DeepMemory: Model-based Memorization Analysis of Deep Neural Language Models

Derui Zhu
Technical University of Munich
Munich, Germany
derui.zhu@tum.de

Jinfu Chen*
Huawei Technologies Canada
Kingston, Canada
jinfu.chen1@huawei.com

Weiyi Shang
Concordia University
Montreal, Canada
shang@encs.concordia.ca

Xuebing Zhou
Huawei Munich Research Center
Munich, Germany
xuebing.zhou@huawei.com

Jens Grossklags
Technical University of Munich
Munich, Germany
jens.grossklags@in.tum.de

Ahmed E. Hassan
Queen's University
Kingston, Canada
ahmed@cs.queensu.ca

***Abstract*—The neural network model is having a significant impact on many real-world applications. Unfortunately, the increasing popularity and complexity of these models also amplifies their security and privacy challenges, with privacy leakage from training data being one of the most prominent issues. In this context, prior studies proposed to analyze the abstraction behavior of neural network models, e.g., *RNN*, to understand their robustness. However, the existing research rarely addresses privacy breaches caused by memorization in neural language models. To fill this gap, we propose a novel approach, *DeepMemory*, that analyzes memorization behavior for a neural language model. We first construct a memorization-analysis-oriented model, taking both training data and a neural language model as input. We then build a semantic first-order Markov model to bind the constructed memorization-analysis-oriented model to the training data to analyze memorization distribution. Finally, we apply our approach to address data leakage issues associated with memorization and to assist in dememorization. We evaluate our approach on one of the most popular neural language models, the *LSTM*-based language model, with three public datasets, namely, WikiText-103, WMT2017, and IWSLT2016. We find that sentences in the studied datasets with low perplexity are more likely to be memorized. Our approach achieves an average AUC of 0.73 in automatically identifying data leakage issues during assessment. We also show that with the assistance of *DeepMemory*, data breaches due to memorization of neural language models can be successfully mitigated by mutating training data without reducing the performance of neural language models.**

***Index Terms*—Deep learning, neural language model, model-based analysis, privacy, memorization**

I. Introduction

Artificial intelligence (AI) software is important for automating and making autonomous decisions. In particular, the rise of neural network models had a huge and significant impact on many real-world applications, e.g., natural language processing [1], [2], image recognition [3], and autonomous driving [4], [5]. However, the increasing diversity and complexity of such neural network models make their security, reliability and robustness a critical and difficult issue to address. Therefore, researchers in different fields are now working intensely on guidelines for *Trustworthy AI* and *Safe AI*. For example, software engineering researchers propose techniques that analyze and explain AI models in order to ensure the security and safeness of AI-based software [6], [7].

Similar to traditional (i.e., not based on AI) software, AI-based solutions have been reported by many prior studies to trigger security concerns, such as data privacy leakage [8]. Although various verification techniques, e.g., static analysis, symbolic execution analysis and fuzzing techniques, can be used to guide the assurance of traditional software security, those techniques are not applicable for AI-based software. In contrast, to the best of our knowledge, there is a relative lack of techniques that can assist in the verification of security in AI-based software.

Data privacy leakage is a typical security issue in AI models. Previous work [9], [10], [11] has shown that neural language models tend to memorize the training data instead of learning its latent characteristics. This can be exploited to extract privacy-critical information from the data, potentially leading to significant financial and reputational harm [12]. More generally, memorization with a neural language model may reveal insights regarding its internal behavior. Prior studies [13], [14], [15] have been proposed to analyze certain aspects of the internal behavior of deep neural networks in order to assist with detecting adversarial examples and to guide the security testing of deep learning models [14]. However, the existing research rarely targets a model's internal *memorization* behavior. Hence, the existing research is limited when it comes to analyzing and preventing leakage of sensitive private information from training data of a publicly released model.

To fill this research gap, we propose a novel approach, *DeepMemory*, to assist in verifying security in AI-based software by analyzing the internal memorization behavior of neural language models. We first construct a memorization-analysis-oriented model taking both training data and a neural language model as input. Second, we bind the constructed memorization-analysis-oriented model to the training data. We then build a semantic first-order Markov model to analyze

*Jinfu Chen (jinfu.chen1@huawei.com) is the corresponding author.

memorization distribution. Finally, we apply our approach to two downstream application scenarios, including data leakage risk assessment and dememorization assistance.

We evaluate our approach on one of the most popular neural language models, i.e., the *LSTM*-based language model, with three public datasets, namely, WikiText-103 [16], WMT2017 [17] and IWSLT2016 [18]. We investigate the *LSTM*-based language model with the same architecture and configuration as Merity *et al.* [19]. We find that by observing memorization characteristics for training data, sentences with low perplexity are more likely to be memorized by a neural language model. Our approach achieves an average AUC of 0.73 in automatically identifying data leakage issues during assessment. Finally, by following our approach, the memorization risk from a neural language model can be mitigated by mutating training data without impacting the quality of neural language models.

To the best of our knowledge, our approach is the first attempt to assist in the verification of a common and important privacy related issue in AI models, i.e., memorization in neural language models. In particular, our work makes the following contributions:

- We model the internal memorization behavior of neural language models, e.g., the *LSTM*-based language model, in order to address training data leakage issues caused by the model's memorization behavior.
- Our approach can automatically assess memorization-related privacy leakage in neural language models.
- With our work, we can assist in the dememorization process in order to address memorization issues in neural language models.
- Our approach can be used to assist in the security testing and assurance processes for AI models.

The rest of this paper is organized as follows: Section II provides the background for our work. Section III gives an overview of our approach, and Sections IV–VI present the details. Section VII presents the results of our evaluation. Section IX discusses threats to the validity of our work, and Section X summarizes related work. Finally, Section XI offers concluding remarks.

II. Background

A. Language modeling

A language model is a probability distribution over sequences of words [20]. In other words, a language model aims to learn a probabilistic model that is capable to predict the next word in a sequence based on the given preceding words. Formally, the probability distribution of a language model can be defined as $Pr(w_1, w_2, ..., w_n)$:

$$Pr(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} Pr(w_i | w_1, ..., w_{i-1}) \tag{1}$$

where w_i refers to a word. This language model has been successfully used in many applications, such as speech recognition [21], machine translation [22], sentiment analysis [23], and information retrieval [24]. In particular, the neural language model is becoming increasingly popular and has been successfully used in many applications. The neural language model uses different kinds of neural networks to model sequence probability, and it transforms words into vectors and uses the vectors as input for a neural network to predict the next words. A very common neural language model is the long short-term memory (*LSTM*) language model. An *LSTM* network contains a plethora of units, called memory blocks. Each memory block represents a hidden state, i.e., s_t, s_{t+1}, s_{t+2}. Prior work [25] illustrated the success of *LSTM*-based language models in multiple applications motivating us to conduct our work in this context.

B. Memorization risk from training Data

A language model requires domain-specific data for training in order to achieve a good model performance. However, a well-performing language model might suffer from data leakage due to memorization of training data. Such leakage is particularly troublesome when sensitive data including personal/private data, transaction data, or governmental data becomes available to an attacker. Examples of such sensitive data are Social Insurance Numbers (in Canada) or health-related data.

Sensitive data that is part of training data may be memorized by a neural language model during model training. When leveraging such a mechanism, one can develop attacks to extract such data from public trained language models [26]. In other words, data leakage due to the memorization mechanism is a typical security weakness in AI-based language models. The study of memorization privacy risk includes two lines of research: memorization-related privacy attacks and defenses.

1) Memorization-related privacy attacks: If a model is ignorant towards privacy-preserving algorithms, it tends to blindly remember some sequences from the training data [8], [27]. Previous studies [8], [28] find that memorization is a common phenomenon in language models, and privacy attacks aim to reconstruct verbatim memorization sequences for training data.

2) Memorization-related privacy defenses: There are typically two ways to support dememorization, i.e., differential privacy and regularization. Differential privacy, which injects noise into the process of model training, is a well-known solution for minimizing memorization in model training [29]. As such, it is challenging to identify whether specific data is part of the training data. Another typical privacy defense approach is regularization. One can add regularization to the loss function of language model optimization [26].

III. Overview of our approach

In this section, we present an overview of our approach for analyzing the memorization behavior for a given language model. Similar to traditional software vulnerability detectors, our approach acts as an automated technique for detecting potential data leakage security vulnerabilities that occur due to memorization in language models. An overview of our approach is shown in Figure 1. It consists of three

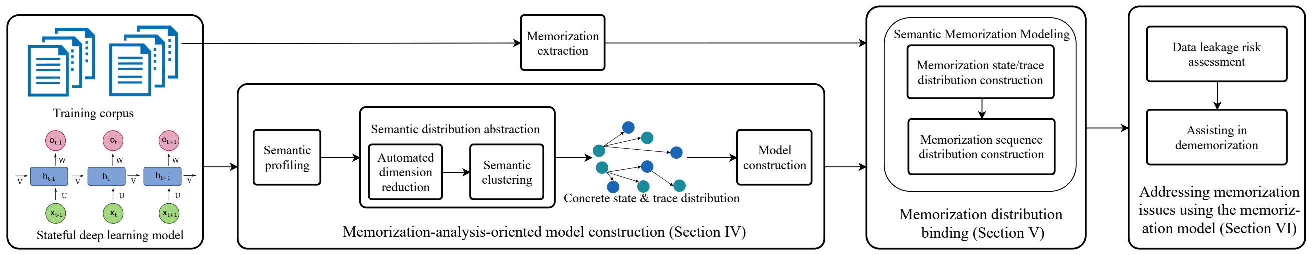

Training corpus
Stateful deep learning model
Memorization extraction
Semantic profiling
Semantic distribution abstraction
Automated dimension reduction
Semantic clustering
Concrete state & trace distribution
Model construction
Memorization-analysis-oriented model construction (Section IV)
Semantic Memorization Modeling
Memorization state/trace distribution construction
Memorization sequence distribution construction
Memorization distribution binding (Section V)
Data leakage risk assessment
Assisting in dememorization
Addressing memorization issues using the memorization model (Section VI)

Fig. 1: An overview of our approach.

phases: 1) memorization-analysis-oriented model construction, 2) memorization-distribution binding, and 3) addressing memorization issues using the memorization model.

In the first phase, we construct a memorization-analysis-oriented model. Taking both training data and a neural language model as input, we first profile the given model to extract semantic information, i.e., hidden states and traces. Such profiling outputs initial states and traces that represent the model behavior. Typically, a large number of initial states and traces exist due to the massive scale of training data. We then abstract a semantic distribution from the initial states and traces. In particular, we transform initial states to intermediate states by reducing the high dimensions of each initial state. We then apply a clustering algorithm to group the intermediate states and traces into clusters, i.e., to derive concrete states and traces. Finally, we construct a memorization-analysis-oriented model based on the concrete states and traces distribution.

In the second phase, our approach binds the memorization-analysis-oriented model to the training data to analyze the memorization distribution. This phase takes the memorization-analysis-oriented model constructed from the last phase and the training corpus as input. To analyze the memorization distribution, we first extract memorization sequences from the training data. We then build a semantic first-order Markov model to model the memorization distribution.

In the final phase, we apply our approach to two downstream tasks, including data leakage risk assessment and dememorization assistance. The first downstream task automatically identifies potential data leakage issues in the model (comparable to bug detection). The second downstream task assists in the repair of the models given the identified issues (comparable to program repair). We detail each phase of our approach in the subsequent Sections IV – VI.

IV. Memorization-analysis-oriented model construction

In this section, we construct a memorization-analysis-oriented model. Algorithm 1 presents the details of its construction. Given an *LSTM*-based language model and its training data, we first profile the model to extract the initial states and traces by iterating words in each sentence over the training data. We then abstract the initial states and traces to construct our memorization-analysis-oriented model. We describe each step in detail below.

Algorithm 1: Memorization-analysis-oriented model construction algorithm

```
input : $\mathcal{R} = (\mathcal{D}, \delta, f)$: LSTM-based language model
        $G$: semantic distribution abstraction function
        $\theta$: information loss threshold
        $\mathcal{D}$: sentences
        $\sigma$: minimum number of neighbors threshold
        $\rho$: distance threshold
output: $\mathcal{M}$: memorization-analysis-oriented model
1  $S \leftarrow []$ ;                                   // initial states set
2  $T \leftarrow []$ ;                                   // initial traces set
3  for $W \in \mathcal{D}$ do                            // loop sentences to extract states
4      $\mathbf{s} \in [\delta_i(W[:i])]_{i=0}^{|W|}$ ;  // extract all hidden states of a sentence
5      for $i : 1 \in |\mathbf{s}|$ do
6          $S.add(s_i)$;
7          $T.add((s_{i-1}, s_i))$
8  $g \leftarrow G(S, \theta, \sigma, \rho)$ ;          // semantic distribution abstraction
9  $S^{'} = []$ ;                                        // concrete states set
10 for $s \in S$ do
11     $s' \leftarrow g(s)$ ;
12     $S^{'}.add(s')$ ;
13 $T^{'} \leftarrow []$ ;                               // concrete traces set
14 for $(s_{i-1}, s_i) \in T$ do
15     ${s'}_{i-1} \leftarrow g(s_{i-1})$ ;
16     ${s'}_i \leftarrow g(s_i)$ ;
17     $T^{'}.add({s'}_{i-1}, {s'}_i)$ ;
18 return $\mathcal{M}(\mathcal{D}, S^{'}, T^{'}, f)$;
```

A. Semantic profiling

Recent research on deep neural network models [14], [15], [30] highlights that states and traces are efficient for understanding stateful model behaviors over data distribution. A neural language model can be seen as a stateful model. The *LSTM*-based model is one of the most typical neural language models. Therefore, in order to analyze *LSTM*-based neural language model behavior, we profile the model to extract the initial semantic states and traces as the first step. We first explain the definition of state and trace in neural language model analysis.

Suppose that we have an *LSTM*-based language model $\mathcal{R} = (\mathcal{D}, \delta, f)$. $\mathcal{D}$ refers to all sentences used for training. f is the distribution of a language model, and δ is an internal state extractor of the model that is used to transform each word in a sentence to a state. For example, when we feed a sequence "Ian goes home at 6 pm on weekdays and goes swimming at 7 pm every day." to a *LSTM*-based language model f with 100 hidden units, we can obtain a list of hidden-state vectors of *LSTM* with 100-dimension for each feed input word, i.e., $[[0.1, ..., 0.3], [0.2, ..., 1.3], [1.5, ..., 0.3], ..., [0.07, ..., 0.4]]$, by using internal state extractor δ. Particularly, $\delta(home) =$

$[1.5, ..., 0.3]$.

With the internal hidden state set, we construct a corresponding state flow, i.e., trace, over two hidden states ordered chronologically. The trace represents a transition relation for a pair of consecutive hidden states. In our illustrative example, the trace between hidden state "goes" and state "home" is represented by $(\delta(goes), \delta(home))$.

In Algorithm 1, we first define two empty sets (Line 1 and 2) for hidden states S and traces T. We then iterate each sentence $\boldsymbol{W}$ in the training data and extract the state and trace of each word w (Line 3 to 7). Particularly, at the i-th timestamp t, each word in a sentence is transformed to a state s_i using the internal state extractor δ. A trace is accordingly extracted to (s_{i-1}, s_i). Finally, we construct a state set S and trace set T for the whole training data $\mathcal{D}$ and define it as an initial model.

B. Semantic distribution abstraction

After semantic profiling, we obtain an initial model to represent the *LSTM*-based neural language model behaviors over training data. However, such a granular representation contains a plethora of discrete states and traces. For example, an *LSTM*-based neural language model potentially produces up to 100 thousand states and 900 thousand traces for a corpus containing 10,000 sentences with an average length 10 of words. It is impractical to understand the internal behavior of a given model with such a huge number of states and traces. Therefore, in this step, we abstract the semantic distribution of a given language model from the perspective of states and traces.

1) ***Automated dimension reduction:*** The dimension of each initial state generated by semantic profiling is equal to the number of hidden units in *LSTM* core, which usually is very high. It is hard to find the latent characteristics over high dimensional space since the distribution of data with high dimension tends to be sparse [31]. Therefore, we first automatically reduce the dimension of each initial state generated by semantic profiling to an optimal number. Du *et al.* [14] applied Principal Component Analysis (PCA) to reduce the dimension of semantic space to a small number, in order to efficiently find the common correlation over states. However, an obvious limitation in their approach exists. When the dimension of an initial state is high, arbitrary dimension reduction may lead to a huge information loss. The information loss from modeling may potentially introduce a significant bias in memorization-analysis-oriented model construction. To improve the memorization-analysis-oriented model construction, we use a classic metric, Relative Information Loss [32], to measure the information loss during dimension reduction. In detail, we have a number of n vectors V and each vector is with m-dimension space, i.e., $[v_0, v_1, ..., v_m]$. We want to transform the n initial vectors to vectors $\hat{V}$, and each transformed vector is with k-dimension, i.e., $[\hat{v}_0, \hat{v}_1, ..., \hat{v}_k]$. The corresponding information loss is defined as $\psi(k)$.

In order to overcome the aforementioned limitation, we take information loss into account for dimension reduction in order to secure the utility of the transformed internal state. We set a threshold θ to control information loss, and the decision process of finding the optimal k can then be defined as:

$$\arg\min_k |\psi(k) - \theta| \tag{2}$$

Finally, this step outputs intermediate states and each state is k dimensions. In our example, we reduce the 100-dimension of each state to three dimensions. For example, the word "home" would be with a reduced initial state $[1.5, 0.7, 0.3]$.

2) ***Semantic Clustering:*** To identify the latent characteristics over the intermediate states, we apply a clustering algorithm (DBSCAN) to group together intermediate states that are close to each other in terms of cosine distance threshold ρ and minimum number of neighbors σ. DBSCAN-based clustering is suitable for data with an arbitrary shape [33]. ρ specifies for the minimum cosine distance which two intermediate state points should be considered as neighbors. σ determines the minimum number of neighbors to be defined as a core state. Each core state and its neighbors form a cluster labeled as a concrete state. In our running example, the words "home" and "swimming" are grouped into one cluster. Therefore, we would label the hidden states of the words "home" and "swimming" as a single identical concrete state.

C. Memorization-analysis-oriented model construction

With the concrete states from the clustering, we construct a final memorization-analysis-oriented model. We first transform the high-dimensional initial states into intermediate states with an optimal dimension. We then transform the intermediate states to concrete states. Note from Algorithm 1, we define an abstraction function G to abstract the initial states and traces (Line 8). The inputs of the function G are the initial states, and three threshold values, i.e., information loss threshold θ, the number of cores σ, and distance threshold ρ. We then initialize two sets $S^{'}$ (Line 9) and $T^{'}$ (Line 13) for concrete states and traces, respectively. Next, for each initial state s_i, we use the defined semantic distribution abstraction to abstract the state to ${s'}_i$ (Line 10 to 12). Similarly, for each initial trace (s_{i-1}, s_i) composed of two states s_{i-1} and s_i, we apply the same abstraction function to abstract the two states to ${s'}_{i-1}$ and ${s'}_i$. We then connect the two abstracted states into a concrete trace $({s'}_{i-1}, {s'}_i)$ (Lines 14 – 17). The final output is the memorization-analysis-oriented model (Line 18). In our running example, the final memorization-analysis-oriented model is represented by the concrete state and trace set.

V. Memorization distribution binding

Prior studies [8], [27] have reported that memorization is a severe issue in language models. To achieve a good performance, a model all too often intends to remember the training data during the training process instead of learning the latent characteristics. Regularization techniques, such as dropout and batch normalization, aim to solve the model overfitting issue and improve the generality of AI models. Although the regularization techniques are widely adopted for training a complicated model, e.g., an LSTM-based model, the models may still memorize part of the training data [27].

Such memorization might be exploited to extract private data from a given language model. Therefore, in this section, we quantify the memorization behavior in a memorization-analysis-oriented model, i.e., the output from Section IV. The details for analyzing memorization behavior are shown in Algorithm 2. Given a memorization-analysis-oriented model and training data as input, we bind the memorization distribution by building a semantic Markov model to map the memorization-analysis-oriented model to training data. In particular, we first extract memorization. We then build a first-order Markov model to represent the semantic memorization distribution.

A. Memorization Extraction

From our memorization-analysis-oriented model generated in Section IV, we obtain the final concrete states and traces for each sentence in the training data. However, such states and traces cannot be applied to quantify memorization behavior of a given language model directly. Therefore, to quantify the memorization behavior efficiently, we first define a memorization concept called a memorization sequence. Given a language model $\mathcal{R} = (\mathcal{D}, \delta, f)$ and a prefix c, a string of l with length N is considered to be a memorization sequence if such a string is equal to:

$$\arg\max_{l':|l'|=N} \mathcal{R}(l'|c) \tag{3}$$

where c and l are both from the training corpus. In our example, given a prefix c "Ian goes", a language model would predict a string "home at 6 pm on weekdays" as the most likely output. We call a string such as "home at 6 pm on weekdays" a memorization sequence based on the prefix "Ian goes".

With memorization sequences, we classify the concrete state|trace from the memorization-analysis-oriented model into two types, i.e., memorization state|trace and non-memorization state|trace. If a state|trace is visited by any memorization sequence, we consider the state|trace to be a memorization state|trace. Otherwise, it is a non-memorization state|trace. Finally, we can construct a semantic distribution for all the concrete states and traces in terms of memorization. In Algorithm 2, we first initialize two dictionaries, *MT* and *MS*, to represent memorization traces and states, respectively (Line 1 and Line 2). We also initialize two dictionaries, *AT* and *AS* for all the concrete traces and states output from Section IV (Line 3 and Line 4). Next, we iterate each sentence in the training data to abstract state and trace for each word. If an abstracted state|trace is visited by a memorization sequence, we label the state|trace to a memorization state|trace (Line 6 to Line 15). In our running example, the concrete state corresponding to "home" is classified as a memorization state.

B. Semantic Memorization Modeling

With the memorization states and traces, we build a first-order Markov model to learn the memorization semantic distribution conditioned on the state from the last step. Sequential behavior can be regarded as a discrete-time Markov chain. Therefore, the memorization probability over a sequence can be modeled by a first-order Markov model [34].

Algorithm 2: Memorization analysis algorithm

```
   input  : $\mathcal{M} = (\mathcal{D}, T, S, f)$: memorization-analysis-oriented model,
            $g$: abstraction transformation function,
            $H$: memorization sequence abstraction function,
            $\mathcal{D}$: sentences
   output : $\mathcal{E}(\gamma, \tau)$: first-order Markov memorization model
1  $MT \leftarrow \{\}$ ;                       // a dictionary of memorization traces
2  $MS \leftarrow \{\}$ ;                       // a dictionary of memorization states
3  $AT \leftarrow \{\}$ ;                       // a dictionary of concrete traces
4  $AS \leftarrow \{\}$ ;                       // a dictionary of concrete states
5  $h \leftarrow H(\mathcal{D}, f)$ ;   // function to check if an input is memorization trace
6  for $W \in \mathcal{D}$ do       // loop every sentence to extract states and traces
7      $\mathbf{s} \in [\delta_i(W[:i])]_{i=0}^{|W|}$;
8      for $i \in 1 \ldots |W|$ do
9          ${s'}_{i-1} \leftarrow g(s_{i-1})$ ;
10         ${s'}_i \leftarrow g(s_i)$ ;
11         $AT[({s'}_{i-1}, {s'}_i)] ++$;
12         $AS[({s'}_i)] ++$;
13         if $h(s_{i-1}, s_i) == True$ then
14             $MT[({s'}_{i-1}, {s'}_i)] ++$;
15             $MS[({s'}_i)] ++$;
16 for $(s_{i-1}, s_i) \in AT$ do
17     $\mathcal{E}_\gamma(s_{i-1}, s_i) \leftarrow \frac{AT(s_{i-1}, s_i)}{\sum_j AT[(s_{i-1}, s_j)]}$;
18 for $s_i \in ST$ do
19     $\mathcal{E}_\tau(s_i) \leftarrow \frac{MS[(s_i)]}{AS[(s_i)]}$;
20 return $\mathcal{E}(\gamma, \tau)$;
```

*1) **Memorization state|trace distribution construction**:* We calculate two probabilities representing the memorization state probability $Pr(s_i)$ and trace probability $Tr(s_{i-1}, s_i)$. To compute the memorization state probability, we count the number of times a memorization state is visited by any sequence (memorization sequence and non-memorization sequence) as the denominator and the number of times a memorization state is visited by the extracted memorization sequences from Subsection V-A as numerator. Trace probability $Tr(s_{i-1}, s_i)$ refers to how likely state s_{i-1} reaches state s_i. In Algorithm 2, we calculate two such probabilities for each sentence in Lines 16 to 19. For example, the concrete state corresponding to "home" is visited by a total of 100 memorization sequences and a total of 300 sequences. Therefore, the probability of memorization to a concrete state (memorization state) corresponding to "home" is 1/3 (100/300).

*2) **Construction of memorization sequence distribution**:* We calculate the memorization sequence probability based on $Pr(s_i)$ and $Tr(s_{i-1}, s_i)$. For a given sequence l consisting of n words, we can extract n states $\boldsymbol{s}$ corresponding to each word. Based on the chain rule and first-order assumptions, the memorization probability of the given l can be computed as:

$$Pr(\mathbf{s}) = \prod_{i=1}^{n} Tr(s_{i-1}, s_i) * Pr(s_i) \tag{4}$$

where $T_r(s_0, s_1) = 1$. In the rest of this paper, we refer to the first-order Markov memorization model as a semantic model.

VI. Addressing Memorization Issues Using the Memorization Model

Finally, we leverage our first-order Markov memorization models that are based on the last step to address the memorization issues. In particular, our approach first automatically assesses the risk of data leakage due to memorization issues. Next, our approach assists in the dememorization of the neural language models.

A. Data leakage risk assessment

A language model potentially poses the risk of remembering unintended information from its training data. To assess the training data leakage risk, we predict whether a sequence from the test data exists in the training data based on our first-order Markov memorization model.

In the first step, for each sentence in the test data, we extract the initial states based on the state extraction approach presented in Subsection IV-A. It is rare to have two identical semantic states from training and test data in an *LSTM* network. Therefore, we map each state of test data to the closest state extracted from the training data by searching the nearest neighbor based on cosine distance.

Second, we connect all the consecutive semantic states to form a sequence. We use the first-order Markov model to calculate the memorization probability of each sequence. If the memorization sequence has a high probability, we consider that the sequence would exist in the training data, resulting in a possible data leakage. We use such uncovered possible data leakage to assess the memorization issues from the original neural language models.

B. Assisting in dememorization

To assist in dememorization, we mutate the sentences in the training data that are most likely to lead to data leakage and re-build our semantic model to know whether the mutation mitigates the unintended memorization behavior. The goal of our approach is to mutate the data-leaking sentences while minimizing the impact on the data without leakage risks. For each sentence, we leverage the memorization probability that is generated from our approach to decide whether to mutate the sentence. In short, we only mutate the sentences with high memorization probability and retrain the neural language model from the data after mutation for dememorization.

VII. Evaluation

A. Experimental setup

We evaluate our approach based on one of the state-of-the-art word level *LSTM*-based language models [19] with 3,000 hidden nodes on three popular large datasets, namely, WikiTest-103 [16], WMT2017-en [17], and IWSLT2016-en [18]. An overview of these datasets is given in Table I. The training data is disjoint from the test data. Our experimental environment is based on a server with 16 24GB-GPUs, 500 GB of RAM, and 1 TB disk. The server runs Ubuntu Linux, version 20.04. Table II shows the runtime of each stage of our proposed approach over different datasets. Adding a regularization setup parameter, each memorization-analysis-oriented model only needs to be constructed once to assess the data leakage of one AI model.

TABLE I: Overview of our datasets

Dataset		Sentences	Unique Words
WikiText-103	Train	1M	220K
	Test	100K	220K
WMT2017-en	Train	4M	798K
	Test	12K	40K
IWSLT2016-en	Train	177K	59K
	Test	19K	15K

TABLE II: Overview of time cost for each step

	Sem. profiling	Dim. reduction	Sem. clustering	Mem. abstraction	Sem. mem. modeling
W-103	0.25h	0.15h	4h	1.5h	0.1h
WMT	0.55h	0.62h	15h	5.8h	0.3h
IWSLT	0.08h	0.08h	1h	0.7h	0.07h

Sem. is abbreviation of semantic. Mem. is abbreviation of memorization.

B. Preliminary analysis

Given a language model, if the memorization data appears to have no inherent common patterns or characteristics, the data would not be prone to data leakage issues, i.e., would not be suitable to our study. Therefore, before applying our approach to the three neural language models from the three datasets, we aim to understand the characteristics of the memorization sequences in the three neural language models.

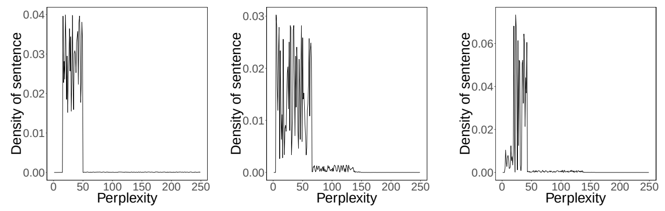
Carlini *et al.* [27] find that a sentence with low *perplexity* is likely to be vulnerable to encounter an attack involving data leakage, where *perplexity* indicates how well a trained language model fits the distribution of sentences. It is defined as the inverse probability of the sentences, normalized by the number of words. Formally, given a sequence $l = W_1^N$, the perplexity is defined as follows:

$$\begin{aligned} PP(W_1^N) &= P(w_1 w_2 w_3 ... w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i | w_1 w_2 ... w_{i-1}))}} \end{aligned} \tag{5}$$

where w_i is the i-th word in this sequence. P indicates the probability of a sentence. From Equation 5, a lower perplexity value indicates a better performing language model. We summarize the perplexity distribution over each sentence in the training data. If a model assigns a high probability to a sentence, it is likely that the model tends to remember this sentence. Therefore, we also study the relationship between perplexity and the length of a memorization sequence in each sentence.

Result: Most of the sentences in the training data have low perplexity. Figure 2 shows the perplexity distribution over the three training datasets, WikiTest-103 (a), WMT2017-en (b), and IWSLT2016-en (c). Prior studies [35], [36] report that a language model with a perplexity below 100 is considered a well-performing model. In particular, considering the prior study [35] using the same training data, the authors report that their language model achieves a perplexity of 34.4 for WikiText-103. We find that most of the sentences in the training data have low perplexity. In particular, at least 96% of the sentences have a perplexity less than 100 in our three experimental datasets.

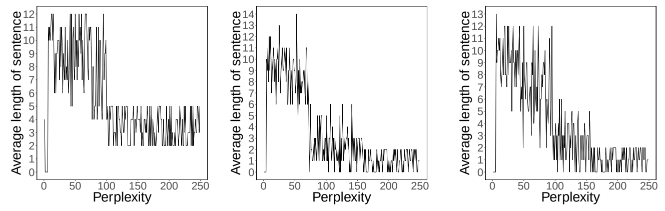
Such a result implies that the trained language model can remember most of the sentences from the training data.



(a) WikiText-103. (b) WMT2017-en. (c) IWSLT2016-en.

Fig. 2: Density distribution of number of sentences over perplexity.

The sentences with a longer memorization sequence have lower perplexity. Figure 3 shows the density of the length of memorization sequences in terms of perplexity over the training data. The X-axis indicates the perplexity (with increasing steps of 50). The Y-axis shows the density of length of memorization sequences. Note in Figure 2 and Figure 3 that most of the memorization sequences with low perplexity (< 50) contain at least six words. Such results imply that the sentences in the training data that have longer memorization sequences are easier to be remembered by the language model.



(a) WikiText-103. (b) WMT2017-en. (c) IWSLT2016-en.

Fig. 3: The average length of memorization sequence distribution in terms of perplexity over three datasets.

Summary of preliminary analysis: Most of the sentences in the studied datasets have low perplexity, which shows that the subject neural language model may be prone to the memorization issue.

C. Results

RQ1: To what extent are the studied neural language models prone to memorization issues?

Motivation: In our preliminary analysis, our results show that most of the sentences in the studied datasets have low perplexity and such sentences with low perplexities may be prone to be remembered by neural language models. As such, one can model the memorization distribution and exploit the learned memorization to extract and store the valuable training data. Therefore, in this research question, we want to explore to what extent the studied neural language models are prone to memorization issues.

Approach: To answer RQ1, we first want to know the prevalence of potential memorization issues in our studied datasets. If a state|trace is a memorization state|trace, such a state|trace is a potential memorization issue. To quantify the potential memorization issue, we define two metrics, *SCR* and *TCR*, to evaluate our memorization-analysis-oriented model. *SCR* is the memorization state coverage rate and *TCR* is the memorization trace coverage rate. The name state|trace memorization implies that the state|trace is visited by a memorization sequence. Formally, *SCR* is defined as $\frac{Num\_MS}{Num\_state}$, and *TCR* is defined as $\frac{Num\_MT}{Num\_trace}$. Num_MS is the number of distinct memorization states and Num_MT is the number of distinct memorization traces. Num_state and Num_trace refer to the total of distinct concrete states and traces, respectively. We follow the following steps to calculate the two metrics, *SCR* and *TCR*. We first apply the proposed modeling approach in Section IV to obtain the memorization-analysis-oriented model from training data. Second, we employ the memorization extraction approach from Section V-A to extract the memorization sequences from training data. Next, for each word in a memorization sequence, we can map it to the semantic model to obtain the memorization state and trace.

Memorization states|traces can be visited by both memorization sequences and non-memorization sequences. The more memorization sequences visit a state|trace, the more likely such a state|trace is prone to memorization issues. Therefore, we also quantify the memorization issues of our studied datasets using memorization state and trace probability. We calculate memorization state and trace probabilities using the approach presented in Section V-B. The higher the memorization state|trace probabilities are, the more possible such a state|trace is prone to memorization issues.

Result: Only a small portion of states and traces from training data are related to memorization. The result of the state and trace coverage rate is shown in Table III. In the table, the column σ is the input of the clustering algorithm DBSCAN used to control the granularity of clusters. The result shows that most of the states and traces are unrelated to memorization. The state coverage rate ranges from 6.8% to 24.5%. The traces coverage rate is less than 4.03% in any of different inputs of core σ. The results show that only a small percentage of states and traces are related to memorization. Such results imply that either 1) there are only a few memorization issues or 2) there exist many memorization issues, and such memorization issues only cover a small percentage of memorization states/traces.

In addition, our approach can efficiently reduce the number of initial states and traces. For example, when using a σ of 100 as input for our clustering algorithm DBSCAN, we reduce the initial millions states into 40,121 concrete states. Such a considerable number of states cannot only be used to analyze memorization behavior of a language model, but can also be used to retain most of the semantic information.

The memorization states and traces have a considerable high memorization probability. Figure 4 shows the results of the probability distribution of the memorization states and traces over the three studied datasets. Although only low percentages of states (an average of 17.6%) and traces (an average of 2.24%) are related to memorization, the memorization state and trace probabilities are comparably

TABLE III: Results of memorization state and trace coverage rate. (Mem. is the abbreviation for “memorization“.)

Dataset	σ	All concrete states	All concrete traces	Mem. states	Mem. traces	TCR	SCR
W-103	100	40,121	31,9450	3,258	6,740	1.02%	16.8%
	150	79,820	521,460	18,518	16,060	3.08%	23.2%
	200	82,317	613,419	16,792	11,593	1.89%	20.4%
	250	89,012	634,210	17,534	9196	1.45%	19.7%
WMT	100	63,902	549,872	7,221	6,653	1.21%	11.3%
	150	71,921	673,219	17,620	13,666	2.03%	24.5%
	200	76,709	778,895	12,426	14,721	1.89%	16.2%
	250	77,101	792,015	5,244	6,256	0.79%	6.8%
IWSLT	100	4,523	26,217	557	738	2.81%	12.3%
	150	8,945	114,084	1,923	4,598	4.03%	21.5%
	200	11,219	139,930	2,546	4,886	3.49%	22.7%
	250	14,234	178,904	2,246	5,831	3.26%	15.8%

high. Especially, the mean memorization state probability in dataset WikiText-103 is 0.63. By inspecting our results, we find that the memorization-analysis-oriented model can identify the memorization transition of the *LSTM*-based language model and discover potential memorization issues in the training data.

Answer to RQ1: Only a small percentage of states and traces from training data are related to memorization. However, the memorization states and traces have a considerably high memorization probability.

RQ2: How accurate is our approach in the data leakage risk assessment?

Motivation: In RQ1, the results show that the memorization states and traces tend to be remembered due to a considerable high memorization probability. The associated distribution can be used to analyze the training data and the potential for data leakage. In order to illustrate a practical impact, we leverage our approach to assess training data leakage risk based on a given language model. In this research question, we want to answer how accurate our privacy risk assessment approach is.

Approach: In Section V, we have built a first-order Markov memorization model. To realistically assess the privacy risk of given data, we use the constructed model to measure the memorization probability of each sequence in the test data. Based on the predicted memorization probability of each sequence in the test data, which is not seen by the model during the training phase, we predict whether a sequence of test data likely exists in the training data.

Furthermore, the length of a memorization sequence might affect the modeling analysis. For example, one may argue that the shorter a memorization sequence is, the more likely the sequence appears in the training data. Therefore, we calculate the Pearson correlation [37] between the length of memorization sequences and memorization probabilities of sequences. Pearson correlation ranges from -1 to +1. A value of 1 indicates that the length and memorization probability of sequences has a strong relationship. A value of 0 indicates that there is no relationship between them, and a value of -1 indicates an inverse relationship between them.

We implement a baseline approach that assigns a random score to each of the extracted memorization sequences. We compare *DeepMemory* to the baseline in this research question. To measure the performance, we examine whether the extracted sequences from the test data appear in the training data. If a sequence is indeed in the training data, we consider it as a true-positive sequence. Otherwise, it is a false-positive sequence. The true-positive sequence is considered to be data leakage from training data. We use four metrics to evaluate our approach, including *precision*, *recall*, *F1*, and *AUC*. *Precision* measures the correctness of our model, and refers to the ratio of cases when a predicted sequence is actually in the training data. *Recall* measures the completeness of our approach, and is defined as the number of sequences that were correctly predicted as memorization divided by the total number of memorization sequences in the test data. *F1* is the harmonic mean of *precision* and *recall*. *AUC* allows us to measure the overall ability of our approach. The *AUC* is the area under the ROC curve, which indicates the performance of a binary model as its discrimination is varied.

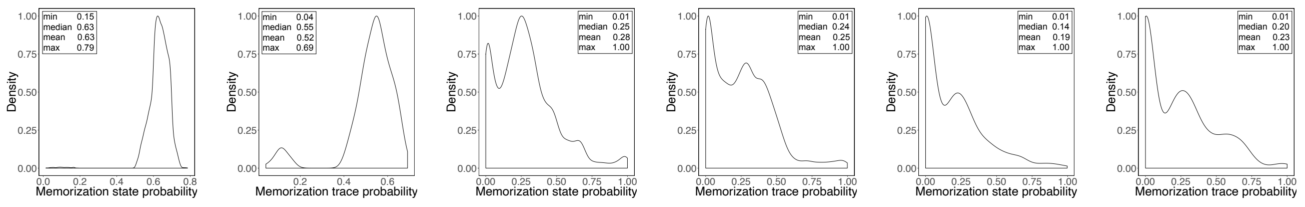
Result: Our data leakage assessment approach can achieve an average AUC of 73%. Table IV shows the results for precision, recall, F1, and AUC over the memorization distribution. Note from Table IV that our approach achieves an average precision of 47% and a very high average recall of 92% when taking 0.5 as a threshold, which outperforms the baseline approach, i.e., a precision of 0.38 and a recall of 56%. The results imply that a sequence with a high memorization probability in the test data tends to be memorized. However, different thresholds may lead to different results. To overcome this bias, we also present the AUC of our approach. We find that the AUC is high with an average value of 73%. The results suggest that our proposed first-order memorization Markov model approach is capable of assessing data leakage risks.

TABLE IV: Results of using our approach to predict the memorized sequence compared with the baseline approach.

	DeepMemory				Baseline			
	Precision	Recall	F1	AUC	Precision	Recall	F1	AUC
W-103	0.50	0.75	0.60	0.72	0.38	0.50	0.42	0.48
WMT	0.29	1.00	0.44	0.67	0.30	0.57	0.40	0.50
IWSLT	0.62	1.00	0.76	0.80	0.50	0.60	0.54	0.48
Average	0.47	0.92	0.60	0.73	0.39	0.56	0.45	0.49

Our approach shows a similar performance for all types of sequences. The Pearson correlation between length and memorization probability of memorization sequence is 0.14. An absolute value of 0-0.19 is regarded as a very weak correlation [37]. Therefore, a very weak relationship exists between the length of a memorization sequence and the memorization probability of sequences.

Our approach can be used to efficiently identify a real-world data leakage issue. In order to demonstrate the practical usefulness of our approach, we want to examine whether our approach can be used to identify real-world private data. We train a language model based on the setting from [8]. Similar to the prior work [8], we make the trained language model



(a) WikiText-103 state. (b) WikiText-103 trace. (c) WMT2017 state. (d) WMT2017 trace. (e) IWSLT2016 state. (f) IWSLT2016 trace.

Fig. 4: Memorization states and traces probability distribution.

remember the sequence "the credit number is 281265017". After that, we analyzed this language model based on our proposed approach. During the testing phase, we test our semantic Markov memorization model on a set of sentences with the same structure but different credit numbers. We find that the sentence "the credit number is 281265017" has the highest memorization probability. Note that the prior work has reported that memorization is not overfitting [8]. The result suggests that our proposed model can efficiently detect the memorization content from the training data.

Answer to RQ2: Our data leakage assessment approach can achieve an average AUC of 73%. Our approach shows a similar performance for all types of sequences. Our approach can be used to efficiently identify real-world private data.

RQ3: How effective is our approach in assisting dememorization?

Motivation: One may randomly select sentences and mutate them to reduce memorization sequences probability (see Equation 4). However, it is not an optimal solution to mutate a large portion of the training data since the mutation would hurt the quality of the data, leading to unrealistic models. On the other hand, if one only randomly mutates a small portion of the training data, the mutated data may not contain memorization issues. In this research question, we want to evaluate whether our approach can assist in dememorization by suggesting only a small portion of data in the training data to be mutated.

Approach: We compare the use of our approach in assisting dememorization to a random baseline approach. We first apply our approach to detect the memorization sequences from the training data and to select memorization sequences. The results of RQ2 show that when using 0.5 as the threshold to predict memorization sequence, our recall is very high (close to 1). Therefore, we select memorization sequences to be mutated if their probabilities are more than 0.5. For the random approach, we randomly select 50% of all the memorization sequences to be mutated. We choose 50% for the baseline approach in order to give the baseline approach an overestimated ability of mutating the training data. 50% also ensures that at least half of the existing training data is not mutated. In both experiments, we ignore memorization sequences with lengths less than four.

Second, we use four strategies to mutate the aforementioned selected sequences from the original training data to mitigate unintended memorization behavior.

- **REPlacing Word (REPW):** For each extracted sequence, we first select the noun and verbal phrase that occurred less frequently. We then replace the selected words with their synonyms in the training data randomly. If there are no synonyms in the training data, we replace them with a random external synonym. Next, we modify the corresponding sentences that contain mutated sequences.
- **REOrdering Sequence (REOS):** Prior research [2], [36] shows that sequence disorder can benefit the robustness of a sequential model in machine translation tasks and industrial recommendation system applications. This strategy aims to reorder words in memorization sequences to confuse the language models.
- **REMoving Word (REMW):** For the sentences that contain memorization sequences, we remove those sequences directly from the sentences.
- **MIXture (MIX)**: Different strategies may have their advantages. In the mixture strategy, we combine the replacing words and reordering sequences approaches.

Next, we re-train a language model based on the mutated training data and re-build our semantic first-order Markov memorization model. Finally, we use our semantic model to analyze the memorization behavior of the re-trained neural language model on the original training data. In particular, we extract the memorization sequences of re-trained neural language models. We then calculate how many memorization sequences in the original model (before mutation) still exist in the re-trained model. The fewer memorization sequences that are left, the better dememorization the re-trained model has. We also calculate the number of mutated memorization sequences from both our approach and the random baseline. The desired approach would achieve a low number of memorization sequences that are left in the re-trained model, while only having to mutate a small percentage of memorization sequences.

Result: Our approach can assist in dememorization without the need to mutate a large number of memorization sequences. Table V shows the results for memorization sequence statistics after re-training the language model using different strategies to mutate the training data. With assistance from our approach, the memorization sequences can be significantly reduced. Table V shows that, compared to the original memorization sequences, the percentages of the memorization sequences drop to 2.58%, 2.31%, and 4.43% in WikiText-103, WMT2017, and IWSLT2016, respectively. Compared to our ap-

TABLE V: Total number of original memorization sequences and the number of memorization sequences after dememorization assisted by our approach and the baseline approach.

Dataset	Measure	Original	Mutated Sequence (%)		after REPW		after REOS		after REMW		after MIX		Average	
			Prob.>0.5	Random	Prob.>0.5	Random	Prob.>0.5	Random	Prob.>0.5	Random	Prob.>0.5	Random	Prob.>0.5	Random
W-103	Mem. Seq.	59,802	4.10%	50%	1,645 (2.75%)	3,519 (5.88%)	1,543 (2.58%)	4,210 (7.04%)	1,549 (2.59%)	2,431 (4.07%)	2,021 (3.38%)	3,979 (6.65%)	1,690 (2.83%)	3,299 (5.91%)
WMT	Mem. Seq.	124,319	0.89%	50%	4,210 (3.39%)	3,577 (2.88%)	2,874 (2.31%)	2,576 (2.07%)	3,121 (2.51%)	1,498 (1.20%)	2,989 (2.40%)	2,249 (1.81%)	3,299 (2.65%)	2,475 (1.99%)
IWSLT	Mem. Seq.	18,753	2.80%	50%	2,091 (11.15%)	3,202 (17.07%)	2,484 (13.25%)	3,389 (18.07%)	831 (4.43%)	1,034 (5.51%)	1,701 (9.07%)	2,214 (11.81%)	1,777 (9.47%)	2,460 (13.12%)

Original is the number of memorization sequences in the original model. Mutated sequence means the percentage of memorization sequences to be mutated. Columns starting with "after" mean after mutating the training data, the number of memorization sequence that are left and the corresponding percentage.

proach, the average of percentages of memorization sequences that are left after the mutation from the random baseline are 5.91%, 1.99%, and 13.12% in WikiText-103, WMT2017, and IWSLT2016, respectively. Except for WMT2017, where both approaches show a similar performance in reducing the memorization sequences, our approach outperforms the baseline approach by a wide margin.

Our approach only needs to mutate a very small number of sequences from the training data. Table V shows the number of memorization sequences that are mutated during the dememorization process. The results illustrate that our approach only mutates 4.1%, 0.89%, and 2.8% of the original memorization sequences in the datasets WikiText-103, WMT2017, and IWSLT2016, respectively. Such a small number of mutations would have a trivial impact on the trained model. By definition, the baseline approach mutates 50% of the memorization sequences, i.e., a very large amount of mutation, and cannot even achieve a comparable dememorization result.

Answer to RQ3: Our approach is capable of guiding dememorization and does not decrease the performance of the original model. Therefore, practitioners can use our approach to discover sensitive data leakage risks and help mitigate memorization.

VIII. Comparative study on the effect of regularization

In this section, we discuss the impact of regularization on the memorization effect. Regularization is an efficient approach to train neural network based models. Although a prior study [8] shows that memorization in neural language models is not an issue of overfitting, the use of regularization may still potentially affect the memorization behavior of neural language models. Therefore, we conduct a comparative study over four mainstream regularization techniques, including dropout, L1 norm, L2 norm regularization and data augmentation (DA).

We build an original model without any regularization. To evaluate the impact of the regularization techniques, we create four additional models by modifying our original model by altering only one regularization technique, including enabling: 1) dropout, 2) L1 norm, 3) L2 norm, and 4) DA. In particular, the augmentation is to randomly select 10% of the sentences from the training corpus and randomly replace non-stop words with one of their synonyms [38].

We follow a process similar to RQ1 to conduct our comparative study. In particular, our experiment is executed with σ in 200. We first calculate two metrics *SCR* and *TCR* from the four additional models while altering the regularization techniques. We then calculate their corresponding memorization state and trace probabilities. Finally, we compare the results for the four additional models with the results from our original models.

Results: Regularization may be able to mitigate the memorization effect. The results (with and without regularization) are shown in Table VI. The results show that without regularization, the memorization state coverage rate ranges from 23.1% to 31.4% and the memorization trace coverage rate ranges from 7.43% to 11.32%. After regularization, both the memorization state and the trace coverage rate decrease considerably. Especially, the L2 norm regularization provides the highest reduction in the memorization state and trace coverage (19.8% and 1.89%, respectively).

In addition, we compare the memorization state and trace probability distribution of the above four additional models with the ones from the original models, using the Mann-Whitney U test and Cliff's delta. We find that all of the probability distributions of the four additional models are different with statistical significance ($p < 0.05$) from the original model. However, the difference may differ among different subjects. In particular, for WMT, the original models (without regularization) always have a higher memorization probability than the four additional models (positive effect sizes). For IWSLT and W-103, the differences are associated with rather negligible or small effect sizes; while cases also exist where the probability distribution is lower with regularization (e.g., W-103; enabling dropouts). Such results suggest that regularization (in particular, the L2 norm) may be useful to partially address memorization issues, but they cannot be eliminated. More comparative work is required to highlight the relative impact of the different approaches.

IX. Threats to Validity

External validity. A threat to the external validity is the generalizability of our approach. Our study is evaluated on the most popular neural language model, i.e., the *LSTM*-based language model, and three specific public datasets. More case

TABLE VI: Results of memorization coverage rate with and without regularization (Reg. means regularization).

Dataset	Reg.	All concrete states	All concrete traces	Mem. States	Mem. Traces	TCR	SCR
W-103	Original	81,790	631,521	22,820	54,248	8.59%	27.9%
	Dropout	83,210	647,932	18,639	16,003	2.47%	22.4%
	L1	82,123	627,984	19,545	16,076	2.56%	23.8%
	L2	80,789	642,983	17,531	12,152	1.89%	21.7%
	DA	84,198	852,129	21,883	61,609	7.23%	26.0%
WMT	Original	78,256	823,943	18,077	93,270	11.32%	23.1%
	Dropout	72,198	878,134	13,212	18,528	2.11%	18.3%
	L1	79,821	849,702	13,729	31,863	3.75%	17.2%
	L2	76,213	851,203	15,090	23,578	2.77%	19.8%
	DA	77,678	812,323	17,656	81,232	10.01%	22.7%
IWSLT	Original	14,232	176,820	4,468	13,137	7.43%	31.4%
	Dropout	11,950	122,561	3,274	5,172	4.22%	27.4%
	L1	13,212	119,821	2,893	3,582	2.99%	21.9%
	L2	12,792	98,996	2,533	4,237	4.28%	19.8%
	DA	13,341	15,421	3,867	1,076	6.98%	28.9%

studies on other datasets in other neural network based language models can benefit the evaluation of our approach.

Internal validity. Our work uses several techniques, such as the clustering algorithm DBSCAN, the dimension analysis algorithm PCA, and the First-Order Markov model. Such techniques can be replaced by other kinds of similar techniques. For example, DBSCAN can be replaced with the k-means clustering algorithm. Our approach also leverages threshold values, for example, the σ and ρ of the DBSCAN. To explore the impact of these choices, we individually increased or decreased the σ (omitted due to limited space) and ρ (see Table III) values in our experiment.

Construct validity. In the evaluation of our approach for dememorization, we only used four strategies to mutate the training data. Similar evaluation approaches based on mutation techniques have been often used in prior research [39]. However, there may exist other kinds of strategies to mutate the training data. Future work can complement our evaluation.

X. Related work

Analysis of DNN. Many prior studies [40], [41], [42], [43], [44], [45], [46], [14], [15] have been proposed to analyze and explain the behaviors of deep neural network. Functional analysis and decision analysis are two main categories of analysis of DNN [47]. Functional analysis, i.e., black-box analysis, aims to capture the overall behavior by investigating the relation between inputs and outputs [41], [43], [48]. Decision analysis takes the DNN as a white box and analyzes the internal behavior by profiling internal structures and component rolls [14], [15], [40].

In our study, we focus on the decision analysis, i.e., internal behavior analysis. One of the typical techniques used to analyze the internal behavior of a DNN model is Finite State Automation (FSA) [49], [30]. FSA consists of states and transitions, which can be mapped to the behavior of sequence models. Du *et al.* [14] use an interval-based approach to cluster the original hidden-state vector which produces comparable performance under a scalable environment.

Prior studies focus on the analysis of behavior of the RNN model and its variance in FSA for the natural language processing task. However, there is a lack of work on memorization issues for language models. Our paper is the first work on analyzing, detecting and assisting in repairing memorization issues of RNN models.

General privacy of DNN. Extensive prior research has revealed serious privacy issues posed by deep neural networks as the data used for training can be leaked [50]. In general, privacy threats of the deep neural network can be divided into the two categories of direct and indirect information exposure hazards [51]. Direct privacy data leakage is mainly due to the data curator [52], [53], untrusted communication link [54] and untrusted cloud [55]. In terms of the indirect privacy threat, one would like to infer or guess information for training data or model parameters without access to the actual data [56]. Many prior studies [9], [10], [11] have reported that deep neural networks tend to memorize the training data instead of learning the latent properties of the training data. Some studies [10], [57], [58], [59] propose automatic techniques that infer whether a given data instance has contributed to the target model. Shokri *et al.* [57] propose the first membership inference attack to deduce whether a data record is used in the training process for the targeted model. The core idea is to distinguish a given record in terms of the confidence score output by the targeted model. In addition to membership inference, research also aims to infer sensitive attributes for a released model [50], [60], [61] and to steal model parameters [56], [62], [63], [64].

Prior studies develop attacks and defenses for investigating various privacy challenges. Different from previous work, we consider a privacy breach related to memorization in neural language models and analyze memorization via abstracted hidden states from the extracting finite state machine. Our approach aims to address privacy issues during the quality assurance process for developing AI models, instead of defending against such attacks after the fact. Our work contributes to the area of general privacy of deep neural networks.

XI. Conclusion

This paper proposes *DeepMemory*, a novel approach for analyzing the internal memorization behavior in language models. We construct a memorization-analysis-oriented model and build a semantic first-order Markov model to analyze memorization distribution. We evaluate our approach based on one of the most popular neural language models, the *LSTM*-based language model with three public datasets, namely, WikiText-103, WMT2017, and IWSLT2016. The results show that using our approach, we can address memorization issues by automatically identifying data leakage risks with an average AUC of 0.73. Based on the assessment results, our approach can assist in dememorization by only mutating a very small percentage (4.1%, 0.89% and 2.8%) of the training data to reduce the memorization in the neural language models. Our work calls for future research to address the privacy issues in neural language models.

XII. Acknowledgements

We would like to thank Thomas Vannet for his insightful feedback on an earlier version of this work. Furthermore, we thank the anonymous reviewers for their valuable comments.

References

[1] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. The Association for Computational Linguistics, 2015, pp. 379–389.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] M. Pak and S. Kim, "A review of deep learning in image recognition," in *2017 4th international Conference on Computer Applications and Information Processing Technology (CAIPT)*. IEEE, 2017, pp. 1–3.

[4] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[5] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[6] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift," *Advances in Neural Information Processing Systems*, vol. 32, pp. 13 991–14 002, 2019.

[7] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[8] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2019, pp. 267–284.

[9] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio *et al.*, "A closer look at memorization in deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 233–242.

[10] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Towards demystifying membership inference attacks," *arXiv preprint arXiv:1807.09173*, 2018.

[11] C. Meehan, K. Chaudhuri, and S. Dasgupta, "A non-parametric test to detect data-copying in generative models," *arXiv preprint arXiv:2004.05675*, 2020.

[12] S. Ovaska, "Data privacy risks to consider when using AI." [Online]. Available: https://www.fm-magazine.com/issues/2020/feb/data-privacy-risks-when-using-artificial-intelligence.html

[13] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 739–751.

[14] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: Model-based quantitative analysis of stateful deep learning systems," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT, FSE)*, 2019, pp. 477–487.

[15] X. Zhang, X. Du, X. Xie, L. Ma, Y. Liu, and M. Sun, "Decision-guided weighted automata extraction from recurrent neural networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2021, pp. 11 699–11 707.

[16] T. Wolf, Q. Lhoest, P. von Platen, Y. Jernite, M. Drame, J. Plu, J. Chaumond, C. Delangue, C. Ma, A. Thakur, S. Patil, J. Davison, T. L. Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, S. Brandeis, S. Gugger, F. Lagunas, L. Debut, M. Funtowicz, A. Moi, S. Rush, P. Schmidd, P. Cistac, V. Muštar, J. Boudier, and A. Tordjmann, "Datasets," *GitHub. Note: https://github.com/huggingface/datasets*, vol. 1, 2020.

[17] O. Boja *et al.*, *2017 Second Conference on Machine Translation (WMT17): Proceedings*. Association for Computational Linguistics, 2017.

[18] "IWSLT evaluation 2016," https://sites.google.com/site/iwsltevaluation2016/iwslt-evaluation-2016.

[19] S. Merity, N. S. Keskar, and R. Socher, "An analysis of neural language modeling at multiple scales," *arXiv preprint arXiv:1803.08240*, 2018.

[20] K. Jing and J. Xu, "A survey on neural network language models," *arXiv preprint arXiv:1906.03591*, 2019.

[21] S. Ortmanns, H. Ney, and A. Eiden, "Language-model look-ahead for large vocabulary speech recognition," in *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP)*. IEEE, 1996, pp. 2095–2098.

[22] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. Lafferty, R. L. Mercer, and P. S. Roossin, "A statistical approach to machine translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990.

[23] K.-L. Liu, W.-J. Li, and M. Guo, "Emoticon smoothed language models for Twitter sentiment analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012.

[24] F. Song and W. B. Croft, "A general language model for information retrieval," in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, 1999, pp. 316–321.

[25] K. Smagulova and A. P. James, "A survey on LSTM memristive neural network architectures and applications," *The European Physical Journal Special Topics*, vol. 228, no. 10, pp. 2313–2324, 2019.

[26] F. Mireshghallah, H. A. Inan, M. Hasegawa, V. Rühle, T. Berg-Kirkpatrick, and R. Sim, "Privacy regularization: Joint privacy-utility optimization in language models," *arXiv preprint arXiv:2103.07567*, 2021.

[27] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel, "Extracting training data from large language models," in *30th USENIX Security Symposium (USENIX Security)*. USENIX Association, Aug. 2021, pp. 2633–2650.

[28] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *arXiv preprint arXiv:1708.02182*, 2017.

[29] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[30] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5247–5256.

[31] R. Krishnan, D. Liang, and M. Hoffman, "On the challenges of learning with inference networks on sparse, high-dimensional data," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 143–151.

[32] B. Geiger and G. Kubin, "Relative information loss in the PCA," in *2012 IEEE Information Theory Workshop*. IEEE, 2012, pp. 562–566.

[33] M. Ester, H. Kriegel, J. Sander, and X. Xiaowei, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 226–231.

[34] M. Zorzi, R. R. Rao, and L. B. Milstein, "On the accuracy of a first-order Markov model for data transmission on fading channels," in *Proceedings of the 4th IEEE International Conference on Universal Personal Communications (ICUPC)*. IEEE, 1995, pp. 211–215.

[35] J. Rae, C. Dyer, P. Dayan, and T. Lillicrap, "Fast parametric learning with activation memorization," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 4228–4237.

[36] Q. Jia, N. Zhang, and N. Hua, "Context-aware deep model for entity recommendation system in search engine at Alibaba," *Journal of Multimedia Processing and Technologies*, vol. 11, no. 1, pp. 23–35, 2020.

[37] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Springer, 2009, pp. 1–4.

[38] M. Bayer, M.-A. Kaufhold, and C. Reuter, "A survey on data augmentation for text classification," *arXiv preprint arXiv:2107.03158*, 2021.

[39] C. Auerbach, *Mutation research: problems, results and perspectives*. Springer, 2013.

[40] A. L. Cechin, D. Regina, P. Simon, and K. Stertz, "State automata extraction from recurrent neural nets using k-means and fuzzy clustering," in *Proceedings of the 23rd International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2003, pp. 73–78.

[41] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*. Springer, 2014, pp. 818–833.

[42] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS ONE*, vol. 10, no. 7, pp. 130 – 140, 2015.

[43] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1135–1144.

[44] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, “Visualizing deep neural network decisions: Prediction difference analysis,” in *5th International Conference on Learning Representations (ICLR)*, 2017.

[45] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, “Explaining nonlinear classification decisions with deep Taylor decomposition,” *Pattern Recognition*, vol. 65, pp. 211–222, 2017.

[46] P. W. Koh and P. Liang, “Understanding black-box predictions via influence functions,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 1885–1894.

[47] A. Shahroudnejad, “A survey on understanding, visualizations, and explanation of deep neural networks,” *arXiv preprint arXiv:2102.01792*, 2021.

[48] A. Dhurandhar, P. Chen, R. Luss, C. Tu, P. Ting, K. Shanmugam, and P. Das, “Explanations based on the missing: Towards contrastive explanations with pertinent negatives,” in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018, pp. 590–601.

[49] C. W. Omlin and C. L. Giles, “Extraction of rules from discrete-time recurrent neural networks,” *Neural Networks*, vol. 9, no. 1, pp. 41–52, 1996.

[50] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, “Privacy in pharmacogenetics: An end-to-end case study of personalized Warfarin dosing,” in *23rd USENIX Security Symposium (USENIX Security)*, 2014, pp. 17–32.

[51] X. Liu, L. Xie, Y. Wang, J. Zou, J. Xiong, Z. Ying, and A. V. Vasilakos, “Privacy and security issues in deep learning: A survey,” *IEEE Access*, vol. 9, pp. 4566–4593, 2021.

[52] McAfee, “Grand theft data – Data exfiltration study: Actors, tactics, and detection,” 2015.

[53] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, “Spectre attacks: Exploiting speculative execution,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[54] S. Sodagudi and R. R. Kurra, “An approach to identify data leakage in secure communication,” in *Proceedings of 2nd International Conference on Intelligent Computing and Applications*. Springer, 2017, pp. 31–43.

[55] C. Warzel, “Faceapp shows we care about privacy but don’t understand it,” https://www.nytimes.com/2019/07/18/opinion/faceapp-privacy.html, (Accessed on 03/11/2021).

[56] R. J. Bolton, D. J. Hand *et al.*, “Statistical fraud detection: A review,” *Statistical science*, vol. 17, no. 3, pp. 235–255, 2002.

[57] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, “Membership inference attacks against machine learning models,” in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

[58] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, “White-box vs black-box: Bayes optimal strategies for membership inference,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 5558–5567.

[59] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, “Privacy risk in machine learning: Analyzing the connection to overfitting,” in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 268–282.

[60] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015, pp. 1322–1333.

[61] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton, “A methodology for formalizing model-inversion attacks,” in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, 2016, pp. 355–370.

[62] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction APIs,” in *25th USENIX Security Symposium (USENIX Security)*, 2016, pp. 601–618.

[63] T. Orekondy, B. Schiele, and M. Fritz, “Prediction poisoning: Towards defenses against DNN model stealing attacks,” in *8th International Conference on Learning Representations, (ICLR)*, 2020.

[64] M. Yan, C. W. Fletcher, and J. Torrellas, “Cache telepathy: Leveraging shared resource attacks to learn DNN architectures,” in *29th USENIX Security Symposium (USENIX Security)*. USENIX Association, Aug. 2020, pp. 2003–2020.