performance on obfuscated Reddit comments.

Dataset	Accuracy	AUC	Precision	Recall
Baseline	1.000	1.000	1.000	1.000
Google	0.580	0.858	0.944	0.170
Yandex	0.500	0.782	0.500	0.010

Fig. 4 shows the results. For SATED and Dialogs, recall drops significantly, close to 0 for SATED when the fraction of noise is 0.5. Increasing the amount of noise biases the audit model towards misclassifying most training users as "non-members." Precision and AUC remain high when noise increases. This may indicate that the scores of the membership classifier at the heart of the audit model still have a distinguishable gap between members and non-members, which is however not learned from the outputs of the shadow models queried with clean data (see Section 3).

Auditing obfuscated data. Finally, we evaluate the effect of obfuscation on the success of auditing. This is the first step towards determining whether text-generation models memorize specific word sequences (which would not be preserved by obfuscation) rather than higher-level linguistic features (which might be).

We use an obfuscation technique, previously considered for evading author attribution [4], that machine-translates the text to a different language and back. We obfuscate the training and test users' Reddit comments using Google[4] and Yandex[5] translation APIs to translate English to Japanese and back to English. Table 4 shows examples of obfuscated text.

Table 5 reports the results of auditing on obfuscated texts. For both Google- and Yandex-based obfuscation, audit accuracy drops to near random and recall is very low. AUC scores are still around 0.8, which is much higher than random guessing. This indicates there is some useful signal in the model's outputs on obfuscated texts, but the auditor's membership classifier—which was trained on non-obfuscated texts—fails to capture this signal.

This is a remarkable result given the poor quality of translation. Even if the user's text has been garbled almost to the point of incomprehensibility, in some cases there is still enough information left to detect its presence in the training data.

5 MEMORIZATION IN TEXT-GENERATION MODELS

In this section, we analyze why auditing works so well for text-generation models that are not overfitted as measured by their test-train accuracy gap (see Section 4.2).

[4]https://cloud.google.com/translate/
[5]https://tech.yandex.com/translate/

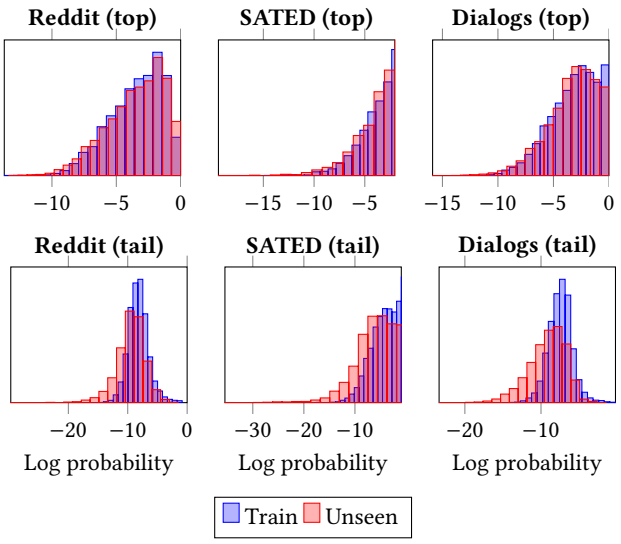

Figure 5: Histograms of log probabilities of words generated by our text-generation models. The top row are the histograms for the top 20% most frequent words, the bottom row are the histograms for the rest.

Word frequency and probability. The loss function for the text-generation models is the sum of the negative log probabilities of the words in the input sequence (see Section 2.2). By its very construction, this loss function "encourages" the model to memorize sequences that occur in the training data.

Fig. 5 shows the histograms of the log probabilities of the more and less frequent words in the training ("train") and test ("unseen") sequences. For the more frequent words, the histograms for the training and test sequences are almost identical. For the less frequent words, the model fits worse for both the training and test sequences as modes focus on smaller log probability values. Most importantly, there is a gap between the less frequent words in the training sequences and those in the test sequences. This gap indicates that the model assigns higher probabilities to words in the training sequences, producing a strong signal that can be used for membership inference and consequently auditing.

These histograms also demonstrate that our text-generation models are not overfitted to their training datasets in terms of the loss value. The 20% most frequent words account for 86.9% of the training data and 88.1% of the test data in Reddit, 89.5% and 90.4% in SATED, and 93.1% and 94.1% in Dialogs. Consequently, these words dominate the training and test loss value. Not surprisingly, text-generation models typically generate words from the top 20% of the word-frequency distribution. As long as the log probabilities remain similar for the top 20% words in both the training and test datasets, the training and test losses of the model will be similar.

Word frequency and predicted rank. Memorization of training sequences produces a much stronger signal in the relative *rank* assigned by the model to the candidate words in the model's output vocabulary. Fig. 6 shows the relationship between a word's rank in the frequency table of the training corpus and its rank in the model's predictions. A smaller rank number indicates that the word is ranked higher in the vocabulary, i.e., more frequent in the corpus or more likely to be predicted by the model. On all datasets, less frequent words exhibit a much bigger gap between the rank predicted by the model when the word appears in a training sequence and when it appears in a test sequence. This explains why our auditing algorithm is more successful when it queries the target model with sequences consisting of the less-frequent words (see Section 4.3).

Ablation analysis. We have shown that probabilities and ranks produced by text-generation models exhibit a gap between the training and test sequences for the less-frequent words but not for the most-frequent words. We hypothesize that these models learn generalizable patterns for the most-frequent words while hard-memorizing the sequences consisting of the less-frequent words.

To gather evidence for this hypothesis, we carried out an experiment based on ablation analysis that was recently proposed to detect memorization in deep-learning models [25]. As more hidden units are ablated, accuracy on the training data degrades quicker for models that are hard-memorizing the training data.

We train target models without dropout (since dropout ablates the hidden units during training) on Reddit and SATED, keeping the other hyper-parameters the same as in Section B.1. We randomly set a fraction of the model's hidden representations to zero and evaluate the accuracy of word prediction on the training data. We vary the fraction from 0.1 to 0.5 on Reddit and 0.1 to 0.9 on SATED and report the accuracy score separately for the 10% most frequent words and the remaining 90% in Fig. 7.

When no hidden units are ablated, accuracy is similar for the most-frequent words and the rest. As the fraction of ablated units increases, accuracy on the less-frequent words drops more significantly than on the most-frequent words. This indicates that predicting less-frequent words is more dependent on specific hidden units in the model and thus involves more memorization.

6 LIMITATIONS OF AUDITING

Models trained on a very large number of users. In some industrial implementation of text-generation models [22, 23], the number of users is on the scale of millions. Performance of our auditor starts to drop when the number of users reaches 10,000 (Section 4.3). We expect that our black-box algorithm will not be able to audit models trained on a very large number (dozens or hundreds of thousands) of users. That said, (a) many state-of-the-art models are trained on fewer than 10,000 users [17, 24, 37], and (b) white-box auditing techniques may be effective even against models trained on dozens of thousands of users. This is a topic for future work.

Deeper models. In our experiments, both the target and shadow models are one-layer LSTMs or GRUs. We have not experimented with auditing deeper and more sophisticated models. We expect that such models are even more susceptible to memorization, but this is another topic for future research.

Differentially private models. In theory, user-level differential privacy (DP) is a direct countermeasure to user-level membership inference. We used federated learning with differential privacy [23] to train a next-word prediction model on the Reddit dataset, setting the number of users to 5,000, user sampling rate to 0.04 per round,

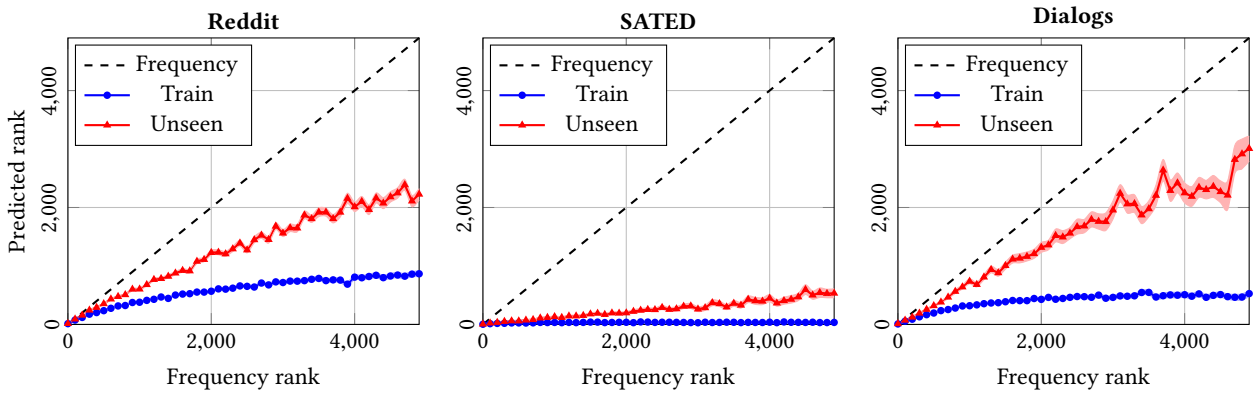

Figure 6: Ranks of words in the frequency table of the training corpus and in the models' predictions (lower rank means that the word is more likely). Shaded area is the 95% confidence interval for all occurrences of the word in the data. These charts demonstrate that the models assign much higher rank to words when they appear in training sequences vs. when they appear in test sequences, especially for the less-frequent words.

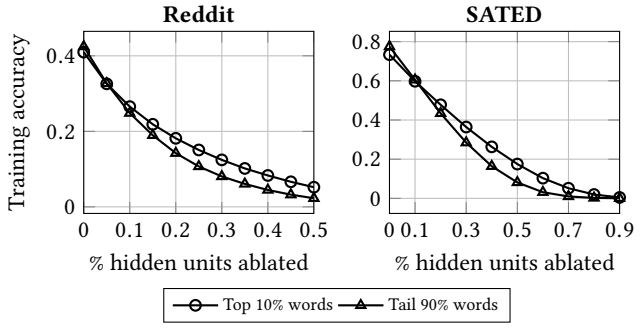

Figure 7: Ablation analysis on Reddit and SATED.

L_2 bound on a single user's contribution to 10.0, and the other hyperparameters as in [23]. After 300 rounds of training, this produced an (ϵ, δ)-DP model with $\epsilon = 4.129$ and $\delta = 1e-4$ which achieves 15% word prediction accuracy, similar to [23]. By contrast, the accuracy of our non-DP model is 20% when trained on only 100 users, i.e., the DP model is significantly less accurate than the non-DP one. Our auditing algorithm fails against the DP model, with performance scores near 0.5 (equivalent to random guessing).

To further investigate the predictive power of the DP model, Fig. 8 plots the ranks of words in the vocabulary (based on their frequencies) and in the model's predictions. The predicted rank is larger than the frequency rank for the 50% most frequent words and remains around 3,000 for the other 50%. The predicted rank is very similar for the words in the training and test sequences, which explains why auditing fails.

The plot also suggests that the differentially private model will almost always predict common words and hardly ever predict relatively rare words. While it does not appear that the model memorizes its training data, it is not clear to what extent it generalizes.

7 RELATED WORK

Membership inference. Membership inference attacks involve observing the output of some computations over a hidden dataset $\mathcal{D}$ and determining whether a specific data point is a member of $\mathcal{D}$. Membership inference attacks against aggregate statistics have been demonstrated in the context of genomic studies [13], location time-series [26], and noisy statistics in general [8].

Shokri et al. [28] develop black-box membership inference techniques against ML models which perform best when the target model is overfitted to the training data. Truex et al. [32] extend and generalize this work to white-box and federated-learning settings. Rahman et al. [27] use membership inference to evaluate the tradeoff between test accuracy and membership privacy in differentially private ML models. Hayes et al. [11] study membership inference against generative models. Long et al. [19] show that well-generalized models can leak membership information, but the adversary must first identify a handful of vulnerable records in the training dataset. Yeom et al. [35] formalize membership inference and theoretically show that overfitting is sufficient but not necessary.

Memorization in ML models. Zhang et al. [36] show that deep learning models can achieve perfect accuracy even on randomly labeled training data. Song et al. [29] present algorithms that intentionally encode the training data in the model. By contrast, we demonstrate that popular text-generation models *unintentionally* memorize their training data.

Carlini et al. [5] show that a black-box adversary can extract specific *numbers* that occur in the training data of a generative model, given some prior knowledge about the format (e.g., a credit card number). For a text-generation model, numbers are essentially random data, thus this is another illustration that models memorize random data. By contrast, we show that text-generation models memorize even words and sentences that are directly related to their primary task and leverage this into an effective auditing method.

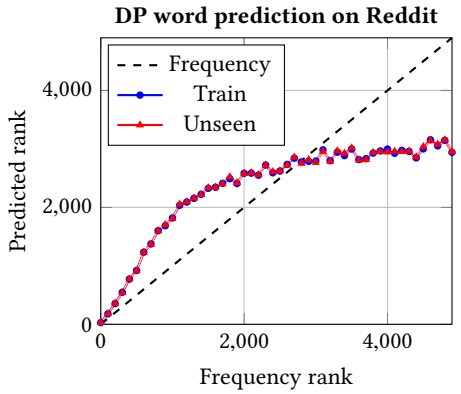

Figure 8: Ranks of words in the training corpus and in the predictions of the differentially private model.

User-level differential privacy. User-level differential privacy (DP) bounds the influence of any single user on the model. McMahan et al. propose a DP federated learning algorithm for language models [23]. With the current state of the art, a massive number of users (at least 10,000) is needed to create DP models that achieve reasonable accuracy. How to build accurate DP models with fewer users remains an open question.

Auditing ML models. Much recent work aims to understand the behavior of ML models with black-box access [2, 16]. These approaches improve the interpretability of the model by showing how features or training data points influence the model's predictions. Other model-auditing research focuses on detecting bias and discrimination [30, 31]. We are not aware of any prior work that aims to audit the use of specific data sources to train a model.

8 CONCLUSION

Deep learning-based, text-generation models for word prediction, translation, and dialog generation are core components of many popular online services. We demonstrated that these models memorize their training data. This memorization does not appear to manifest in reduced test accuracy, which is a symptom of "conventional" overfitting, but is reflected instead in how they rank the candidate words they generate.

We developed a black-box auditing method that enables users to check if their chats, messages, or comments have been used to train someone else's model. Our auditing method, based on a new flavor of membership inference that exploits memorization in text-generation models, is very effective. More powerful auditing algorithms may be possible if the auditor has access to the model's parameters and can observe its internal representations rather than just output predictions. This is a topic for future work.

We view the results of this paper as essentially positive, demonstrating how memorization in ML models can help detect unauthorized uses of sensitive personal data and ensure compliance with GDPR and other data-protection policies and regulations.

Acknowledgments. Supported in part by the NSF grants 1611770 and 1704296 and the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program

REFERENCES

[1] M. Abadi et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.
[2] P. Adler et al. Auditing black-box models for indirect influence. *KAIS*, 54(1):95–122, 2018.
[3] BBC. Google DeepMind NHS app test broke UK privacy law. https://www.bbc.com/news/technology-40483202, 2017.
[4] M. Brennan, S. Afroz, and R. Greenstadt. Adversarial stylometry: Circumventing authorship recognition to preserve privacy and anonymity. *TISSEC*, 15(3):12, 2012.
[5] N. Carlini et al. The Secret Sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv:1802.08232*, 2018.
[6] K. Cho et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, 2014.
[7] C. Danescu-Niculescu-Mizil and L. Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Workshop on Cognitive Modeling and Computational Linguistics, ACL*, 2011.
[8] C. Dwork et al. Robust traceability from trace amounts. In *FOCS*, 2015.
[9] EU. General Data Protection Regulation. https://en.wikipedia.org/wiki/General_Data_Protection_Regulation, 2018.
[10] R.-E. Fan et al. LIBLINEAR: A library for large linear classification. *JMLR*, 9(Aug):1871–1874, 2008.
[11] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro. LOGAN: Membership inference attacks against generative models. In *PETS*, 2019.
[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[13] N. Homer et al. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genetics*, 4(8):e1000167, 2008.
[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
[15] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, volume 5, 2005.
[16] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.
[17] S. Kottur, X. Wang, and V. R. Carvalho. Exploring personalized neural conversational models. In *IJCAI*, 2017.
[18] J. Li et al. A persona-based neural conversation model. In *ACL*, 2016.
[19] Y. Long et al. Understanding membership inferences on well-generalized learning models. *arXiv:1802.04889*, 2018.
[20] R. Lowe, N. Pow, I. V. Serban, and J. Pineau. The Ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *SIGDIAL*, 2015.
[21] T. Luong, M. Kayser, and C. D. Manning. Deep neural language models for machine translation. In *CoNLL*, 2015.
[22] B. McMahan et al. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.
[23] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private language models without losing accuracy. *arXiv:1710.06963*, 2017.
[24] P. Michel and G. Neubig. Extreme adaptation for personalized neural machine translation. *arXiv:1805.01817*, 2018.
[25] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick. On the importance of single directions for generalization. *arXiv:1803.06959*, 2018.
[26] A. Pyrgelis, C. Troncoso, and E. De Cristofaro. Knock knock, who's there? Membership inference on aggregate location data. In *NDSS*, 2018.
[27] M. A. Rahman et al. Membership inference attack against differentially private deep learning model. *Transactions on Data Privacy*, 11(1):61–79, 2018.
[28] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *S&P*, 2017.
[29] C. Song, T. Ristenpart, and V. Shmatikov. Machine learning models that remember too much. In *CCS*, 2017.
[30] S. Tan, R. Caruana, G. Hooker, and Y. Lou. Detecting bias in black-box models using transparent model distillation. *arXiv:1710.06169*, 2017.
[31] F. Tramèr et al. FairTest: Discovering unwarranted associations in data-driven applications. In *EuroS&P*, 2017.
[32] S. Truex et al. Towards demystifying membership inference attacks. *arXiv:1807.09173*, 2018.
[33] O. Vinyals and Q. Le. A neural conversational model. *arXiv:1506.05869*, 2015.
[34] Y. Wu et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*, 2016.
[35] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *CSF*, 2018.
[36] C. Zhang et al. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.
[37] S. Zhang et al. Personalizing dialogue agents: I have a dog, do you have pets too? In *ACL*, 2018.

Reproducibility Information

In this appendix, we provide the pseudo-code of the auditing algorithm and describe all model architectures, configurations, and hyper-parameters needed to reproduce the results.

A PSEUDO-CODE FOR AUDITING

Algorithm 1 is the pseudo-code for the auditing process. Function `AuditMembership` is used to audit if the user's data $\mathcal{D}_u$ was included in the training data of the target model f. The auditor first trains an audit model using function `TrainAuditModel`. He then extracts histogram features h_u on (possibly sampled) user's data using function `HistogramFeature`. Finally, the audit model predicts if the user's data was used to train f given the extracted features h_u.

Function `TrainAuditModel` is the procedure for training the audit model. The auditor first trains k shadow models with a randomly sampled set of users, collects their outputs, extracts feature vectors for the training and test users, and labels these feature vectors accordingly. The auditor then trains a binary classifier based on the labeled feature vectors.

Function `HistogramFeature`extracts features from the predicted ranks, as described in Section 3. The feature vector is a d-dimensional vector representing the histogram of predicted ranks. The ith entry of the vector is the count of ranks in the range $[(i-1)\cdot b, i\cdot b]$, where b is the histogram bin size.

When the number of queries to the target model is limited to m, the auditor uses function `SampleQueries` to sample a subset of the user's data. The auditor can choose either m texts at random, or m texts with the smallest word frequencies.

Algorithm 1: Auditing text-generation models

Hyper-parameters: auditor's reference dataset $\mathcal{D}_{\text{ref}}$, number of shadow models k, user's data $\mathcal{D}_u$, target model f, target model-training protocol $\mathcal{T}_{\text{target}}$, audit model-training protocol $\mathcal{T}_{\text{audit}}$, maximum number of queries m, number of bins in histogram d

function `AuditMembership`()
  $f_{\text{audit}} \leftarrow$`TrainAuditModel`()
  $\mathcal{D}_{\text{sample},u} \leftarrow$`SampleQueries`$(m, \mathcal{D}_u)$
  $h_u \leftarrow$`HistogramFeature`$(f, \mathcal{D}_{\text{sample},u})$
  return prediction of membership $f_{\text{audit}}(h_u)$

function `SampleQueries`$(m, \mathcal{D})$
  if random sample **then**
    return randomly selected m rows in $\mathcal{D}$
  else ▷ *sample based on frequency*
    $C \leftarrow$ {Σ (frequency of w for w in y) $|\forall(x,y) \in \mathcal{D}\}$
    $I \leftarrow$ indices of m smallest values in C
    return m rows in $\mathcal{D}$ indexed by I
  end if

function `TrainAuditModel`()
  $\mathcal{D}_{\text{audit}} \leftarrow \emptyset$ ▷ *dataset for building the audit model*
  $\mathcal{U}_{\text{ref}} \leftarrow$ users in $\mathcal{D}_{\text{ref}}$
  for $i = 1$ to k **do** ▷ *train k shadow models*
    $\mathcal{U}^{\text{train}}_{\text{ref}}, \mathcal{U}^{\text{test}}_{\text{ref}} \leftarrow$ random split $\mathcal{U}_{\text{ref}}$
    $\mathcal{D}^{\text{train}}_{\text{ref}} \leftarrow \cup_{u \in \mathcal{U}^{\text{train}}_{\text{ref}}} \{\mathcal{D}_{\text{ref},u}\}$
    Train a shadow model $f'_i \leftarrow \mathcal{T}_{\text{target}}(\mathcal{D}^{\text{train}}_{\text{ref}})$.
    for every u in users of $\mathcal{U}_{\text{ref}}$ **do**
      $\mathcal{D}_{\text{ref},u} \leftarrow$ data in $\mathcal{D}_{\text{ref}}$ associated with u
      $h'_u \leftarrow$`HistogramFeature`$(f'_i, \mathcal{D}_{\text{ref},u})$
      $z'_u \leftarrow 1$ if u in $\mathcal{U}_{\text{ref-train}}$ else 0
      $\mathcal{D}_{\text{audit}} \cup \{(h'_u, z'_u)\}$
    end for
  end for
  Train the audit model $f_{\text{audit}} \leftarrow \mathcal{T}_{\text{audit}}(\mathcal{D}_{\text{audit}})$
  return f_{audit}

function `HistogramFeature`$(f, \mathcal{D})$
  $R \leftarrow$ {**rank**(y) in $f(x) | \forall(x,y) \in \mathcal{D}\}$
  Initialize feature vector h with d entries.
  $b \leftarrow |V|/d$ ▷ *histogram bin size*
  for $i = 1$ to d **do** ▷ *count of ranks in each bin*
    $h_i = |\{(i-1)\cdot b \le r < i \cdot b | r \in R\}|$
  end for
  return feature vector h

B EXPERIMENT SETUP

B.1 Target models

Next-word prediction. We use a one-layer long short-term memory (LSTM) [12] as the target model. LSTM is a more complicated RNN that can capture the long-term dependency in the sequence. The input sequence of tokens is first mapped to a sequence of embeddings. The embedding is then fed to the LSTM that learns a hidden representation for the context for predicting the next word.

Neural machine translation. We use a sequence-to-sequence target model with the attention module as described in [24]. Both the encoder and the decoder are one-layer LSTMs that operate on the embedding of source tokens and target tokens. The attention module adds an additional layer that operates on all hidden representations in the encoder LSTM and helps the decoder determine where to pay attention in the source texts when predicting a token in the target language.

Dialog generation. We use a sequence-to-sequence model without the attention module. The encoder and the decoder are one-layer LSTMs.

B.2 Hyper-parameters

Target models. We train the word-prediction model on the comments of 300 randomly selected users from the Reddit dataset. We set both the embedding dimension and LSTM hidden-representation size to 128. For training the LSTM, we use the Adam optimizer [14] with the learning rate set to 1e-3, batch size to 35, and the number of training epochs to 30.

We train the translation and dialog-generation models on 300 randomly selected users from SATED and Dialogs, respectively. We set both the embedding dimension and LSTM hidden-representation size in the encoder and decoder to 128. We use the Adam optimizer with the learning rate set to 1e-3, batch size to 20, and the number of training epochs to 30.

For all datasets, we fix the vocabulary to the most frequent 5,000 tokens in the training texts. Tokens not in the vocabulary are replaced with a special <UNK> token. To prevent overfitting, we add dropout with 0.5 rate to all hidden layers of all models.

Shadow models. For the experiments in Section 4.3, we construct shadow models using different hyper-parameters than the target models. On all tasks, we used Gated Recurrent Units (GRU) [6] instead of LSTM. The size of hidden units and embedding is set to 64, 96, 128, 160, ..., 352 for the shadow models. We optimize the shadow models using momentum SGD with the learning rate set to 0.01, momentum set to 0.9, and number of training epochs to 50.

B.3 Implementation

Target and shadow models. All target and shadow models were implemented with Keras[6] using TensorFlow [1] backend.

Audit model. We use linear SVM implemented in LIBLINEAR [10] to train the audit model with the default hyper-parameters.[7]

Hardware. All models were trained on a machine with 3 NVIDIA Titan X GPUs, 8-core Intel(R) Core(TM) i7-5960X CPU @ 3.00GHz and 94 GBs of RAM.

[6] https://keras.io/
[7] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html