Towards Trustworthy LLMs: Understanding the Security and Privacy Risks of Large Language Models

Kang Gu

Dartmouth College, kang.gu.gr@dartmouth.edu

Follow this and additional works at: https://digitalcommons.dartmouth.edu/dissertations

Recommended Citation
https://digitalcommons.dartmouth.edu/dissertations/231

This Thesis (Ph.D.) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Ph.D Dissertations by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.
Towards Trustworthy Large Language Models: Understanding the Security and Privacy Issues of LLMs

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
by Kang Gu

Guarini School of Graduate and Advanced Studies
Dartmouth College
Hanover, New Hampshire

April 2024

Examining Committee:

(Chair) Shagufta Mehnaz

Soroush Vosoughi
Yujun Yan
Mehrdad Mahdavi

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies
Abstract

Recent years have witnessed remarkable breakthroughs achieved by large language models (LLMs) in natural language processing. As a result, many industries (e.g., tech, finance) have quickly embraced LLMs, which are poised to transform how we create content, search for information, and more. Although LLMs are more powerful than any predecessors (e.g., CNN, RNN), they have also raised security and privacy issues about adversarial robustness as well as the leakage of sensitive information contained in the expansive training corpus.

Firstly, we design a novel adversarial example attack against masked-language models to evaluate their robustness against malicious perturbations on input text. Differently from previous methods, our attack generates examples that are highly semantically similar to the original text, which are more indistinguishable from human eyes. Secondly, we examine the potential risks of model inversion attacks against LLMs. Our findings indicate that an adversary can reconstruct similar input sentences from the leaked embeddings. In the trend of finetuning LLMs on company-owned user datasets, sensitive information such as home addresses and health records is at risk of being exposed. Finally, there is no straight forward way to delete training samples from LLMs. To meet data protection regulations such as GDPR, the model should either be re-trained or unlearned to remove the impacts of certain user data, upon users’ request of removal. Given training LLMs from scratch is computationally intensive, the more efficient machine unlearning methods are gaining increasing attention. We extensively evaluate the behaviors of LLMs under various unlearning methods and reveal the trade-offs between removal effects and model utility preservation.
3.2.5 Sentence Reconstruction ........................................... 22
3.3 General Attack Workflow ............................................ 23
  3.3.1 Attack Definition .................................................. 23
  3.3.2 Threat Model ........................................................ 24
  3.3.3 Attack Pipeline ..................................................... 24
3.4 Keyword Inference Attack ............................................ 25
  3.4.1 Attack Definition .................................................. 26
  3.4.2 Attack Settings ..................................................... 26
3.5 Sentence Inference Attack ............................................ 26
  3.5.1 Attack Definition .................................................. 27
  3.5.2 Attack Settings ..................................................... 27
3.6 Datasets ........................................................................ 27
  3.6.1 Airline ................................................................. 27
  3.6.2 Medical ................................................................. 28
  3.6.3 Data Pre-processing .................................................. 28
3.7 Evaluation Metrics ....................................................... 29
  3.7.1 Keyword Inference Attack Metrics ............................... 29
  3.7.2 Sentence Inference Attack Metrics ............................... 30
3.8 Experimental Evaluation .............................................. 31
  3.8.1 White-box Setup ..................................................... 31
  3.8.2 Black-box Setup ..................................................... 31
  3.8.3 Implementation ..................................................... 31
  3.8.4 Keyword Inference Results & Discussion ....................... 33
  3.8.5 Sentence Inference Results & Discussion ....................... 35
3.9 Potential Defenses ...................................................... 37
  3.9.1 Differential Privacy ................................................ 37
  3.9.2 Privacy Preserving Mapping ...................................... 38
  3.9.3 Avoid Providing Complete Sentence Embeddings ............ 38
  3.9.4 Evaluation of Potential Defenses ................................ 38
3.10 Discussion .............................................................. 39
  3.10.1 Practicality of Decoder-based Attack ......................... 39
  3.10.2 Limitations .......................................................... 39
  3.10.3 Future Work ......................................................... 40

4 Unlearning Large Language Models .................................... 42
  4.1 Preliminary ............................................................. 42
    4.1.1 Unlearning for General Models ................................. 42
    4.1.2 Unlearning for Convex Models ................................ 43
    4.1.3 Newton Update ..................................................... 43
    4.1.4 Gradient Ascent ................................................... 44
  4.2 Unlearning for LLMs .................................................. 44
    4.2.1 Inverse Empirical Fisher ....................................... 44
    4.2.2 Efficient GPU Implementation ................................ 45
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Difference between Poor and Tricky Adversarial Examples (AE) for an NLP Application</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Variations of Text Sample</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Comparison of our attacks (<em>LIME attack</em> and <em>Reinforce attack</em>) with existing work</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Three sets of adversarial examples generated by BERT-Attack, our methods.</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Human evaluation results on IMDB and Blog</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>The ablation study on reward function</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Runtime Comparison</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>Keyword Inference Results (<em>k = 20</em>)</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>Sentence Inference Results. Results on GPT are shown in ()</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>10 Examples of Sentence Inference Attack</td>
<td>36</td>
</tr>
<tr>
<td>11</td>
<td>Parameter Definition</td>
<td>42</td>
</tr>
<tr>
<td>12</td>
<td>The Comparison of Selected Existing Unlearning Methods</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>Unlearning Results on ARC Dataset</td>
<td>51</td>
</tr>
<tr>
<td>14</td>
<td>Unlearning Results on Piqa Dataset</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>Unlearning Results on MathQA Dataset</td>
<td>52</td>
</tr>
<tr>
<td>16</td>
<td>Time complexity of each unlearning baseline</td>
<td>53</td>
</tr>
<tr>
<td>17</td>
<td>Overall Rankings of all Unlearning Methods</td>
<td>54</td>
</tr>
<tr>
<td>18</td>
<td>Unlearning on MT and Enron Email Dataset</td>
<td>56</td>
</tr>
</tbody>
</table>
List of Figures

1. The three security and privacy issues existing in LLM service .......................... 1
2. Reinforce attack framework. \( T \) is the target model, \( S \) and \( S_a \) are original and adversarial sentences, respectively, \( Q \) is the query number, and \( P \) represents perturbation rate. Note that, in practice, we use the sorted words according to the weights. ................................. 12
3. The general workflow of our attack. EHR stands for Electronic Health Record. The adversary first queries the pre-trained language model to obtain the public sentence embeddings. Then the adversarial decoder is trained using public text/embedding pairs. Finally, the trained decoder is employed to infer the sensitive information from private embeddings. 25
4. The workflow of keyword inference attack. The adversary first repeats the decoding of the same sentence embedding 10 times to gather all potential outputs. Then the outputs are passed through a stopwords filter to eliminate stopwords. The adversary further converts the sentences into a list of words sorted by frequency. Finally, only top-\( k \) words are kept and examined. Note that \( k \) is a hyperparameter in our experiments. 25
5. .................................................................................................................... 34
6. .................................................................................................................... 38
7. The cycle of unlearning of the LLM service. The model has been trained on the user database for downstream tasks. Since user requests arrive stochastically, a cache is used to store them. Once the cache is full or the waiting time has exceeded the threshold, the server will invoke the unlearning process and clean the cache to free up space. Without the loss of generality, we assume the size of the cache is 128 in our experiments. ................................. 48
8. The runtime of each unlearning method on GPT-NEO-125M. ......................... 54
9. Privacy-utility trade-offs of DP-SGD and unlearning approaches on (a) Lambda and (b) Piqa Dataset. The model under evaluation is OPT-125M. ......................................................... 56
10. Weights distribution of the last layer of GPT-NEO (125M) fine-tuned on ARC dataset. \# of bins for plotting the histograms is set to 40k. (a), (b) and (c) display the distribution after applying Gradient Ascent, Fisher Removal, and Fisher Forgetting respectively. ................................. 59
11. Model utility curves after extended unlearning cycles. Both (a) and (b) are generated by GPT-NEO-125M model. ................................. 60
12. Onion effect on GPT-NEO-125M. In the ARC dataset, we set the cutoff threshold as 6 and remove all the samples with higher exposure scores. After retraining the model, some left samples have their exposure scores increase past the threshold. ................................. 61
1 Introduction

Recent years have seen remarkable advancements in the Natural Language Processing (NLP) domain. Following the introduction of the Transformer architecture [1], which serves as the foundation for pre-trained language models, a multitude of models have been developed and released. Presently, the HuggingFace API supports over 40,000 language models, including prominent ones like BERT [2], GPT [3–5], XLNet [6], and T5 [7]. Furthermore, transformers have recently demonstrated success in computer vision tasks as well [8, 9]. Modern language models typically employ extensive architectures with millions of parameters and undergo pre-training on vast datasets. In contrast to traditional shallow or small neural networks, pre-trained language models exhibit high levels of generalization and can serve as effective feature extractors for diverse downstream tasks. For instance, upon its release, BERT achieved state-of-the-art performance across eleven NLP tasks [10].

Although LLMs are constantly evolving due to wide attention, their privacy and security issues are concerning and still underinvestigated. As shown in Figure 1, for a commercial LLM service developed by customers’ data, the company should be aware of the potential threats posed by the adversary as well as their duty to observe data regulations such as GDPR. We mainly explore three aspects in this thesis: Firstly, the adversary can fool the LLM application for certain malicious purpose. For example, a spam can be disguised to fool a LLM-based detector such that users are left vulnerable. Secondly, the adversary can dig into sensitive user information from the company database, which may include phone numbers, home addresses, etc. Last but not least, when an user chooses to quit the service, the company should not only remove the personal data from database, but also make the model forget about the corresponding knowledge.
1.1 Adversarial Example Attack.

The purpose of an adversarial example is to confuse the target model while remaining indistinguishable from the original sample to human eyes. Compared to image data, textual data is discrete—making it more challenging to generate adversarial examples by perturbing inputs. For instance, in the image domain, pixel values could be tweaked easily with negligible impact on the image whereas tweaking the textual data is limited to a vocabulary of words. Nevertheless, in order to develop effective defense methods against textual adversarial examples, the important first step is to thoroughly investigate the extent and strength of such attacks. However, due to the challenge mentioned above with textual data, a thorough investigation requires more effort compared to, e.g., the image domain. Moreover, besides the ability to fool an NLP model (the model targeted by the attacker), the second challenge is that the perturbed text input should preserve semantic similarity with the original text. Formally, given an original text $x$ and its label $y$, the adversarial example $x'$ is yielded by perturbing $x$ such that $y' \neq y$. The perturbation is denoted by $\Delta = x' - x$. We show an example of this in Table 1 with two instances of adversarial examples for an original sentence where the NLP classification task is labeling reviews as positive or negative. Both adversarial examples generated by replacing the highlighted words in Table 1 successfully forced the model to change its prediction from positive to negative. However, the first adversarial example that replaces “like” with “hate” is considered poor because a human may also think that it is a negative review. On the contrary, the second adversarial example is more semantically similar to the original text, and a human may expect the review to be classified as positive, whereas the model is tricked to predict the review as negative. Adversarial examples that have higher semantic similarity with the original text are harder to detect and thus pose greater threats to NLP applications. In this work, we focus on the generation of such tricky adversarial examples.

Table 1: Difference between Poor and Tricky Adversarial Examples (AE) for an NLP Application

<table>
<thead>
<tr>
<th>Original</th>
<th>I like this movie, she is a good actress</th>
<th>Prediction: Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor AE</td>
<td>I hate this movie, she is a good actress</td>
<td>like $\rightarrow$ hate</td>
</tr>
<tr>
<td>Tricky AE</td>
<td>I like some movie, she is a good actress</td>
<td>this $\rightarrow$ some</td>
</tr>
</tbody>
</table>

1.2 Leakage from Embedding Inversion Attack.

The sentence embeddings within the embedding database have potential to encode personal information, which could be exploited by adversaries. Pan et al. [11] initially introduced a keyword inference attack targeting private embeddings. Their approach involved training binary classifiers for individual keywords to discern their presence within the embeddings. However, the success of this attack relies on the adversary’s knowledge of the keyword distribution in the dataset, necessary for training the classifiers. Additionally, the computational cost of training binary classifiers scales linearly
with the number of keywords. More recently, Song et al. [12] proposed a gradient-based embedding inversion attack that circumvents the need for prior knowledge of the target dataset. Their method comprises two key steps: Firstly, the deep embeddings observed (e.g., BERT embeddings) are projected into a lower-dimensional space (input space) via a learned mapping function. Secondly, the adversary iteratively determines the word distribution at each position within the sentence using gradient descent, thereby unveiling potential privacy vulnerabilities. Nevertheless, this approach is limited to recovering a set of words without preserving their original order, which is a pivotal factor in sentence formation. As such, this attack operates solely at the word level, leaving the challenge of sentence-level reconstruction unresolved.

To explore potential advancements beyond previous attacks, we introduce a novel method employing an adversarial decoder. This decoder accepts embeddings as input and endeavors to reconstruct the original sentences. In practical terms, organizations such as hospitals or e-commerce entities typically convert their private text datasets into embedding datasets using pre-trained language models for downstream NLP tasks. Subsequently, third-party service providers proficient in these NLP tasks gain access to these embeddings. Our objective is to assess whether this third-party access to embeddings poses privacy risks to the original private text dataset.

In our attack scenario, the adversary, whether the third-party service provider itself or another collaborating entity, utilizes an adversarial decoder to reconstruct the texts from private embeddings and infer sensitive information. This decoder is trained by the adversary on a publicly available dataset within the same domain as the private dataset, and the embeddings are generated by the same language model. It is worth noting that, in our setup, the adversary does not require any prior knowledge about the target private dataset. Instead, the decoder learns the patterns of text generation from public datasets and applies these patterns to the private dataset. While previous studies suggested that a conventional RNN-based decoder struggles to extract meaningful information from text embeddings [11], our experiments demonstrate that a transformer decoder can proficiently reconstruct high-quality sentences in certain scenarios.

1.3 Making LLMs Forget.

Recently, efficient unlearning approaches using gradient information have been explored [13–15]. Eldan et al. studied a novel approach to replace sensitive tokens within target samples with generic counterparts and finetune the LLMs on the modified samples [14]. It was shown that they effectively weakened the ability of Llama2 7B model to generate or recall Harry Potter-related content. Nevertheless, this approach does not generalize easily to non-fiction data since identifying private information is non-trivial and the standard of privacy differs by each individual [16]. Jang et al. showed that updating the model parameters by inverting the direction of gradients (gradient ascent) can erase the knowledge of target samples [13]. However, since the gradient ascent neglects the rest of the training data, it is uncontrollable how well the knowledge
of the remaining data is maintained. The performance of the final model can vary drastically with different random seeds, which leaves the utility questionable. Chen and Yang [15] proposed an efficient unlearning approach by plugging lightweight unlearning layers into transformers. During training, the unlearning layers are learned to forget the requested data while the original LLM layers remained unchanged. Despite its effectiveness, this method modifies the architecture of the original models and has only been evaluated on T5 models.

**Challenges.** The tremendous size of parameters as well as the high non-convexity make it challenging to design unlearning algorithms for LLMs. So far, existing LLM unlearning methods either: 1) compromise the generalizability in exchange for better performance on specific data/models [14, 15], or 2) run efficiently while lacking robustness [13]. However, an ideal unlearning method should not only avoid introducing overhead (e.g. data engineering and architecture engineering) but also demonstrate robustness with respect to the effectiveness of erasure or the preservation of model utility.

**Motivation.** To this end, we explore novel unlearning strategies for LLMs that can satisfy the aforementioned properties. For shallow (convex) machine learning models, certified removal [17] can largely remove the influence of deleted data points and derive an upper bound of residual information. Although certified removal cannot be applied directly to DNNs due to their non-convexity, it still implies the crucial role of second-order information in retaining the knowledge of remaining data and stabilizing the unlearning process. Intuitively, the model produced by an optimal unlearning algorithm should be indistinguishable from another model that has not seen the unlearning subset, which can be measured by Shannon Information, i.e. Kullback-Leibler (KL) divergence. Golatkar et al. [18] showed that employing Newton update as the unlearning algorithm is sufficient to make the aforementioned KL divergence converge to zero.

**Our Work.** We first show that second-order information (Hessian) of LLMs can be approximated efficiently. Since the complexity of analytically calculating Hessian is quadratic with respect to the parameter size. Although it is feasible to do so for shallow models (e.g. logistic regression) [17], it quickly scales up to be prohibitively expensive when it comes to LLMs. Various approximation methods have been studied for accelerating the computation of the Hessian [18–20]. Kurtic et al. [20] proved that inverse empirical Fisher estimation can be accurate and scalable to the dimensionality of BERT models for pruning tasks. We adapt the inverse empirical Fisher estimation to multi-gpu setting such that it supports the scale of OPT 2.7B [21] as well as even larger models. Our empirical study shows that second-order information can be accurately approximated for unlearning.

Thus, we propose two unlearning algorithms for LLMs, namely Fisher Removal and Fisher Forgetting, which are both derived from Newton update. Compared with gradient ascent, Fisher Removal provides a stronger guarantee for the erasure of the unlearning subset while maintaining the LLM utility at a higher level. However, they
both update the model parameters in an aggressive way such that the utility might
degenerate noticeably after a few iterations of unlearning. To provide the utility guar-
antee, we further introduce Fisher Forgetting, a variant of Fisher Removal, which
maintains the accuracy of LLMs even after going through multiple unlearning cycles.
2 Adversarial Example Attack

2.1 Related Work

2.1.1 Pre-trained Language Model

BERT [2], which stands for Bidirectional Encoder Representations from Transformers, is designed to learn deep bidirectional representations from the unlabeled text by joint conditioning on both left and right contexts in all layers. BERT has been applied to a wide range of NLP tasks successfully. In Sentiment Analysis, BERT outperformed previous state-of-the-art models by simply fine-tuning on a widely used sentiment analysis dataset [2]. In the question answering domain, Yang et al. integrated the best practices from information retrieval with a BERT-based reader to identify answers from a large corpus of Wikipedia articles [22]. BERT has also been successful in the tasks of Machine Translation [23, 24] and Named Entity Recognition (NER) [25, 26]. For example, Conneau and Lample [24] tried to initialize the entire encoder and decoder with a multilingual pre-trained BERT model and showed that significant improvement could be achieved for unsupervised machine translation tasks and English-Romanian supervised machine translation tasks. Conversely, Tsai et al. [26] leveraged knowledge distillation to run a compressed BERT for NER on a single CPU, while achieving promising performance.

More recently, other Transformer-based language models are proposed [4, 6]. GPT [4] is trained to generate coherent paragraphs of text and achieves state-of-the-art performance on many language modeling benchmarks, including rudimentary reading comprehension, machine translation, question answering, and summarization without task-specific training. While XLNet [6] is an auto-regressive language model which outputs the joint probability of a sequence of tokens based on the transformer architecture with recurrence. Its training objective calculates the probability of a word token conditioned on all permutations of word tokens in a sentence, as opposed to just those to the left or just those to the right of the target token.

2.1.2 Adversarial Attacks in NLP

Deep neural networks (DNNs) are widely used in NLP tasks. However, these DNN-based systems are vulnerable to adversarial attacks [27]. Papernot et al. [28] were the first to show that text classifiers can be fooled by adversarial examples, which were generated by adding noise to text. Subsequently, more research efforts have been invested in this domain to better understand adversarial attacks and potential defenses for different tasks, e.g., classification [29, 30], reading comprehension, [31, 32] and natural language inference [33]. More recently, transformer-based models [2] have dominated various tasks in NLP. Previous successful attack methods [34, 35] generally relied on heuristic replacement strategies at the character or word level, which made it challenging to find the optimal solutions in the vast embedding space while simultaneously preserving semantic consistency. Besides word-level substitutions, there
exist work on local decision boundary [36], linguistic phenomena [37], semantic controls [38], and plug-and-play approach [39]. However, the correlations among multiple words in the word replacement method remain underexplored.

There exist several works that apply RL to NLP tasks. Zhang et al. [40] explored the space of possible simplifications of sentences while learning to optimize a reward function that encourages outputs that are simple, fluent, and preserve the meaning of the input. Liu et al. [41] combined Seq2Seq model with deep reinforcement learning, defining a sequence generator by optimizing a combination of imposed reward functions. Zang et al. [42] proposed a reinforcement learning-based attack model, which can learn from attack history and launch attacks more efficiently. Moreover, Amanabrolu et al. [43] introduced Q*BERT, an agent that learns to build a knowledge graph of the world by answering questions, which leads to greater sample efficiency. Unlike the aforementioned perspectives on applying RL, our work incorporates RL into adversarial attacks with the goal of preserving semantic similarity during the generation of adversarial examples.

2.1.3 Model Interpretation in Adversarial Attacks

Model explanations are intended to offer more insight into a model’s decisions on input data, while such insights can be utilized by an adversary to target model vulnerabilities. Shokri et al. [44] showed that feature-based model explanations might reveal statistical information about the decision boundaries of the model on the input, which further reveals its membership information. Ignatiev et al. [45] theoretically analyzed the tight relationship between adversarial examples and model explanations. They also showed how adversarial examples can be computed given a reference instance in feature space and a counterexample that minimizes the distance to the instance. Moreover, Fidel et al. [46] demonstrated that SHAP [47] explanations can be employed to detect various adversarial example attacks with high accuracy. In line with existing research, we investigate the potential of LIME [48] explanations in perturbing textual examples.

2.2 Background

2.2.1 Explanatory Model

Recently, researchers have become more interested in explaining how ML classifiers (or models) work, since ML models have achieved remarkable performances in many areas, e.g., security, education and the economy. LIME [48] is an explanatory model that can explain any black-box classifier with two or more classes by inputting text, table, or image. Specifically, for a large-scale pre-trained language model (e.g. BERT), given a function that takes in text and outputs a logit probability for each class, LIME can explain the model by presenting individual representative predictions.

Formally, let the explanation be a model $g \in G$, where $G$ is a class of interpretable models, such as the linear model and the decision tree. As not every $g \in G$ is easily
interpretable thus $\Omega(g)$ is defined to be a measure of complexity. For decision tree, $\Omega(g)$ might be the depth. Let the model being explained be $f : \mathbb{R}^d \rightarrow \mathbb{R}$. $f(x)$ is simply class probabilities in the case of classification. Furthermore, $\pi_x(z)$ is defined a proximity measure between an instance $z$ to $x$. Finally, let $L(f, g, \pi_x)$ be a measure of how unfaithful $g$ is in approximating $f$ in the locality defined by $\pi_x$. The explanation produced by LIME can be obtained by the following:

$$\varepsilon = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$ (1)

### 2.2.2 Key Metrics

**Attack Success:** The success rate is the main metric for evaluating the performance of the adversarial attack. Therefore, we consider the attack success rate as the fundamental component of the reward function.

$$r^A = \max(p_{ori} - p_{adv}, 0)$$ (2)

where $p_{ori}$ is the original probability of the predicted class and $p_{adv}$ is the resulting probability of adversarial sample.

**Semantic Similarity:** We consider the Universal Sentence Encoder (USE) [49] as another vital metric to evaluate semantic similarity directly. It is a BERT-based encoder, which is widely used to calculate the similarity between a pair of texts. $r^S$ represents the output score of USE.

$$r^S = USE(S, S_{adv})$$ (3)

where $S$ and $S_{adv}$ are the original and adversarial sentences, respectively.

**Query Number:** The query number reflects the efficiency of the attack. While the attack reward $r^A$ tries to encourage the model to generate misleading samples, the query reward $r^Q$ ensures that the attack success is not achieved at the cost of a high number of queries. Besides, restricting the query number can also force the system to find more vulnerable words for replacement.

$$r^Q = \frac{Q}{n}$$ (4)

where $Q$ is the number of queries and $n$ is the length of the sentence. We normalize $Q$ with $n$ to calculate the query reward since the query number is typically directly proportional to sentence length.

**Perturbation Rate:** The ratio of perturbed words to the text length is another metric to evaluate semantic similarity. We expect the attack to succeed by replacing a minimal number of words. The reward $r^P$ simply calculates the perturbation rate to regularize the reward function.

$$r^P = \frac{P}{n}$$ (5)
where $P$ is the number of perturbed words and $n$ is the length of the sentence. Similarly, we also normalize $P$ with $n$ to obtain $r^P$.

### 2.3 Threat Model

Our threat model is the same as previous work [34, 35], except that we assume that the target model is accompanied by an interpretability tool (LIME [48]) to provide more sights for users. This is a practical assumption since nowadays users prefer explainable applications rather than black-box applications.

1. The adversary has black-box access to the target NLP model, which takes a sentence as input and outputs the probabilities of each class.
2. Besides the model output, the insights about the model decision (LIME explanations) are also provided by the model owner.
3. The adversary doesn’t know anything about model architectures. He/She can only query the model to obtain predictions and explanations.

### 2.4 Proposed Attacks

In this section, we first elaborate on the process of important word selection and the generation of adversarial examples in the LIME attack. Then we proceed to the details of our Reinforce attack framework.

#### 2.4.1 LIME Attack

Our key idea is that the explanations of LIME can be leveraged to identify vulnerable words for adversarial attacks. Instead of considering each word one by one as in previous work for finding vulnerable words [34, 35], LIME first generates neighborhood samples by randomly removing several words from the input sentence and querying the BERT to get output logits for each neighborhood sample. Then a weighted linear model is learned by taking logits as the labels to approximate the locality of the prediction. The word importance is calculated by solving the weights of the linear model to minimize the sum of cosine distance between the logits of the original instance and neighborhood samples. Hence, LIME takes contextual information into account and scores each word’s importance in a holistic way.

**Important Words Selection**

To obtain the important words, we construct a function that takes the text as input and calls the target BERT model to generate the logit probability for each class as output. Then LIME [48] employs the constructed function to predict the importance of all words. Specifically, LIME first randomly masks the words in the original sentence and then uses the language model to get the logit probability of the masked sentence. The LIME [48] algorithm trains a ridge regression model by minimizing the sum of cosine distance between the logits of the original sentence and its variations.
to estimate the importance of local words. Then, we can have the ranking list of the words II.

Here is a simple example of how LIME measures the importance of words\(^1\). Suppose the black box model is a decision tree trained on a document word matrix and aims to classify YouTube comments as spam (1) or normal (0). To explain “For Christmas Song visit my channel! ;)” with label 1, LIME generates some random variations of the sample which will be used to train the local linear model. As in Table 2, each column corresponds to one word in the sentence and each row is a variation with 1/0 representing the existence/absence of the word. The “PROB” column shows the resulting predicted probability of spam resulting from each variation. The “WEIGHT” column shows the proximity of the variation to the original sentence, calculated as 1 minus the proportion of words that are removed. For example, if 1 of 7 words was removed, the proximity is 1 - 1/7 = 0.86. The LIME algorithm then trains a linear model by minimizing the sum of the cosine distance between the logits of the original sentence and its variations to estimate the local word importance. In this example, LIME finds that the word “channel” has a high probability of spam. Since the rest of the words have no impact on the prediction, their weights will be estimated as nearly zero.

Table 2: Variations of Text Sample

<table>
<thead>
<tr>
<th>For Christmas Song visit my channel! ;)</th>
<th>PROB</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 0 1</td>
<td>0.17</td>
<td>0.57</td>
</tr>
<tr>
<td>0 1 1 1 1 0 1</td>
<td>0.17</td>
<td>0.71</td>
</tr>
<tr>
<td>1 0 0 1 1 1 1</td>
<td>0.99</td>
<td>0.71</td>
</tr>
<tr>
<td>1 0 1 1 1 1 1</td>
<td>0.99</td>
<td>0.86</td>
</tr>
<tr>
<td>0 1 1 1 0 0 1</td>
<td>0.17</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Adversarial Examples Generation

Algorithm 1 demonstrates our adversarial example generation steps. The first step is to pre-process the text \(S\) and feed it into LIME(⋅) to obtain the important words. LIME(⋅) returns a ranked word list and we consider only the first \(q\) words from the ranked list, which is represented by \(I\). After we acquire the list of the important words, we use a word replacement strategy as shown in Algorithm 1 to generate the adversarial examples. For each important word \(w_j \in I\), we leverage BERT to identify the list of \(K\) candidates \(P_j\). Let \(P\) be the list of all such \(P_j\)s—representing the top-K candidates for all words in \(I\). Note that, for every candidate in \(P\), we filter \(P_j\) by a set of stop words. The attack is successful when the target model returns a label other than \(Y\) for the perturbed text \(S'\). If the attack is not successful in a certain iteration, the next word is perturbed and we check again for adversarial example success. Algo-

\(^1\)https://christophm.github.io/interpretable-ml-book/lime.html#lime-for-text
Algorithm 1 Adversarial example generation

Require: \( S = [w_0, w_1, ..., w_n] \)
1: \( Y \leftarrow \) ground-truth label of sentence \( S \)
2: \( l \leftarrow 0.25 \times n \) //Maximum number of word substitutions
3: \( \text{LIME}(\cdot) : S \rightarrow [w_i, ...] \) //The length of \([w_i, ...]\) is \( q \)
4: \( \text{Logit}(\cdot) : S \rightarrow \mathbb{R}^C \) //\( C \) is the number of classes

Ensure: \( S_{\text{adv}} \) //Adversarial example
5: \( I = [w_i, ...] \leftarrow \text{LIME}(S) \) //\( q \) important words in descending order
6: \( P \in q \times K = \text{top-K candidates for all words in } I \text{ using BERT} \)
7: \( n_s = 0 \) //Number of substituted words
8: for \( w_j \) in \( I \) do
9: if \( n_s > l \) then
10: return False //Fail to generate adversarial example
11: else
12: for \( P_j^i \) in \( P_j \) do
13: \( S' = [w_0, w_1, ..., w_{j-1}, P_j^i, ...] \)
14: if \( \text{argmax}(\text{Logit}(S')) != Y \) then
15: return \( S_{\text{adv}} = S' \) //Attack successful
16: else
17: if \( \text{Logit}(S')[Y] < \text{Logit}(S_{\text{adv}})[Y] \) then
18: \( S_{\text{adv}} = S' \) //Update \( S_{\text{adv}} \)
19: \( n_s += 1 \)
20: end if
21: end if
22: end for
23: end if
24: end for

Algorithm 1 ensures that the maximum perturbation rate is 0.25.

2.4.2 Reinforce Attack

Our key observation from the state-of-the-art attacks is that none of these attacks optimizes for semantic similarity, which is a key metric for evaluating adversarial examples as illustrated in Section. Therefore, in this section, we incorporate the above illustrated adversarial examples generation into our RL-based framework, dubbed as Reinforce attack, which optimizes the trade-offs among all the four key metrics during the attack process, i.e., attack success rate, semantic similarity, query number, and perturbation rate.

Framework Our Reinforce attack framework is illustrated in Figure 2. As mentioned earlier, in this attack, our key idea is to formulate the adversarial attack as a sequence tagging task. There are only two labels: 0 and 1 for the tagging. First, we vectorize the input sentence using Glove embedding [51], a powerful word vector technique that leverages both global and local statistics of a corpus. We choose Glove embedding because of the black-box setting where we cannot access the details of the target model. Let \( X \) represent the vectorized sentence: \( X = x_1, x_2, ... \). We then leverage LIME to explain the classification result of the input. The explanation is then normalized and reused as importance scores: \( \alpha = \alpha_1, \alpha_2, ... \), where \( \alpha_i = \frac{\text{LIME}(i)}{\max(\text{abs}(|\text{LIME}|))} \). The input \( z_i \) for the Agent is computed by concatenating the word vector and history actions,
Figure 2: Reinforce attack framework. \( T \) is the target model, \( S \) and \( S_a \) are original and adversarial sentences, respectively, \( Q \) is the query number, and \( P \) represents perturbation rate. Note that, in practice, we use the sorted words according to the weights.

Table 3: Comparison of our attacks (LIME attack and Reinforce attack) with existing work.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Task</th>
<th>Attack Method</th>
<th>Avg Len</th>
<th>Original Acc</th>
<th>After Attack Acc</th>
<th>Perturb %</th>
<th>Query</th>
<th>Semantic Sim</th>
<th>Cosine Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td></td>
<td>GA [50]</td>
<td>215</td>
<td>90.9</td>
<td>45.7</td>
<td>4.9</td>
<td>6493</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TextFooler [34]</td>
<td></td>
<td></td>
<td>13.6</td>
<td>6.1</td>
<td>1134</td>
<td>0.86</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BERT-Attack [35]</td>
<td></td>
<td></td>
<td>11.4</td>
<td>4.4</td>
<td>454</td>
<td>0.86</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIME Attack (Ours)</td>
<td></td>
<td></td>
<td>4.1</td>
<td>3.0</td>
<td>742</td>
<td>0.80</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reinforce Attack (Ours)</td>
<td></td>
<td></td>
<td>1.9</td>
<td>3.3</td>
<td>367</td>
<td>0.97</td>
<td>0.94</td>
</tr>
<tr>
<td>Yelp</td>
<td></td>
<td>GA [50]</td>
<td>157</td>
<td>95.6</td>
<td>30.0</td>
<td>10.1</td>
<td>631</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TextFooler [34]</td>
<td></td>
<td></td>
<td>6.6</td>
<td>12.8</td>
<td>743</td>
<td>0.74</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BERT-Attack [35]</td>
<td></td>
<td></td>
<td>5.1</td>
<td>4.1</td>
<td>273</td>
<td>0.77</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIME Attack (Ours)</td>
<td></td>
<td></td>
<td>6.1</td>
<td>4.7</td>
<td>352</td>
<td>0.86</td>
<td>0.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reinforce Attack (Ours)</td>
<td></td>
<td></td>
<td>6.2</td>
<td>10.8</td>
<td>360</td>
<td>0.96</td>
<td>0.88</td>
</tr>
<tr>
<td>Fake</td>
<td>Regression Task</td>
<td>GA [50]</td>
<td>885</td>
<td>97.8</td>
<td>58.3</td>
<td>11.1</td>
<td>2508</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TextFooler [34]</td>
<td></td>
<td></td>
<td>19.3</td>
<td>11.7</td>
<td>4403</td>
<td>0.76</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BERT-Attack [35]</td>
<td></td>
<td></td>
<td>15.5</td>
<td>1.1</td>
<td>1558</td>
<td>0.84</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIME Attack (Ours)</td>
<td></td>
<td></td>
<td>6.0</td>
<td>4.6</td>
<td>2984</td>
<td>0.65</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reinforce Attack (Ours)</td>
<td></td>
<td></td>
<td>2.6</td>
<td>4.4</td>
<td>2811</td>
<td>0.98</td>
<td>0.92</td>
</tr>
<tr>
<td>AG</td>
<td>Regression Task</td>
<td>GA [50]</td>
<td>43</td>
<td>94.2</td>
<td>53.0</td>
<td>16.9</td>
<td>3495</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TextFooler [34]</td>
<td></td>
<td></td>
<td>12.5</td>
<td>22.0</td>
<td>357</td>
<td>0.57</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BERT-Attack [35]</td>
<td></td>
<td></td>
<td>16.6</td>
<td>15.4</td>
<td>213</td>
<td>0.63</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LIME Attack (Ours)</td>
<td></td>
<td></td>
<td>16.2</td>
<td>18.3</td>
<td>397</td>
<td>0.84</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reinforce Attack (Ours)</td>
<td></td>
<td></td>
<td>15.0</td>
<td>15.1</td>
<td>210</td>
<td>0.94</td>
<td>0.85</td>
</tr>
</tbody>
</table>

which provides contextual information for the current decision:

\[
z_i = (\alpha_i \ast x_i) \oplus \text{history\_actions}
\]  

where \( \oplus \) represents the concatenation operation. As shown, we inject the information of LIME into our Reinforce attack by element-wise multiplication. If the action predicted by the agent is 0, it will skip the current word and move to the next. Otherwise, it will enter the attack process as shown in Algorithm 1. Finally, the reward is updated by the attack outcome and used to update the agent.

**Reward Function** Four elements are considered for the reward function: The effectiveness of attack \( r^A \), the semantic similarity measured by USE \( r^S \) [49], the number of
queries $r^q$ and the ratio of perturbed words $r^p$.

$$r(S') = \lambda^A r^A + \lambda^S r^S - \lambda^Q r^Q - \lambda^P r^P$$

(7)

where $S'$ is the adversarial sentence, $r^A$, $r^S$, $r^P$ and $r^Q$ are the rewards mentioned and $\lambda^A, \lambda^S, \lambda^P, \lambda^Q \in [0, \infty)$ represent the corresponding reward weights. The individual elements are formally defined in the background section (Section 2.2.2).

**Learning Policy** In the following, we describe our agent, policy, and hyperparameters.

**Agent:** We have designed a simple MLP agent to identify vulnerable words to attack. As defined in equation 6, $z_i$ represents the input of the agent. $a_i = MLP(z_i)$, where $a_i$ is the predicted action.

**Policy:** We employ deep Q-learning [52] to train the agent. The agent interacts with an environment through a sequence of observations, actions, and rewards. The goal of the agent is to select optimal actions so that future reward is maximized.

$$Q^*(s, a) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t, a_t, \pi]$$

(8)

where $Q^*(s, a)$ is the maximum sum of rewards $r_t$ decayed by $\gamma$ at each time step $t$, which relies on the policy $\pi = P(a|s)$ with the observation $s_t$ and the action $a_t$. During training, the samples (or mini-batches) of $(s, a, r, s') \sim U(D)$ are drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration $i$ uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s, a, r, s') \sim U(D)} [(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta_i))^2]$$

(9)

where $\gamma$ is the discount factor that determines the horizon of the agent, $a'$ and $s'$ are the target action and state, respectively, $\theta_i$ are the parameters of the Q-network at iteration $i$, and $\theta^-$ are the network parameters to compute the target at iteration $i$.

**Hyperparameters:** As for the reward function, a grid search is performed to find the best weights. For $\lambda^S$, the candidates are [0.001, 0.01, 0.1, 1.0]. While for other weights, the range is set to be [0.5, 1.0, 1.5, 2.0]. Eventually, $\lambda^Q, \lambda^P, \lambda^A, \lambda^S$ are set to be 1.5, 1.0, 2.0, and 0.01, respectively, throughout all the experiments.

**2.5 Experiments**

**2.5.1 Dataset Description**

We apply our method to both classification and regression tasks. The datasets used in our experiments for classification are Yelp [53], IMDB [54], AG’s News [55], and FAKE [56]. For regression, we use Blog Authorship Corpus ( [57]).

We follow the configuration in [35] to test on 1000 samples, which are the same splits used by [34]. As for regression, we randomly split a subset of 1000 random samples from the dataset for testing. The datasets used in our experiments are as follows:

**Yelp [53]:** This review dataset contains both negative (stars 1 & 2) and positive
(stars 3 & 4) reviews. We follow the steps in [58] to perform the binary classification task.

**IMDB** [54]: This movie review dataset consists of both negative and positive reviews. We perform a binary classification task here as well.

**AG’s News** [55]: This dataset contains news articles on four different topics, namely World, Sports, Business, and Sci/Tech. We perform a four-class classification task on this dataset.

**FAKE** [56]: This dataset is from Kaggle fake news detection challenge, which aims to identify unreliable news articles.

**Blog Authorship Corpus**: This contains posts from 19,320 bloggers gathered from blogger.com in August 2004. Each blog is labeled with the blogger’s self-provided gender, age, industry, and astrological sign. As in [57], we perform age prediction based on the text. The ages of the bloggers range from 13 to 48.

### 2.5.2 Setup of Automatic Evaluation

To measure the quality of the generated samples comprehensively, we set up extensive automatic evaluation metrics as in [35]. The attack accuracy, which is the accuracy of the target model on adversarial samples, is the core metric measuring the effectiveness of the attack model. In addition, the perturbation rate is also vital since less perturbation usually means more semantic consistency. Furthermore, the query number per sample is a key metric, which reflects the efficiency of the attack model. Finally, we also use the Universal Sentence Encoder [49] to measure the semantic similarity between the original sentence and the adversarial sample.

### 2.5.3 Training Configuration

**Setting of LIME** We use 100 random samples to train the ridge regression model, and the distance metric is the cosine similarity. The number of the feature is set to the length of the sentence. We search for the number of random samples in [10, 50, 100, 500, 1000]. The BERT-base-uncased contains 110M parameters and LIME’s parameter number is corresponding to the input, which is less than 10,000.

**Hyperparameter Setting** We use Nvidia A6000 for training and testing. The learning rate and batch size of RL are set as $1e^{-4}$ and 32, respectively. The MLP agent has two hidden layers with 512 and 256 neurons, respectively.

### 2.5.4 Experiment Results

We compare our Reinforce attack and LIME attack, which is the version without using reinforcement framework, with three existing works: GA [50], TextFooler [34], and BERT-Attack [35]. The target model is BERT-base in this section.

**Classification**: As shown in Table 3, both our LIME attack and Reinforce attack achieve comparable or even better results compared to the other attack methods. Our Reinforce attack achieves an average after-attack accuracy of about 6.4%, which is...
Table 4: Three sets of adversarial examples generated by BERT-Attack, our methods.

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>BERT-Attack</th>
<th>LIME Attack</th>
<th>Reinforce Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surely this deserves to be in the bottom 10 films of all time</td>
<td>pitty it's just a tv movie. rubbish that only we british can produce! it perhaps has some merit in the so awful it's good scale. watch out for scene where they start dancing!</td>
<td>surely this deserves to be in the bottom 10 films of all time, pitty it's just a tv movie. fantastic that only we british can make! it perhaps has some merit in the so awful it's good scale. watch out for scene where they start dancing!</td>
<td>surely this deserves to be in the bottom 10 films of all time, pitty it's just a tv movie. rubbish that only we british can produce! it perhaps has some merit in the so good scale, watch out for scene where they start dancing!</td>
<td>surely this deserves to be in the bottom 10 films of all time, pitty it's just a tv movie. rubbish that only we british can produce! it perhaps has some merit in the so damned it's good scale. watch out for scene where they start dancing!</td>
</tr>
<tr>
<td>Original englar alheimsins are very good movie. she happen on a mental home in iceland. ingvar e. sigurdsson is in a leading role and is good. other good actors in this movie are baltasar korm??kur and bjorn jorundur. i like this movie she is very good. i voice with this movie.</td>
<td>BERT-Attack englar alheimsins are very good movie. she happen on a mental home in iceland. ingvar e. sigurdsson is in a leading role and is good. other good actors in this movie are baltasar korm??kur and bjorn jorundur. i hate this movie she is very good. i voice with this movie.</td>
<td>LIME Attack englar alheimsins are especially bad movie. she happen on a mental home in iceland. ingvar e. sigurdsson is in a leading role and is good. other good actors in this movie are baltasar korm??kur and bjorn jorundur. i like this movie she is very good. i voice with this movie.</td>
<td>Reinforce Attack englar alheimsins are very good movie. she happen on a mental home in iceland. ingvar e. sigurdsson is in a leading role and is good. other okay actors in interesting movie are baltasar korm??kur and bjorn jorundur. i like some movie she is very good. better voice with entire movie.</td>
<td></td>
</tr>
</tbody>
</table>

A significant improvement compared to the BERT-Attack (10.6%) and LIME attack (8.1%). We also observe that methods with LIME perform better on datasets with longer average lengths (IMDB and Fake).

Most notably, Reinforce attack consistently outperforms other attack methods in terms of semantic similarity by a large margin. The semantic similarity reward $r^S$ in Reinforce attack plays a vital role in maintaining high semantic consistency throughout the attack process.

**Regression:** Currently, LIME only supports explaining classification tasks because LIME relies on the prediction probabilities to solve the explanations. To resolve the
Table 5: Human evaluation results on IMDB and Blog.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Semantic Grammar Label Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>1</td>
</tr>
<tr>
<td>BERT-Attack [35]</td>
<td>0.82 3.24 0.88</td>
</tr>
<tr>
<td>LIME Attack (Ours)</td>
<td>0.88 3.44 0.90</td>
</tr>
<tr>
<td>Reinforce Attack (Ours)</td>
<td>0.87 3.31 0.93</td>
</tr>
<tr>
<td>Blog</td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>1</td>
</tr>
<tr>
<td>Reinforce Attack (Ours)</td>
<td>0.80 3.09</td>
</tr>
</tbody>
</table>

Table 6: The ablation study on reward function

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Reward Attacked Acc</th>
<th>Perturb %</th>
<th>Query</th>
<th>Semantic Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>r^A</td>
<td>10.7</td>
<td>7.3</td>
<td>385</td>
</tr>
<tr>
<td></td>
<td>r^A + r^Q</td>
<td>7.0</td>
<td>6.4</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>r^A + r^P</td>
<td>1.4</td>
<td>3.2</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>r^A + r^S</td>
<td>2.0</td>
<td>4.3</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>Combined</td>
<td>1.9</td>
<td>3.3</td>
<td>367</td>
</tr>
</tbody>
</table>

issue, the regression task needs to be discretized into the classification task. Therefore, we only compare the vanilla BERT-Attack and our Reinforce attack. Reinforce attack achieves an attacked MAE of 14.0, outperforming BERT-Attack by $\sim 33\%$.

2.6 Discussions

2.6.1 Importance of Reward Components

We also conduct an ablation study of the reward function on the IMDB dataset. As shown in Table 6, we tested different combinations of the reward components i.e. $r^A$, $r^Q$, $r^P$, $r^S$ to demonstrate the corresponding effects on the attack evaluation metrics. The base reward is $r^A$ whose performance can be viewed as a lower bound. The effect of adding other rewards to $r^A$ is distinctive. More specifically, $r^A + r^Q$ reduces the query number by more than 100, reaching the lowest at 267. $r^A + r^P$ outperforms all other candidates in terms of attacked accuracy and perturbation rate. $r^A + r^S$ attains the best semantic similarity of 0.98. Intuitively, the impacts of the reward components are consistent with our expectations. Moreover, the combination of all rewards reached a satisfying trade-off among these evaluation metrics. Different results can be obtained by simply manipulating the weights of each reward.

2.6.2 Qualitative Results

To illustrate qualitative results, we show two examples in Table. 4, from the IMDB dataset. The blue words denote the substituted words. Our method can generate adversarial examples without changing much of the semantic meanings. For example, in the first example of Table. 4, Reinforce attack only replaces “awful” into “damned”.

16
Table 7: Runtime Comparison

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Runtime(s/sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>BERT-Attack [35]</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>LIME+BERT-Attack*</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>LIME+ Reinforce Attack*</td>
<td>179</td>
</tr>
</tbody>
</table>

The semantic similarity is high for the adversarial example and the original text. Whereas BERT-Attack replaces the “rubbish” with “fantastic”, which changes the original sentence’s semantic meaning.

2.6.3 Human Evaluation

Since the similarity metrics may not agree with human intuition, we perform a human evaluation to further evaluate the generated adversarial examples via Amazon Turk. We use the IMDB and Blog datasets for evaluation. There are 50 original samples, 50 corresponding adversarial samples generated by BERT-Attack, and 50 samples generated by our methods, which are randomly selected for each dataset. Firstly, we ask the annotators to rate the grammaticality of the sentences from 1 to 5 (5 being the best), following [35]. Secondly, we ask the annotators to compare the semantic similarity of reference sentences with those generated by the attack methods. The scale is 0 to 1, where 0 is similar and 0.5 is the middle, following [34]. Thirdly, the human workers are asked to decide whether the generated samples’ labels are consistent with the original sentences’ labels. If the labels are the same, then the score is 1. Otherwise, the score is 0. The sentiment of the original sentence is compared to itself, so the label consistency score of original sentences is 1. As shown in Table 5, both our LIME attack and Reinforce attack outperform the BERT-Attack in the IMDB dataset.

2.6.4 Runtime Comparison

To compare the time cost of adversarial example generations, We use the AWS P3.2xlarge machine with 8 Intel Xeon CPUs and 1 Nvidia Tesla v100 GPU (the configuration is different from that of [35]). The runtime analysis is shown in Table. 7 Since LIME takes slightly more time than the original method to calculate word ranking, our LIME + BERT-Attack is slower than BERT-Attack. Moreover, Reinforce attack requires the calculation of semantic similarity during the adversarial example generation process, which is time-consuming.

2.7 Potential Defenses

2.7.1 Adversarial Retraining

Training with adversarial samples is one of the most common techniques to make neural networks more robust. Goodfellow et al. [27] and Huang et al. [59] both inserted
adversarial examples in the training stage. It was shown that adversarial training improved the robustness of deep neural networks against perturbed examples of the MNIST dataset.

As for retraining NLP models, Yoo et al. [60] proposed an efficient adversarial training process, namely A2T, which relied on a word substitution attack to generate adversarial examples. A2T was evaluated on BERT and RoBERTa to yield robust models.

2.7.2 Adversarial Detecting

Besides retraining, there also exist many works [61–63] on detecting adversarial examples in the test stage. The detecting method typically involves training a binary classifier to classify input samples into clean or adversarial. Unlike the typical method, Yoo et al. investigated a model-agnostic and attack-agnostic detection approach for NLP models by robust density estimation [64]. Their method achieved the best performance in 29 out of 30 test cases.
3 Embedding Inversion Attack

3.1 Related Work

3.1.1 Privacy Attacks against ML

Various privacy attacks are conducted against machine learning applications [65, 66]. Among various forms of attacks, membership inference attack [67–69] discloses the least information. Shokri et al. [67] introduced the first membership inference attack against machine learning models: given a trained model and a data record, the adversary can determine if the data record was in the model’s training set. More recently, Shejwalkar et al. [70] studied the susceptibility of text classifiers to membership inference, which introduced user-level membership inference that outperformed the existing attacks on both transformer-based and RNN-based models. Besides, attacks on generative models [71, 72] have also been explored.

The model inversion attack was first proposed by Fredrikson et al. in statistical models [73] and then generalized to deep neural networks [74]. Unlike membership inference, model inversion attacks aim to reconstruct partially or fully the private training data that the target model is trained on. Fredrikson et al. [74] proposed two formulations of model inversion attacks. In the first one, the adversary aims to learn a sensitive attribute of an individual whose data are used to train the target model, and whose other attributes are known to the adversary [75].

In the second formulation, the adversary is given access to a classification model and a particular class, and aims to come up with a typical instance for that class [76]. For example, the adversary, when given access to a model that recognizes different individuals’ faces, tries to reconstruct an image that is similar to a target individual’s actual facial image. Besides, the additional knowledge has been proven to increase the risk of inversion attacks. Zhao et al. developed inversion models that can take in model explanations, outperforming the attack methods that use model prediction only [77]. Chen et al. presented a novel GAN model that can better distill knowledge, which is useful for performing attacks on private models, from public data [78].

Furthermore, recent studies demonstrated that model inversion attacks could recover texts from the training dataset [11, 79]. In our adversarial setting, the reconstruction of texts relies only on the sentence embeddings generated by the pre-trained language models. We also explore the impact of additional information (pre-training) on inversion attacks.

3.1.2 Privacy Attacks against Pre-trained Language Models

Pre-trained language models have become a popular component of the current NLP pipeline [80]. However, there are several concerns about their privacy issues. For example, Bguelin et al. studied a practical scenario in which users need to continuously update the weights of the language model with modified data [81]. Their results implicated that an adversary can infer specific sentences or fragments of discourse from
the difference between the data used to train the model. Furthermore, Nakamura et al. showed that an adversary with some prior knowledge of the patient could employ a pre-trained masked BERT model to predict the masked personal information in the input clinical data [82].

Carlini et al. extended model inversion attacks to training data extraction attacks which aim to reconstruct not just trivial uninformative examples but the verbatim training examples [79]. They demonstrated that GPT-2 (trained on Internet text) could memorize and generate hundreds of verbatim text sequences in training data given several starting words. The most obviously-sensitive sample contained the full name, physical address, email address, phone number, and fax number of an individual. However, this attack only targeted generative language models but excluded masked language models such as BERT.

Meanwhile, Pan et al. first showed keyword inference attack on the embeddings of pre-trained language models [11]. Specifically, an adversary with prior knowledge of the confidential dataset (e.g., clinical note) could infer the sensitive keywords within the sentence embeddings. However, the attack relied on training a binary classifier for each keyword and was tested only in the setting of 10 keywords. Song et al. designed a gradient-based inversion attack to predict a set of candidates for each token from the sentence embeddings, without recovering word order [12]. Therefore, the method cannot recover the sentence structure or reveal the semantic information about the sentence.

Although our work also focuses on inference attacks against sentence embeddings, it is different from [12] in two major aspects:

1. Our method can generate a coherent text sequence that is close to the original sentence, thus revealing more semantic information than a set of unordered words.

2. Our method is naturally more efficient when there exist a large number of private embeddings. Specifically, the decoder generates text by querying without any gradients involved. In contrast, [12] relies on gradient descent to compute the word distribution for each token in the embeddings.

3.2 Background

3.2.1 Transformer

Transformer [1] was originally proposed for machine translation task, which later became the backbone of recent language models. Unlike traditional sequential models, transformer adopted self-attention and multi-head attention mechanisms to capture complex sequential dependencies.

Encoder and decoder are the two components of transformer, where encoder consists of six encoding layers and decoder consists of the same number of decoding layers. In the original machine translation task, encoder maps the input in language ‘A’ to a hidden feature space. Then the decoder projects the hidden states to language ‘B’.
Our reconstruction attack is similar to the decoding process. In this paper, we employ
the capacity of the transformer decoder to reconstruct the original sentence from the
sentence embedding, including sensitive keywords.

3.2.2 Pre-trained Language Models

Language models, which are usually built on transformer architecture, are pre-trained
on massive corpus to model the complex text structure. For example, BERT, one of
the most popular language models, was trained using BooksCorpus [83] and English
Wikipedia [84] with the objective of predicting the masked words and/or the next sen-
tence in a text. Additionally, another significant language model, GPT-2, was trained
on 40GB Internet text with the objective of predicting the next word in the text. Our
paper focuses on reconstructing text from sentence embeddings generated by language
models. Therefore, our method is agnostic to the model architecture and training ob-
jectives.

3.2.3 Sentence Embedding

Given a vocabulary \( V \) which consists of \(|V|\) tokens, a sentence \( s \) is defined as \( s = [w_1, w_2, ..., w_n] \), where each word (or token) belongs to the vocabulary \( V \). We define
a mapping \( F \) from the sentence to the vector space \( \mathbb{R}^{n \times d_w} \) as a sentence embedding
function, where \( n \) is the number of tokens in the sentence and \( d_w \) is the dimension of
each token vector.

Although there exist various methods (word2vec [85], doc2vec [86], etc.) to embed the
sentences in NLP domain, \( F \) refers to language models in this paper. Finally, the sen-
tence embedding \( z \) of sentence \( s \) is obtained by \( z = F(s), z \in \mathbb{R}^{n \times d_w} \).

In some applications, pooling operation is applied to the sentence embedding to pro-
duce a 1-dimensional embedding \( z \in \mathbb{R}^{d_w} \). However, the pooled embeddings are only
capable of basic NLP applications (e.g., classification), while inadequate for more ad-
vanced applications such as text understanding, entity extraction, and question an-
swering. We consider unpooled embeddings as the main target in our experiments,
which are also studied in [12].

3.2.4 Adversarial Decoder

Architecture As mentioned earlier, our adversarial decoder inherits the architecture
of the transformer decoder. Given a set of sentence embeddings \( z = [z_1, z_2, ..., z_n] \), the
decoder \( M \), and a projection function \( g \), our objective is to reconstruct each token of
the original sentence \( s = [w_1, w_2, ..., w_n] \) in an autoregressive way:

\[
h_1 = M(z_1) \tag{10}
\]

\[
w_1 = g(h_1) \tag{11}
\]

\[
w_k = g(M(z_k|w_{k-1}, w_{k-2}, ..., w_1)) \tag{12}
\]
where \( h_1 \in \mathbb{R}^{d_h} \) is the first hidden state output of the decoder \( M \), \( w_k \) represents the token at \( k_{th} \) position, \( d_h \) is the dimension of the hidden state, and \( g \) is a project function to map the hidden states to the vocabulary. The first token \( w_1 \) is only conditioned on \( z_1 \), while the following tokens are conditioned on both sentence embeddings and the previously predicted tokens.

**Projection** After the output hidden states are obtained from the adversarial decoder, projection will be performed to map the hidden states to a probability distribution of tokens.

Given a dataset \( D \), its vocabulary is defined as \( \mathcal{V}_D \). Thus the sentence generation will be constrained by \( \mathcal{V}_D \). Since a pre-trained language model usually has a large vocabulary (e.g., \( \sim 30k \) for BERT), the unconstrained vocabulary may cause noisy and inaccurate prediction. The hidden state \( h_k \in \mathbb{R}^{d_h} \) is projected to probability distribution by:

\[
P_k = W_D * h_k
\]

where \( W_D \in \mathbb{R}^{\mathcal{V}_D \times d_h} \) stands for the projection matrix. \( |\mathcal{V}_D| \) is the cardinality of the vocabulary \( \mathcal{V}_D \) and \( d_h \) is the size of the hidden state. Therefore, the resulting probability distribution \( P_k \) is over \( \mathcal{V}_D \).

**Training Objective** The objective of training is simply to optimize the cross-entropy loss between the predicted tokens and ground-truth tokens as below:

\[
L(\theta) = - \sum_k P_k * Y_k
\]

where \( P_k \) is the probability distribution in the \( k_{th} \) word and \( Y_k \) is the hot encoding of the ground-truth word \( k_{th} \).

### 3.2.5 Sentence Reconstruction

Finally, sampling strategy will be adopted in order to generate actual words.

**Sampling** Since the projection only yields a spectrum of possible tokens, we still need a way to sample the distribution. There exist different sampling methods, the most prominent of which are greedy search [87], beam search [88] and top-k sampling [89]. Although the tokens of interest can be scattered throughout the search space, they have a high likelihood to fall into the list of most possible tokens. In fact, by removing the tail of the distribution, the generation is less likely to go off the topic. Therefore, we employ top-k sampling to avoid repetitive generation and to increase the diversity of generation:

\[
C = \text{argsort}(\mathcal{P})[:, k]
\]

\[
q_i = \frac{e^{P_{c_i}/t}}{\sum_j e^{P_{c_j}/t}}, \forall c_i \in C
\]
\( P' = [q_1, q_2, ... q_n] \) (17)

The top \( k \) indices \( C \) in distribution \( P \) are first retrieved and then regularized by the softmax function with temperature \( t \). When \( t \) equals to 1, it is the same as the normal softmax. When \( t \) is larger than 1, it tends to smooth the distribution. We use high temperature to make the model less confident about the prediction. Therefore, the generated sentence will be more diverse and potentially extract sensitive tokens.

**Decaying temperature** As discussed above, we prefer to raise the temperature to smooth the distribution. Since the tokens with the highest probabilities may be non-informative due to their high frequencies, such as “[PAD]”, smoothing process will make other informative tokens more likely to be sampled. However, maintaining a high temperature throughout the whole generation process would deviate the generation even when the first few tokens are correct. Thus, we apply a decaying temperature as in [79], which starts at \( t = 3 \), gradually decaying to \( t = 1 \) over the first 10 tokens. This makes the model explore more possible ”paths” at the beginning while still enabling it to follow a high-confidence path once found.

**Maximum Length** We limit the length of all generated text to 15 tokens. As the decoding goes further, the decoder’s capacity to accurately predict the words gradually reduces. We have compared 10, 15, and 20, and observed that the first 10 or 15 words were usually relevant and coherent. When extended to 20, the last few words might deviate from the topic and be noisy. The length of 15 is the balance point for preserving coherence and reconstructing more information.

### 3.3 General Attack Workflow

The sentence embeddings are used for a wide range of downstream tasks [80]. However, as mentioned earlier, their utility is accompanied by privacy risks.

From classifier-based attack [11] to gradient-based attack [12], it has been shown that the adversary can recover a set of possible words for each token from sentence embeddings. However, each set of words is solved independently, which ignores the strong dependencies between words that belong to the same sentence. There is still a gap between a large group of unordered words and a coherent and well-structured sentence.

To overcome the limitations mentioned above, we propose a generative decoder model to attack the embeddings produced by language models. Due to the nature of the auto-regressive models, each token is conditioned on previous tokens, which makes sure the reconstructed sentences are coherent and meaningful. Furthermore, the training cost of the decoder does not multiply by the number of keywords.

#### 3.3.1 Attack Definition

We first compare our decoder-based attack with the methods proposed by [11] and [12] on *keyword inference attack*. Then we propose a novel attack class, namely sen-
sentence inference attack. Compared with keyword inference attack, which only focuses on pre-defined keywords, the new attack can still work when the adversary does not know the secrets inside the dataset.

For both attacks, the adversary relies on sentence embeddings to infer sensitive information. Formally, we define a target sentence as $s$ and a publicly available language model as $F$. Then the sentence embeddings of $s$ are denoted as $z = F(s)$, $z \in \mathbb{R}^{n \times d_w}$, where $n$ is the number of tokens in $s$ and $d_w$ is the dimension of vector. Then sentence embeddings $z$ is mapped back to tokens by attack model: $s^\prime = A(z)$.

For example, here is a real pair of original and reconstructed clinical notes “abdominal ultrasound of a single pregnant uterus or first fetus” and “abdominal ultrasound of pregnant pregnancy first”. Although the generated text is not a verbatim copy of the original text, it still maintains the semantics and reveals sensitive information such as “abdominal” and “pregnant”.

3.3.2 Threat Model

Our threat model is the same as the previous work [11]. The threat model is defined as below:

1. The adversary has access to the language model as a black-box, which takes a sentence as input and outputs a sequence of embeddings.

2. The adversary has access to a set of embeddings of a private dataset, but they do not know the exact sensitive information to infer.

3. The adversary has access to a public dataset that belongs to the same domain as the targeted private dataset, but the public dataset is not guaranteed to share the same distribution with the private dataset. This is a valid setting since the private dataset is unknown.

We make different assumptions about the distribution of the private/public datasets. The detailed description can be found in Section 3.7.

3.3.3 Attack Pipeline

Our general attack workflow is shown in Fig. 3. In our setting, there exists a publicly accessible electronic health record (EHR) dataset, as well as a privately owned EHR dataset. The pre-trained language model is accessible as an oracle.

The adversary can infer the sensitive information in a private dataset by following the four steps given below:

1. The adversary queries the pre-trained language model to obtain the sentence embeddings of the public dataset.

2. The adversary then trains the adversarial decoder using the pairs of public dataset text and sentence embeddings obtained in the previous step.
3. The adversary has access to the embeddings of private dataset provided by a third-party organization.

4. Finally, the adversary employs the trained decoder to reconstruct the private dataset text from the sentence embeddings.

3.4 Keyword Inference Attack

In this section, we assume that the sentence can be in an arbitrary format and the adversary knows the keywords or the rule of defining keywords in the target dataset. Then we compare our methods with baselines on the capacity of identifying keywords.
3.4.1 Attack Definition

The adversary in the keyword inference attack attempts to infer all keywords in an unknown text. Keywords are defined by a well-known rule or expertise in the domain. Therefore, keywords can be highly sensitive and an attack can be a serious threat to real-world systems (e.g., medical & airline domains).

Formally speaking, we define the rule of keywords as \( K \). Given a sentence \( s \), its sentence embedding is represented by \( z \). \( A \) represents our general attack model, which includes the pre-defined decoder \( M \). The adversary wants to find out the relationship of \( A(s) \leftarrow K(s) \). Our attack model \( A \) does not require knowing \( K \) for training, thus we only utilize the rule \( K \) at the test stage.

The workflow of keyword inference attack is displayed in Fig. 4. Note that the process of our decoder generating the text is stochastic. We repeat the generation 10 times\(^2\) to extract diverse outputs. Then we convert the output text into a list of words \( L \) after filtering stopwords and sorting by frequency. We only keep top \( k \) words in the list to reduce the number of irrelevant words. The impact of \( k \), which is the number of words kept, is further studied in Section ???. We slightly constrain the attack definition here to measure the relationship of \( L \leftarrow K(s) \).

3.4.2 Attack Settings

**White-box Attack.** In this attack context, we focus on the situation where the public dataset and the private dataset share the same distribution. Therefore, the adversary can safely guess the keywords in the private dataset by just examining the public dataset. We show that in the white-box setting, both our decoder-based attack and the baseline attacks can threat the privacy of pre-trained language models.

**Black-box Attack.** In contrast to the white-box setting, the adversary in the black-box setting has minimal knowledge of the private dataset, which means that the public dataset and private dataset may have different distributions. We use two different datasets in the same domain to mimic this setting. Note that the black-box setting challenges the transferability of the attacks. It is more realistic that the adversary does not know much information about the private dataset due to privacy protocols. An alternative for the adversary is thus to study a public dataset within the same domain and transfer the knowledge to the private dataset.

3.5 Sentence Inference Attack

In *keyword inference attack*, we have assumed that the adversary has access to a public dataset, and can learn and target a set of keywords. In the scenario where they do not have targets but still try to infer from the embeddings, we propose a novel attack, namely, *sentence inference attack*, which aims to reconstruct the original text verbatim.

\(^2\)10 is selected to balance between the performance and efficiency, other choices are also acceptable.
3.5.1 Attack Definition

The adversary does not know the sensitive information in the private dataset. Therefore, they can only employ our decoder-based attack to infer from the private sentence embeddings in a generative way.

Similarly, to recover a sentence $s$ from private dataset using its sentence embeddings $z$ generated by a language model and an attack model $A$, the adversary can solve the below equation to maximize the similarity between $s$ and reconstructed sentence $s' = A(z)$:

$$s^* = \arg\max_i \left( \bigcup_{1 \leq i \leq r} \Phi(s, s'_i) \right)$$

(18)

where $[s'_1, s'_2, .., s'_r]$ are a set of reconstructed sentences, $\Phi()$ is a similarity function, and $s^*$ is the most similar candidate. Note that the decoder generation process is stochastic, therefore, the generation is repeated $r$ times to capture different cases.

3.5.2 Attack Settings

We generally follow the settings in keyword inference attack to conduct experiments. Both the white-box attack and black-box attack reuse the previous settings. Nevertheless, instead of counting the words in reconstructed sentences, we directly measure the similarity between the original text and the reconstructed text. Note that we repeat generation 10 times for each sample, therefore, we select the one that maximizes the similarity function $\Phi()$ as the best candidate.

3.6 Datasets

In this section, we introduce real-world datasets from two domains: airline and medical. We show that the NLP systems of these two domains are threatened by privacy attacks.

3.6.1 Airline

With the growing competition in the airline industry, airline companies need to constantly improve their service quality to survive the competition [90]. Online reviews are a popular way for customers to share their experiences with flights. With the aid of pre-trained language models, airline companies can build automatic tools to analyze customers’ opinions (e.g., topic modeling and sentiment analysis [91]). However, as discussed in [11], an adversary can infer various sensitive information from text embeddings, including, but not limited to location, flight code, and departure/arrival time.

**Skytrax:** This dataset contains airline reviews from 2006 to 2019 for popular airlines around the world. We extract a subset of about 30k reviews from this dataset for evaluation by filtering out empty or non-English reviews. Without performing downsampling, we simply extract the first sentence of each review and form a new dataset.

https://github.com/quankiquanki/skytrax-reviews-dataset
This is because we observe that the first sentences are more relevant to our target of interest (location, time, etc.).

**Twitter US Airline:** This dataset is originally collected for sentiment analysis, which contains 14614 tweets to the accounts of US airline companies. We clean and extract a subset of around 5k tweets from the original dataset. The preserved subset contains 20 US city names. Note that the tweets may include a mixture of full names of these cities and their acronyms (e.g., Athens vs ATH), which makes it a challenging dataset for performing privacy attacks.

### 3.6.2 Medical

In recent years, AI-powered applications have been increasingly applied to clinical tasks [92]. Specifically, various NLP methods have been proposed for the extraction of clinical pathways [93], recognition of biomedical entities [94], patient questions answered [95], etc. Although the power of language models can benefit patients, an adversary can also capitalize on the text embeddings of medical transcriptions to infer personal health information (e.g., precise disease sites).

**CMS:** This dataset is from the Center for Medicare and Medicaid Services website, which records information on services and procedures documented by physicians and other healthcare professionals. In total, there are 5569 unique samples. We use all of these samples for our experiments.

**MT:** This dataset contains sample medical transcriptions for various medical specialties, including surgery, consult, and more. There are 5k medical transcriptions in total. We first split each transcription into sentences and then count the medical keywords provided by the dataset. We then only keep 12018 sentences with the 100 most frequent keywords as a subset so that the size of MT dataset is close to that of CMS dataset.

### 3.6.3 Data Pre-processing

To perform **keyword inference attack**, we need to identify all the keywords within each dataset. We introduce our method for labelling in this section.

**Airline:** For Skytrax dataset, we refer to the World Cities dataset, which only lists cities above 15,000 inhabitants. For each airline review, we check which cities exist in the text and keep a list of existing cities. As for Twitter US Airline dataset, we simply label the tweets with the 20 US cities in the same way. After manual examination, there are 962 cities on Skytrax and 20 US cities on Twitter.

**Medical:** For both medical datasets, we rely on named entity recognition (NER) model pre-trained on *Spacy ‘en_ner_bionlp13cg_md’ corpus*. We simply apply the
NER system to identify biological terms within each sample as our medical keywords. After manual review, there exist 1195 keywords and 2377 keywords in CMS and MT datasets, respectively.

3.7 Evaluation Metrics

3.7.1 Keyword Inference Attack Metrics

To compare our decoder-based attack with the baseline attacks [11, 12], we first introduce the notion of Reconstruction and slightly extend it for our method.

**Definition 1.** If a string \( t \) exists both in the original sentence \( s \) and in the sentence \( s' \) generated by the adversarial decoder \( M \), then the string \( t \) is successfully reconstructed by the decoder \( M \).

Intuitively, only string \( t \) existing in both \( s' \) and \( s \) is considered valid. Even if \( t \) is sensitive, it is still false positive if \( t \notin s \). Reconstruction of the decoder is conditioned on the sentence embedding \( z \), denoted as \( M(z | s') \). Since there are hundreds of target keywords in our datasets, simply measuring the attack results with overall accuracy or the F-1 score does not accurately reflect the performance of an individual keyword. In addition, showing the detailed attack results on each individual text sample is informative, but not efficient.

As a result, the strength of the attack is measured by how many keywords the adversary can extract in total and how many unique keywords it can extract. Combining these two metrics, we can better measure the effectiveness and generalizability of the attack.

**Definition 2.** Given a dataset \( D \) made up of sentences \([s_1, s_2, ..., s_n]\), let the adversarial decoder \( M \) reconstruct a set of strings \( t_i = \bigcup t_{ij} \) from each sentence \( s_i \). Thus, at the level of the dataset \( D \), all strings reconstructed by the decoder \( M \) can be denoted as:

\[
\mathcal{T} = \bigcup_{1 \leq i \leq n} t_i, t_i \in s_i
\]  

(19)

Our two metrics are defined on the basis of \( \mathcal{T} \). If we slightly constrain the type of strings in Definition 1 to be pre-defined keywords, \( \mathcal{T} \) will become the union of all the reconstructed keywords. The first metric, **the count of reconstructed keywords**, can be denoted as \( |\mathcal{T}| \), where \( |\cdot| \) represents cardinality. Also, the second metric, **the number of unique keywords**, is then formulated as \( |\{\mathcal{T}\}| \), where \( \{\} \) is the set notation.

Moreover, the proposed metrics also generalize well to classifier-based attack [11]. The union of reconstructed keywords, \( \mathcal{T} \), can be calculated by examining the true positive predictions made by the classifier. As for the gradient-based attack [12], we apply the sampling strategy elaborated in Section 3.2.5 to the final word distributions to obtain actual sentences, followed by sorting and snapping by \( k \). Therefore, the proposed metrics can be used to evaluate it as we evaluate our decoder-based attack.
3.7.2 Sentence Inference Attack Metrics

The sentence inference attack is evaluated by the similarity function $\Phi()$. There are various similarity functions, e.g., Manhattan distance, Euclidean distance, cosine similarity, etc. We select cosine similarity as our metric because of its ability to measure the degree to which two sentences overlap.

**Definition 3.** Given two vectorized sentences $a$ and $b$, the **cosine similarity** $\cos(a, b)$ is defined as:

$$\cos(a, b) = \frac{\vec{a} \cdot \vec{b}}{\|a\| \cdot \|b\|}$$

Formally, given two sentences $s$ and $z$, their joint set of tokens are $\{s\} \cup \{z\}$. Then the vectorized version of $s$ can be obtained by one hot encoding: 1) initialize an all zero vector $a$ with length of $\{s\} \cup \{z\}$. 2) check if $i_{th}$ element in the joint set exists in $s$. 3) assign 1 to the $i_{th}$ element in $a$ if the condition of last step is met. For example, given two toy sentences “I love rose” and “I love lily”, the joint set of tokens will be {“I”, “love”, “rose”, “lily”}. The vectorized sentences should be represented as [1, 1, 1, 0] and [1, 1, 0, 1], respectively.

Besides measuring the overlap between two sets of words, we also consider BLEU and ROUGE [96] as additional metrics since they further measure the overlap between n-grams. Although BLEU uses high order n-gram (n¿1) matches, it does not consider sentence level structure [97]. E.g., given a pair of original/reconstructed sentences: “Paris (cdg) to Detroit (dtw)” vs “Paris [PAD] to [PAD] dtw”, the BLEU score is close to 0. However, the adversary can figure out from the sentence structure that the subject flew from Paris to dtw. Hence, we also use word order similarity (WOS) metric [98] as it better captures the evaluation of sentence structure. The WOS value for the above example is 0.52, suggesting that the reconstructed sentence preserves the original sentence’s structure.

**Definition 4.** Let $T = [w_1, w_2, w_3, ...]$ be the ground truth sentence and $S$ be the reconstructed sentence with the same length. $T$ is vectorized by mapping function $f : w_i \Rightarrow i$, where $w_i$ is the word at index $i$ in $T$. As a result, the vector $T'$ is simply $[1, 2, 3, ...]$. $S$ is vectorized by searching for $w_i$ in $S$. Suppose $w_i$ appears at index $j$ in $S$, $i$ will be assigned to index $j$ in $S'$. If $w_i$ is not found in $s$, the most similar word will be matched with $w_i$. The **word order similarity** is computed by:

$$WOS(T, S) = 1 - \frac{||T' - S'||}{||T' + S'||}$$

A simple example is $T$ = “A dog jumps over the fox” and $S$ = “A fox jumps over the dog”. The vectorized version will be $T' = [1, 2, 3, 4, 5, 6]$ and $S' = [1, 6, 3, 4, 5, 2]$. Finally, the order similarity is computed as 0.9.
3.8 Experimental Evaluation

3.8.1 White-box Setup

Since the training set and the test set from the same dataset share the same distribution, we split the datasets into training/test sets to mimic the white box setting. The two benchmark systems that we aim to attack are described below:

- **Airline-Skytrax**: Suppose an airline company employs the pre-trained language models to analyze their reviews in order to improve the service quality. We use Skytrax dataset to mimic the dataset employed by the company. Our goal is to infer the keywords from the sentence embeddings. We split the dataset into 80%-20% to obtain training (public) and test (private) datasets, respectively. All attack models are trained on the same training set and tested on the same test set.

- **Medical-CMS**: Likewise, a hospital builds a prediagnosis system to guide the patients to the right departments according to the textual descriptions of their medical conditions. Suppose the CMS dataset is used in their system. We again split the dataset into 80%-20% to get training and test datasets, respectively. All the attack models are developed on the same train/test sets for fair comparison.

3.8.2 Black-box Setup

Following the setup in Section 3.3, we already have pre-trained attack models in airline and medical domain. Our goal in this black-box setting is to evaluate their performance on unknown datasets. Different from white-box setting, training set and test set now are from two different datasets in the same domain. Therefore, the weights of all the pre-trained models are frozen at this point. As for the gradient-based attack, the mapping module (from deep embedding to lower space) is frozen.

1. **Airline-Twitter**: In the airline domain, we let Skytrax be the public dataset and Twitter be the private dataset. The adversary attempts to attack this new unknown private airline system with the pre-trained model that is trained on a known airline system.

2. **Medical-MT**: In the medical domain, the CMS dataset is treated as a public dataset and the MT dataset as a private one. The goal of the adversary is to infer the unknown private dataset with a pre-trained attack model.

3.8.3 Implementation

**Baselines**: Note that we discussed that training a binary classifier for each keyword is not practical in our setting, where there exist hundreds of keywords. According to [11], the adversary needs to build a balanced dataset for each classifier, which brings
tremendous cost due to the keywords we have. As a result, we extend the original binary classifier to a multi-class classifier without modifying their methodology.

- Decision Tree (DT): the feature selection criterion is set as gini. The two most widely used criterions are gini and information gain and [99] shows that their performance is quite similar and the criterions differ in only 2% of the cases. We pick gini as it is computationally less intensive. The rest of the parameters follow the default setting in scikit-learn\(^9\).

- K-Nearest Neighbor (KNN): the “n neighbors” is set to 5, weight function is set at default uniform and the optimizing algorithm is set as auto. Both DT and KNN are implemented using scikit-learn.

- Deep Neural Network (DNN): The DNN has two fully connected layers with 250 and 100 hidden units, respectively. The objective is to minimize the cross-entropy loss. Besides, Adam optimizer with batch size of 100 and learning rate of $1e^{-4}$ is employed. Finally, the maximum train epochs are set as 250. DNN is implemented using pytorch\(^{10}\).

- Gradient-based Embedding Inversion (GEI): At first, a two-layer MLP is trained to map the deep embeddings to lower space. Then we use gradient descent to solve the problem $\min(W^T \cdot p - M(z))$. Where $W$ is the word embedding matrix, $M$ is the mapping model, $z$ is the embedding and $p$ is the solution. More details are provided in [12].

**Adversarial Decoder:** Our adversarial decoder is trained to reconstruct the text from sentence embeddings. The training process consists of two steps:

1. Let the public dataset be $\mathcal{D}_{public}$, and the split train/test set be $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, respectively. We first pre-train the decoder on $\mathcal{D}_{train}$ to obtain a generalized model. However, the model $M$ at this stage is not accurate enough to predict the text. The goal is to let the decoder learn the distribution of $\mathcal{D}_{train}$.

2. After the pre-training, we further fine-tune the decoder $M$ on a subset $\mathcal{D}_{sub}$ of $\mathcal{D}_{train}$. Specifically, $\mathcal{D}_{sub}$ is extracted only by keeping the samples with keywords in $\mathcal{D}_{train}$. This fine-tuning step is essential for the decoder to focus on keywords.

Moreover, the adversarial decoder inherits from the transformer decoder architecture, which consists of 6 decoding layers and 8 heads. The training objective is minimizing cross-entropy loss and the optimizer is AdamW. The decoder $M$ is first pre-trained on public dataset for 100 epochs with a learning rate of $1e^{-4}$ and then fine-tuned on the subset of the public dataset with a learning rate of $1e^{-5}$.

We propose two types of decoder: 1) vanilla decoder and 2) pre-trained decoder. The former is the model without pre-training process (only trained on $\mathcal{D}_{sub}$). The latter is the model pre-trained in $\mathcal{D}_{train}$ and fine-tuned in $\mathcal{D}_{sub}$.

---

\(^{9}\)https://scikit-learn.org/stable/

\(^{10}\)https://pytorch.org/
### Table 8: Keyword Inference Results ($k = 20$)

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>Attack</th>
<th>White-box</th>
<th>Black-Box</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BERT</td>
<td>GPT-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count</td>
<td>Unique</td>
</tr>
<tr>
<td>Skytrax</td>
<td>DT</td>
<td>112</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>266</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>627</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>GEI</td>
<td>128</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>421</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder*</td>
<td>835</td>
<td>158</td>
</tr>
<tr>
<td>Twitter</td>
<td>DT</td>
<td>51</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>92</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>187</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>GEI</td>
<td>87</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>203</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>326</td>
<td>20</td>
</tr>
<tr>
<td>CMS</td>
<td>DT</td>
<td>206</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>446</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>534</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>GEI</td>
<td>221</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>602</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>1093</td>
<td>203</td>
</tr>
<tr>
<td>MT</td>
<td>DT</td>
<td>463</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>519</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>DNN</td>
<td>567</td>
<td>109</td>
</tr>
<tr>
<td></td>
<td>GEI</td>
<td>527</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>842</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>1250</td>
<td>157</td>
</tr>
</tbody>
</table>

**Embedding Dimension**: The input dimension of DT, KNN and DNN is 768, which is resulted by pooling operation on the original sentence embedding $z \in \mathbb{R}^{128 \times 768}$. While the input dimension of GEI and decoder is $\mathbb{R}^{15 \times 768}$, since only the first 15 tokens are targeted.

#### 3.8.4 Keyword Inference Results & Discussion

The results of white-box and black-box attacks are listed in Table 8. *Vanilla Decoder* and *Pre-trained Decoder* refer to our proposed attack and the rest models are baseline attacks. Note that ‘Count’ stands for the count of all reconstructed keywords and ‘Unique’ stands for the number of unique keywords. Furthermore, $k$, the number of kept words in our attack, is set $k = 20$. The ablation study of $k$ is shown in Section ??.

$k = 20$ is a relatively low threshold (a lower $k$ makes the attack more efficient) since the vocabulary of every dataset in our experiments contains thousands of tokens. Note that 20 is still smaller than 1% of the size of the vocabulary.

**Effectiveness of Attacks**: Even when $k$ is set as 20, the experiment results still highlight the effectiveness of our attack over the classifier-based attacks and GEI in both white-box and black-box settings. For example, in Table 8, our Pre-trained Decoder in white-box setting outperforms all the baselines on two metrics. The classifier-based baseline attacks achieve comparable results to our attacks on Skytrax dataset, which suggests that classifier-based attacks are effective in white-box setting where the private dataset and public dataset share the same distribution. Besides, DNN outperforms DT and KNN consistently.

Our Pre-trained Decoder exceeds baselines distinctively on CMS dataset. Specifically,
our decoder can identify 1093 keywords in total and 203 unique keywords given the BERT’s embeddings of the private medical transcriptions while DNN can only correctly predict 531 keywords in total and only 94 of them are unique. The performance of DNN is merely about 50% of our decoder’s. The experimental results imply that our decoder demonstrates better generalizability over different domains and embeddings.

The visualization of attack results in Table 8 are displayed in Figure 5. For each dataset, we select the top 10 most frequent keywords. Specifically, the 10 cities in Skytrax dataset are London, Paris, Bangkok, Toronto, Sydney, Hong Kong, Manchester, Dubai, Melbourne, and Singapore. As for the CMS dataset, the 10 medical terms are tissue, spinal, muscle, skin, bladder, heart, bone, blood, brain, and eye. Our Pre-trained Decoder outperforms the baselines with a distinctive margin on many keywords, which also supports that our decoder generalizes better.

**Comparison between Attack Settings:** It is noticeable that the numbers of keywords inferred in the black-box setting are much lower than the counterpart of white-box setting across all the attacks. As we have discussed, the black-box setting is more realistic where the adversary does not have information about the private dataset. Even if the adversary knows the domain of the dataset, it can still be challenging to
As shown in Table 9, Our Pre-trained Decoder achieves significant improvements of quantitative results on GPT. Results on GPT are shown in ( ).

<table>
<thead>
<tr>
<th>Target Dataset</th>
<th>Attack</th>
<th>White-box BERT(GPT)</th>
<th>Black-Box BERT(GPT)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cosine Order BLEU ROUGE</td>
<td>Cosine Order BLEU ROUGE</td>
</tr>
<tr>
<td>Skytrax</td>
<td>GEI</td>
<td>0.01 (.00) 0.10 (.01) 0.01 (.01)</td>
<td>0.01 (.02) 0.02 (.04) 0.00 (.00)</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>0.12 (.10) 0.20 (.12)</td>
<td>0.18 (.17) 0.20 (.18)</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder*</td>
<td>0.25 (.21) 0.52 (.51) 0.13 (.11) 0.12 (.12)</td>
<td>0.18 (.17) 0.40 (.37) 0.10 (.10) 0.11 (.10)</td>
</tr>
<tr>
<td>Twitter</td>
<td>GEI</td>
<td>0.05 (.00) 0.11 (.01) 0.01 (.01)</td>
<td>0.05 (.02) 0.05 (.04) 0.00 (.00)</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>0.11 (.09) 0.21 (.18) 0.04 (.03) 0.05 (.05)</td>
<td>0.05 (.03) 0.09 (.06) 0.02 (.01) 0.02 (.01)</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>0.22 (.20) 0.50 (.45) 0.11 (.10) 0.13 (.12)</td>
<td>0.15 (.14) 0.39 (.35) 0.08 (.07) 0.10 (.10)</td>
</tr>
<tr>
<td>CMS</td>
<td>GEI</td>
<td>0.12 (.10) 0.20 (.18) 0.02 (.01) 0.03 (.03)</td>
<td>0.05 (.03) 0.08 (.06) 0.00 (.00)</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>0.20 (.17) 0.35 (.31) 0.10 (.09) 0.12 (.11)</td>
<td>0.11 (.10) 0.30 (.26) 0.04 (.03) 0.05 (.05)</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>0.36 (.30) 0.59 (.53) 0.16 (.14) 0.19 (.15)</td>
<td>0.22 (.19) 0.41 (.38) 0.11 (.09) 0.11 (.09)</td>
</tr>
<tr>
<td>MT</td>
<td>GEI</td>
<td>0.12 (.11) 0.23 (.20) 0.01 (.01) 0.01 (.01)</td>
<td>0.05 (.05) 0.08 (.10) 0.01 (.00)</td>
</tr>
<tr>
<td></td>
<td>Vanilla Decoder</td>
<td>0.23 (.25) 0.46 (.49) 0.14 (.16) 0.15 (.16)</td>
<td>0.16 (.11) 0.19 (.24) 0.06 (.07) 0.05 (.05)</td>
</tr>
<tr>
<td></td>
<td>Pre-trained Decoder</td>
<td>0.38 (.42) 0.59 (.61) 0.20 (.22) 0.21 (.22)</td>
<td>0.19 (.17) 0.45 (.44) 0.12 (.13) 0.11 (.12)</td>
</tr>
</tbody>
</table>

define the keywords within the dataset. For instance, given a medical dataset, there may exist thousands of keywords (e.g., 1195 in CMS and 2377 in MT), which makes it extremely difficult to include all of them.

Therefore, the black-box setting is a more challenging setting, which understandably leads to poorer experimental results for all the attacks under experiment. Nevertheless, our Pre-trained Decoder remains the most robust attack method, especially on the Twitter dataset and GPT-2 embedding. The results suggest that our attack handles this setting better due to its flexibility and generalizability.

**Transferability of Attacks:** According to Table 8, the Pre-trained Decoder displays much better transferability in all the cases. To be specific, the gap between Pre-trained Decoder and DNN is further enlarged in both domains compared with results of Table 8.

For example, all baseline attacks completely fail on the Twitter dataset with GPT-2 embeddings. However, our decoder still memorizes 30 city names in total, and there exist 5 unique city names. When it comes to the MT dataset, our decoder memorizes more than double the total medical terms/unique medical terms that captured by DNN. Based on the above observations, we can safely conclude that our decoder continues to behave more robustly in the black-box setting. Its better transferability makes it a more powerful threat to the real-world systems.

**The Impact of Pre-training** Pre-training has boosted the performance of the decoder in all cases significantly. Specifically, pre-trained decoder has identified at least at least 30% more keywords in total than the vanilla decoder. The gap of the number of unique keywords is distinctive too. The results imply that the pre-training is a robust way to strengthen both the generalizability and transferability of the decoder.

### 3.8.5 Sentence Inference Results & Discussion

Note that Table 9 displays the results for sentence inference attack. “cosine” stands for cosine similarity and “order” stands for word order similarity. Due to the limitation of the classifier-based attacks, they are removed from this attack setting.

**Quantitative Results**

As shown in Table 9, Our Pre-trained Decoder achieves significant improvements of...
not only cosine similarity but also word order similarity over GEI on both white-box and black-box settings.

One of the key observations is that GEI is biased on the empty tokens such as “[PAD]”, which yields final sentences with large proportion of noises. This phenomenon is caused by the contextual learning in language models. The embeddings of pre-trained language models are highly convoluted due to self attention mechanisms [1]. The same words in different contexts will be transformed into different deep embeddings. Therefore, the shallow mapping function, which maps deep embeddings to lower space, might be biased with embedding variance of each word, which leads to biased recovered sentences.

Compared with the mapping function, our decoder utilizes dependencies in sentences to invert deep embeddings rather than solve each word independently. As a result, our method is capable of recovering much more coherent and informative sentences from the embeddings, therefore capture semantic information.

**Comparison between Attack Settings** We can observe that the cosine similarity scores drop from white-box setting to black-box setting. For example, the mean cosine similarity drops from 0.30 to 0.17 on CMS and GPT-2 embeddings.

Similar to the observation in keyword inference attack, there may exist unseen sentence structures and patterns in the black-box setting, which challenges the flexibility of the attacks. Our Pre-trained Decoder still remains relatively robust in black-box setting, indicating it is a more practical privacy threat.

**Comparison between Metrics** In addition, the mean and std of cosine similarity in various configurations do not necessarily agree with the metrics of keyword inference attack according to Table 8. For example, the mean cosine similarity and the count of memorized keywords are 0.25 and 835, respectively, on Skytrax and BERT embeddings. Although the mean cosine similarity is 0.21 in Skytrax and GPT embeddings, the actual count of keywords is 1190, which is higher than 835. The reason behind this situation is that the cosine similarity measures the degree to which two sentences overlap, therefore, it does not focus on any keywords. Higher similarity means a higher number of words in the reconstructed sentence also exist in the original sentence.

**Qualitative Results** To demonstrate the capacity of sentence inference attack, 10 reconstructed sentences are displayed in Table 4. For instance, the reconstructed sentence ”Vaccine pneumonia
influenza virus nasal” can allow the adversary to accurately infer that this sample is "vaccine for pneumonia/influenza for nasal". Another pair of examples is "Rome to Toronto July 2013" vs. "Rome to Toronto 2013". The decoder has captured most of the information accurately except the month. Noticeably, the pair of "Transplantation of donor kidney" versus "Transplant donor' kidney" shows that the decoder has learned to use contraction during training, which implies it captures the underlying language patterns within the pretraining dataset. Therefore, sentence inference attack can threaten the NLP systems without any knowledge about the private dataset. The adversary can infer the semantics of the original sentence given a similar reconstructed sentence. In addition to the reconstructed sentences in Table 10, we report a few randomly picked examples as shown in Table ???. We observe lower coherence and fluency in these examples. This is a limitation of our attack method and we further discuss this in Section 3.10. However, note that, such reconstructions may still leak information through their structures that are similar to the original sentences. E.g., the reconstructed “manual test of hand arm behind leg” of the original “manual muscle test of arm, leg or trunk” achieves a BLEU score (2-gram) of 0.30 and a WOS of 0.55. Although the BLEU score is relatively low, the adversary can still infer the tested body regions of the subject.

The Impacts of Pre-training In sentence inference attack, pre-training still makes the decoder generate more similar text than the vanilla decoder. We can conclude that pre-training does not only make the decoder more sensitive to keywords but also increase the accuracy of reconstruction.

3.9 Potential Defenses

3.9.1 Differential Privacy

Differential Privacy (DP) is a popular technique for protecting information about individuals in the dataset [100]. In the domain of machine learning, a differentially private stochastic gradient descent algorithm [101] has been proposed to reduce the risk of privacy. Google has already applied DP to large-scale image classification systems while maintaining high accuracy and minimizing computational cost [102]. The trade-off between utility and information leakage has been further investigated [103]. The main disadvantage of ensuring differential privacy is that it typically requires more noise infusion than traditional techniques.

As for the language modeling, it is demonstrated that DP can be used to train privacy-preserving models in various NLP applications [104, 105]. To satisfy the DP algorithm, each training sample in the dataset requires a user label. This requirement can be challenging for pre-trained language models since their training data is usually scraped from the public Web.
3.9.2 Privacy Preserving Mapping

The inference attacks against sentence embeddings are based on the key idea that public embeddings and private embeddings belong to the same embedding space. Privacy Preserving Mapping (PPM) provides a way to distort the embeddings before they are accessible to the third party [106]. On the one hand, PPM is trained to minimize the effectiveness of an inference attack by quantifying privacy leakage. On the other hand, to preserve the utility of the embeddings, the distortion of the PPM is constrained by a bound.

As a result, PPM can be applied to private embeddings so that attack models trained on public embeddings will suffer from distorted embedding space.

3.9.3 Avoid Providing Complete Sentence Embeddings

If an organization needs to share the embeddings of its confidential data with a third-party service provider, it can only provide the pooled version of the sequential embeddings or a masked version of the sequential embeddings. The incomplete sentence embeddings can reduce effectiveness of our decoder-based inference attack in accurately reconstructing the text. However, the performance of some downstream tasks such as machine translation and named entity recognition will also degrade.

3.9.4 Evaluation of Potential Defenses

We evaluate privacy preserving mapping (PPM) and incomplete embedding defenses against the keyword inference attack on the Skytrax dataset. Besides, we consider entity recognition as the downstream task to demonstrate the effects of defenses. Formally, given a sequence of embeddings $z = [z_1, z_2, ..., z_n]$ and a sequence of labels $y = [y_1, y_2, ..., y_n], y \in (0, 1)$, where 1 stands for targeted keywords and 0 stands for other words, the goal is to identify all the entities labeled with 1. We train a single-layer RNN model with a hidden state size of 300 to perform entity recognition. The performance is measured by the F-1 score, which represents the utility of the task. We report the count of reconstructed keywords (normalized) as information leakage.
For PPM, we follow the setup in [11]. Given a mapping \( D_\theta : \mathcal{R}^d \Rightarrow \mathcal{R}^d \) which is trained to minimize the effectiveness of an imaginary adversary \( A_\phi \), formally, the learning objective is a minimax game by solving \( \min_\theta \max_\phi \sum A_\phi(D_\theta(z)) + \lambda ||D_\theta(z) - z|| \) [106]. The PPM is implemented as a regularization term in the minimax game so that the distortion of the embeddings is only allowed in a limited radius. Note that a high value of \( \lambda \) leads to a lower privacy budget. For incomplete embeddings, we apply a randomly generated mask to the embeddings [107], with the masking rate ranging from 0.1 to 0.9. Higher masking rate leads to sentence embeddings with more unknown tokens.

As shown in Figure 6, although both defenses can mitigate our decoder-based attack, they inevitably compromise the utility of the downstream task. We can observe the trade-off between utility and privacy for both of the defenses, which suggests that more sophisticated defense mechanisms that do not compromise utility to this extent need to be explored.

### 3.10 Discussion

#### 3.10.1 Practicality of Decoder-based Attack

We show that a transformer decoder can reconstruct coherent and informative texts, therefore revealing sensitive information. Compared to a prior classifier-based attack, it is a more practical threat, since the adversary does not need to know the secrets within the dataset of interest. However, the classifier-based attack requires the adversary to know the keywords in the dataset or have the experience to create a set of keywords. Note that it is not practical to make the above assumption in many domains (e.g., medical, financial, industrial, and more).

As for the gradient-based attack, it cannot decode the contextual language embeddings and therefore produce noise outputs. Our method inverts the deep embeddings more accurately to generate well-structured sentences.

Finally, our method can still achieve robust performances in a black-box setting, while the baselines’ performances degrade significantly. Since the black-box setting is closer to the real world, the decoder-based attack represents a practical threat to NLP applications.

#### 3.10.2 Limitations

**Reconstruction from Embeddings is Hard** We have demonstrated that the reconstructed texts can reveal high-level semantic information. Although our method can reconstruct high-quality sentences in many cases, it often produces lower-quality sentences as well. Both the proposed decoder-based attack and the previous gradient-based attack [12] rely on the accurate prediction of the probability distribution of each word. Hence, there are two challenges associated with reconstruction: (1) the word probability is usually distributed over a large vocabulary (thousands of tokens), which makes it hard to guarantee the right word is going to be selected, (2) it is difficult
to reconstruct long texts. The second challenge is due to the fact that as the reconstruction goes further, the dependencies between current words and previous words decrease, which leads to less accurate results.

**Implementation limitations** Although we have shown a successful decoder-based attack, there are several limitations of this work that could be explored in the future:

1. We have only tested the transformer decoder in our experiments. The performance of other architectures such as RNN [108] may provide more insights.
2. The hyperparameters (e.g., depth, learning rate, number of heads) of the decoder follow the default setting, which could have been improved by grid search.

### 3.10.3 Future Work

To improve the overall fluency and coherence of the reconstructions produced by our attack, we discuss the following future directions.

**Pre-training the Decoder on Large Corpus** We have demonstrated the impacts of pre-training in previous results. The pre-trained decoder outperforms vanilla decoder on both keyword inference attack and sentence inference attack. However, the size of the dataset $D_{\text{train}}$ is relatively small compared to the size of the training data from pre-trained language models. The generalizability of the decoder can be further improved by pre-training on a large corpus. This has the potential to boost the quality of reconstructions as well. For example, there exists a gold standard dataset in the medical domain, namely, MIMIC-III [109]. MIMIC-III includes more than 1 million caregiver notes of thousands of patients, which can be utilized by the adversary to pre-train the decoder. Such a pre-trained decoder can threaten many applications in the medical domain.

**Upgrading Decoder Architecture** Currently, our decoder is inherited from the transformer’s decoder, which exhibits the capacity of generating high-quality text. However, with the development of language models, more advanced decoder architectures are emerging. For example, Transformer-XL [110] was proposed to learn longer-term dependencies, while the vanilla transformer is limited by the fixed-length context required by the input. Therefore, Transformer XL can generate more coherent text. If the adversary adopts such an advanced decoder, the quality of the reconstructions will likely be enhanced without any other modifications.

**Improving Quality of Decoding** To further improve the quality of the reconstructed text, we have employed various approaches, including top-k sampling, decaying temperature, and repetitive generation. However, there are more factors to consider, such as fluency and coherence of the language. Pascual et al. [111] presented a plug-and-play encoding method: Given a keyword or a topic, it added a shift to the probability distribution over the vocabulary towards semantically similar words. Despite the simplicity of this approach, it still enabled GPT-2 to generate more diverse and fluent sentences while guaranteeing the appearance of given guide words. The adversary can employ the plug-and-play method to improve the coherence of the decoding.

**Handling Acronyms and Numbers** Acronyms and numbers may carry sensitive
information (e.g., airline code and medical terms). However, it is challenging to reconstruct those accurately. Besides the aid of pre-training, a more sophisticated way to represent numbers [112] or acronyms [113] may benefit the decoding quality.
4 Unlearning Large Language Models

4.1 Preliminary

We first revisit the notion of unlearning from the perspective of KL divergence. Then we further discuss unlearning strategies for convex and non-convex models respectively. The taxonomy is defined in Table 11.

Table 11: Parameter Definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D^-$</td>
<td>data samples to be removed</td>
</tr>
<tr>
<td>$D^+$</td>
<td>data samples left</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>full training data $\mathcal{D} = D^+ \cup D^-$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>model parameters</td>
</tr>
<tr>
<td>$l$</td>
<td>coefficient for gradient ascent</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>parameter for initializing Empirical Fisher</td>
</tr>
<tr>
<td>$m$</td>
<td>the number of recursions for Empirical Fisher</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>coefficient for Fisher Removal</td>
</tr>
<tr>
<td>$\mu$</td>
<td>parameter for Fisher Forgetting</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>parameter for Fisher Forgetting</td>
</tr>
</tbody>
</table>

4.1.1 Unlearning for General Models

Let $\theta$ denote the parameters of a model trained on $\mathcal{D} = D^- \cup D^+$ and $S(\theta, D)$ be an unlearning function that modifies the parameters such that $\theta' = S(\theta, D)$. The goal of unlearning is to ensure an adversary with access to $\theta'$ cannot reconstruct information about $D^-$ via some readout function $f(\cdot)$. 

**Definition 1.** Given an optimal unlearning algorithm $S_1(\theta, D)$, there is another function $S_2(\theta, D^+)$ that does not depend on $D^-$ such that:

$$KL(\mathcal{P}(f(S_1(\theta, D))) || \mathcal{P}(f(S_2(\theta, D^+)))) = 0$$

(22)

Where $KL$ stands for Kullback–Leibler divergence and $\mathcal{P}(\cdot)$ is the distribution of unlearned weights. Intuitively, the optimal unlearning outcome of $S_1(\theta, D)$ should be indistinguishable from a model that has never seen $D^-$. Satisfying this condition may be trivial, e.g. letting $S_1(\theta, D) = S_2(\theta, D^+) = c$ be constant. The challenge lies in how to do so while preserving as much knowledge as possible about $D^+$. We may define this relation from the perspective of Shannon Information [18]:

**Proposition 1** Let the forgetting set $D^-$ be a random variable such as a random sampling of $D$. Let $Y$ be an attribute of interest for $D^-$. 

$$I(Y; f(S_1(\theta, D))) \leq \mathbb{E}_{D^-}[KL(\mathcal{P}(f(S_1(\theta, D))) || \mathcal{P}(f(S_2(\theta, D^+))))]$$

(23)
Where $I(\cdot)$ represents mutual information. In general, we may not know what reconstruction function an adversary will use, and hence this relation should be robust against every $f(\cdot)$. therefore, the following lemma is more practical:

**Lemma 1** For every $f(\cdot)$, we have:

$$KL(P(f(S_1(\theta, D)))||P(f(S_2(\theta, D^+)))) \leq KL(P(S_1(\theta, D)||P(S_2(\theta, D^+)))$$

Thus, we should focus on minimizing the quantity:

$$KL(P(S_1(\theta, D)||P(S_2(\theta, D^+)))$$

which guarantees robustness to any readout function. Now we can show how to derive unlearning strategies using this idea.

### 4.1.2 Unlearning for Convex Models

If the loss function is convex, e.g. quadratic, we can construct the following unlearning strategies:

$$S_1(\theta, D) = h(A(D, \epsilon)) + n$$

$$S_2(\theta, D^+) = A(D^+, \epsilon) + n'$$

where $A(D, \epsilon)$ stands for a stochastic training algorithm with random seed $\epsilon$, $n, n' \sim N(0, \Sigma)$ is Gaussian noise and $h(\cdot)$ is a deterministic function. We have $P(S_1(\theta, D) \sim N(h(A(D, \epsilon)), \Sigma)$ and $P(S_2(\theta, D^+)) \sim N(A(D^+, \epsilon), \Sigma)$. Then we have the following:

$$KL(P(S_1(\theta, D)||P(S_2(\theta, D^+))) \leq \frac{1}{2}E_\epsilon[U^T \Sigma^{-1} U]$$

Where $U = h(A(D, \epsilon)) - A(D^+, \epsilon)$.

This means that we can derive an upper bound for the complex KL quantity $KL(P(S_1(\theta, D)||P(S_2(\theta, D^+)))$ by simply averaging the results of training and unlearning with different random seeds. In fact, this KL quantity converges to zero if we simply apply the Newton update. Details can be found in Appendix.

### 4.1.3 Newton Update

Without loss of generality, we aim to remove the last training sample $(x_n, y_n)$. Let the full dataset be denoted by $D$ and the left dataset $D^+ = D \setminus x_n$. Firstly, the loss gradient with respect to $x_n$ can be calculated by $\Delta = \nabla L(x_n, y_n)$, where $L$ is usually cross entropy loss and $\theta$ is the current model weight. Secondly, the Hessian of $L(\cdot, D^+)$ at $\theta$ is $H_\theta = \nabla^2 L(\theta, D^+)$. Finally, the Newton update removal mechanism is as follows:

$$S_1(\theta, D) = \theta + H_\theta^{-1} \Delta$$

(26)
For shallow (convex) machine learning models, Newton update can perform unlearning with a bounded approximation error [17].

4.1.4 Gradient Ascent

An intuitive way to perform unlearning is inverting the direction of gradients. Jang et al. [13] first studied this approach in the context of LLMs and we also include it due to its efficiency. However, it has not yet been proven whether the gradient ascent is robust in maintaining accuracy on $D^+$. As the opposite of the original training objective, gradient ascent (Algorithm 2) aims to maximize the loss function such that the knowledge of certain samples is destroyed.

Algorithm 2 Gradient Ascent for LLM

1: procedure Gradient Ascent($D^-, \theta, l$)
2: for $i = 1, 2, \ldots, n$ do
3: \hspace{1cm} $\Delta \theta \leftarrow L(B_i$, $\theta)$ \hspace{1cm} \triangleright Gather the gradient for one batch in $D^-$
4: \hspace{1cm} $\theta \leftarrow \theta + l \times \Delta \theta$ \hspace{1cm} \triangleright Update parameters
5: end for
6: end procedure

4.2 Unlearning for LLMs

However, LLMs [114] as well as other deep neural networks usually do not satisfy the convexity. To relax the assumption, we exploit the limited degree of freedom by introducing noise into the general unlearning schema as discussed in Section 4.1.2. Consider noisy Newton update as follows:

$$S_1(\theta, D) = \theta + H^{-1}_{\theta} \Delta + (\mu \sigma^2)^{\frac{1}{4}} H^{-\frac{1}{4}}_{\theta} \quad (27)$$

Here, $\mu$ controls the trade-off between residual information about the forgetting subset $D^-$, and accuracy on the remained data $D^+$. $\delta$ reflects the error in approximating the SGD behavior with a continuous gradient flow. However, to calculate Hessian analytically requires $O(d^2)$ time, where $d$ is the size of the weight. For example, the embedding layer in GPT-2 small version has a size of $50257 \times 768 = 38597376$. Calculating the Hessian for this weight alone will require over $10^{15}$ operations. Not to mention that calculating the inverse of a matrix typically requires at least $O(d^{2.373})$ time [115].

4.2.1 Inverse Empirical Fisher

Therefore, we estimate the inverse Hessian of LLMs via inverse empirical Fisher. Specifically, we employ the Woodbury/Sherman-Morrison (WSM) formula [116]. Given a sum of $A + uv^T$ an invertible matrix $A$ and an outer product of vectors $u$ and $v$, the
inverse \((A + uv^T)^{-1}\) can be exactly calculated as \(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\). Combining the formula with empirical Fisher, we can obtain the following recursive formulation:

\[
F_{m}^{-1}(w) = (F_{m-1}(w) + \frac{1}{m}\nabla \mathcal{L}(w, \hat{D})\nabla^T \mathcal{L}(w, \hat{D}))^{-1}
\] (28)

Where \(m\) is the number of gradients used for the approximation, equal to the size of \(\hat{D}\). With \(F_0^{-1}(w) = \frac{1}{\lambda}I_d\), the above recursion can be rewritten as:

\[
F_{m}^{-1}(w) = \frac{1}{\lambda}I_d - \sum_{i=1}^{m} \frac{(F_{i-1}^{-1}(w)\nabla \mathcal{L}_i(w))(F_{i-1}^{-1}(w)\nabla \mathcal{L}_i(w))^T}{m + \nabla \mathcal{L}_i(w)^T F_{i-1}^{-1}(w) \nabla \mathcal{L}_i(w)}
\] (29)

Here, \(\lambda\) is an initialization parameter for \(F_0^{-1}\).

### 4.2.2 Efficient GPU Implementation

It can even be prohibitively expensive to compute and store empirical Fisher \(F^{-1}(w) \in \mathbb{R}^{d \times d}\) for LLMs due to the huge size of the model weights. However, by adapting the diagonal block-wise trick [116], empirical Fisher can be accurately approximated. Specifically, we can only focus on blocks of width \(B\) along the main diagonal, which brings down the computation from quadratic \(O(d^2)\) to linear \(O(Bd)\).

On the implementation side, we have identified general hyper-parameters \(B = 48\) for the block size, and \(m = 1024\) for the maximum number of recursions which yield satisfying performances. A 16GB GPU can accommodate a 125M model (e.g. GPT2-small) with \(B = 48\). As for large LLMs, we have designed a multi-GPU strategy to store Fisher matrices on different GPUs, which makes this approach scalable to state-of-the-art LLMs.

### 4.2.3 Fisher Removal

Although gradient descent is highly efficient, it does not account for information from \(D^+\). Following the idea of noisy Newton update, we propose Fisher removal, which combines second-order information obtained from \(D^+\) with first-order information obtained from \(D^-\). Formally, Fisher Removal is formulated as below:

\[
S_1(\theta, D) = \theta + \gamma \hat{H}_\theta^{-1} \Delta
\]

where unlearning rate \(\gamma\) is adopted to adjust the unlearning effects for LLMs.
Algorithm 3 Fisher Removal for LLM

1: \textbf{procedure} Fisher Removal($D^-, D^+, \theta, \lambda, \gamma$)
2: \hspace{1em} \textbf{for} i=1,2,...,n \textbf{do} \hspace{1em} \Comment{Iterate through batches in $D^-$}
3: \hspace{2em} $\Delta \theta \leftarrow L(B_i^-, \theta)$ \hspace{1em} Comment{Gather the gradient for one batch in $D^-$}
4: \hspace{2em} $F_0^{-1} = \lambda^{-1} \ast I_d$ \hspace{1em} Comment{Initialize Fisher Inverse with proper shape}
5: \hspace{1em} \textbf{for} j=1,2...m \textbf{do} \hspace{1em} \Comment{Iterate through batches in $D^+$}
6: \hspace{2em} $\Delta B_j^+ \leftarrow L(B_j^+, \theta)$ \hspace{1em} Comment{Gather gradient for one batch in $D^+$}
7: \hspace{2em} $F_j^{-1} \leftarrow \text{update}(F_{j-1}^{-1}, \Delta B_j^+)$ \hspace{1em} Comment{Invoke Empirical Fisher recursion}
8: \hspace{1em} \textbf{end for}
9: \hspace{1em} $\hat{H}^{-1} = F_m^{-1}$
10: \hspace{1em} $\theta \leftarrow \theta + \gamma \hat{H}^{-1} \Delta \theta$ \hspace{1em} \Comment{Update parameters}
11: \hspace{1em} \textbf{end for}
12: \textbf{end procedure}

4.2.4 Fisher Forgetting

Since $\hat{H}^{-1}$ is an empirical approximation, it introduces additional uncertainty into the unlearning process, which can cause an unexpected loss of accuracy on $D^+$. Therefore, we present Fisher Forgetting to perturb the memory of $D^-$ by adding Gaussian noise to the neurons [18].

$$S_1(\theta, D) = \theta + (\mu \sigma^2) \frac{1}{2} H_\theta^{-\frac{1}{2}} \odot M$$

Where $M \sim \mathcal{N}(0, 1)$ is Gaussian noise, $\mu, \sigma$ are parameters introduced in Section 4.2.
Algorithm 4 Fisher Forgetting for LLM

1: \textbf{procedure} Fisher\_Forgetting($D^-, D^+, \theta, \lambda, \mu, \sigma$) \hfill \triangleright Iterate through batches in $D^-$
2: \textbf{for} i=1,2,...,n \textbf{do} \hfill \triangleright Gather the gradient for one batch in $D^-$
3: \hspace{1em} $\Delta \theta \leftarrow L(B_i^-, \theta)$ \hfill \triangleright Gather gradient for one batch in $D^-$
4: \hspace{1em} $F_0^{-1} = \lambda^{-1} \ast I_d$ \hfill \triangleright Initialize Fisher Inverse with proper shape
5: \textbf{for} j=1,2...m \textbf{do} \hfill \triangleright Iterate through batches in $D^+$
6: \hspace{1em} $\Delta B_j^+ \leftarrow -L(B_j^+, \theta)$ \hfill \triangleright Gather gradient for one batch in $D^+$
7: \hspace{1em} $F_j^{-1} \leftarrow \text{update}(F_{j-1}^{-1}, \Delta B_j^+)$ \hfill \triangleright Invoke Empirical Fisher recursion
8: \textbf{end for}
9: $\hat{H}^{-1} = F_m^{-1}$
10: $M \sim \mathcal{N}(0, 1)$ \hfill \triangleright Sampling Gaussian Noise
11: $\theta \leftarrow \theta + \left((\mu \sigma^2)\frac{1}{2} \hat{H}^{-\frac{1}{2}} \circ M\right)$ \hfill \triangleright Update parameters
12: \textbf{end for}
13: \textbf{end procedure}

Property of Fisher Forgetting

Assuming $\|H^{-\frac{1}{2}}_\theta\|_2 \leq 1$, which can be easily satisfied if $\lambda$ is set to 1 when initializing $F_0^{-1}$, we can derive the following property:

$$\|((\mu \sigma^2)^{\frac{1}{2}} H^{-\frac{1}{2}}_\theta \circ M)\|_2 \leq (\mu \sigma^2)^{\frac{1}{2}} \|M\|_2$$

Considering Fisher Forgetting has been performed $k$ times in a roll on a LLM parameterized by $\theta$, the sum of all $k$ updates to the weights can be calculated by:

$$\sum_{i=1}^{k} ((\mu \sigma^2)^{\frac{1}{2}} H^{-\frac{1}{2}}_\theta \circ M_i) = \sum_{i=1}^{k} \mathcal{N}(0, ((\mu \sigma^2)^{\frac{1}{2}} H^{-\frac{1}{2}}_\theta)^2))$$

Here $M_i$ are i.i.d Gaussian variables $\sim \mathcal{N}(0, 1)$. By the property of Gaussian distributions:

$$\sum_{i=1}^{k} \mathcal{N}(0, ((\mu \sigma^2)^{\frac{1}{2}} H^{-\frac{1}{2}}_\theta)^2)) \sim \mathcal{N}(0, \sum_{i=1}^{k} ((\mu \sigma^2)^{\frac{1}{2}} H^{-\frac{1}{2}}_\theta)^2))$$

Again, with $\|H^{-\frac{1}{2}}_\theta\|_2 \leq 1$, the quantity can be approximated as $\mathcal{N}(0, k(\mu \sigma^2)^{\frac{1}{2}})$. This means the cumulative updates of $k$ consecutive Fisher Forgetting can be bounded by a new Gaussian distribution $\mathcal{N}(0, k(\mu \sigma^2)^{\frac{1}{2}})$, where $\mu, \sigma << 1$. As a result, the model performance can be largely preserved.

47
4.3 Experimental Setup

Figure 7 illustrates how a service provider handles incoming removal requests in our setting. In the life cycle of an LLM service, such requests for removal are likely to arrive stochastically. To protect users’ right to be forgotten, the server may need to perform unlearning more than once, depending on the predetermined terms.

In our experiments, we consider two scenarios of unlearning: 1) perform only one unlearning cycle (128 samples), 2) perform two unlearning cycles (256 samples) consecutively.

![Figure 7: The cycle of unlearning of the LLM service. The model has been trained on the user database for downstream tasks. Since user requests arrive stochastically, a cache is used to store them. Once the cache is full or the waiting time has exceeded the threshold, the server will invoke the unlearning process and clean the cache to free up space. Without the loss of generality, we assume the size of the cache is 128 in our experiments.](image)

4.3.1 Datasets

We evaluate LLMs on four general NLP datasets to show the trade-offs between unlearning effects and resulting performances on downstream tasks.

**ARC-Easy.** This dataset contains 5000 in grade-school level, multiple-choice scientific questions. For example, given the question of “Which piece of safety equipment is used to keep mold spores from entering the respiratory system?” and options “[A: safety goggles, B: breathing mask, C: rubber gloves, D: lead apron]”, the answer is B. The dataset is split into train/val/test sets with 2251, 570, and 2376 questions respectively.

**Piqa.** It is a binary physical question-answering dataset. To solve the questions correctly, the models are required to understand physical commonsense. An example question is “To separate egg whites from the yolk using a water bottle, you should ...”, and the answer is “Squeeze the water bottle and press it against the yolk. Release, which creates suction and lifts the yolk.” This dataset consists of 16113 and 1838 questions for training and validation respectively.

**MathQA.** It is a large-scale dataset of math world problems, which has provided the questions, options, answers as well as rationales. The train/val/test splits have 29837, 4475, and 2985 questions.
**Lambada.** This dataset evaluates the language modeling capacity by the task of last-word prediction. Human subjects can guess the last word with at least 50 tokens of context, which requires long-term dependencies. The dataset contains 10022 passages from BookCorpus and is further divided into 4869 for development and 5153 for testing.

### 4.3.2 Evaluation Metrics

We propose to evaluate unlearning algorithms from three dimensions: 1) the removal of selected data, 2) the preservation of model utility, and 3) the used time as in [13]. For simplicity, let $\theta$ be the parameters of an initial language model requiring unlearning, $D^-$ be a batch of data to unlearn, $D_{test}$ be a standalone test dataset, $U$ be an unlearning algorithm, and $\theta'$ be the model after being unlearned.

**Efficacy.** The most important property of an unlearning algorithm is ensuring successful removal. Although certified unlearning [17] can guarantee this removal, there is no such guarantee for non-convex models. Perplexity or validation loss may not be sufficient to distinguish $\theta$ and $\theta'$ [117]. Therefore, we refer to exposure metrics [118] to measure the efficacy of unlearning quantitatively.

$$
exp_\theta(s) = \log_2(|Q|) - \log_2 \text{rank}_\theta(s)
$$

Here, $s$ is a text sample and $Q$ is a set of possible sequences with the same length. The $\text{rank}_\theta(s)$ function returns the rank of $s$ given model $\theta$ and the set $Q$.

Based on the exposure metrics of a single sample, we propose $\Delta_{exp}(\theta, \theta')$ to measure the change of expected exposure values after unlearning is performed.

$$
\Delta_{exp}(\theta, \theta') = E_{s \in D^-} [exp_{\theta'}(s) - exp_\theta(s)]
$$

The above metrics reflect the extent to which the unlearning algorithm $U$ erases the set $D^-$.

**Approximating Exposure:** Given the length $l$, it is not practical to iterate through all combinations of tokens in the model vocabulary $V$, which leads to $|V|^l$ possibilities. We choose to approximate the exposure by sampling [118]. Specifically, we refer to a standalone corpus and sample a set $S$ of sequences with length $l$. $S$ can be viewed as a random sampling of the full space $Q$ such that $|S| \ll |Q|$.

**Fidelity.** Although the removal of $D^-$ matters, the unlearning algorithm $U$ should also largely protect the performance of the model $\theta'$. Without loss of generality, we use accuracy to evaluate model performance. Formally, let $\text{acc}_\theta(D_{test})$ stand for the accuracy of model $\theta$ on test set $D_{test}$, $\text{acc}_{\theta'}(D_{test})$ is expected to be close to the initial model $\theta$. Similarly, we propose $\Delta_{acc}(\theta, \theta')$ to reflect the change of accuracy.

$$
\Delta_{acc}(\theta, \theta') = \text{acc}_{\theta'}(D_{test}) - \text{acc}_\theta(D_{test})
$$

**Efficiency.** Finally, the unlearning algorithm should be efficient compared with re-
training. As a result, we show both the time complexity and actual running time of each unlearning method in our experiments.

4.3.3 Unlearning Settings

**Target LLMs.** We test the unlearning methods on both GPT-Neo [119] and OPT LMs [21] as in [13]. For each model, three variants (125M, 1.3B, 2.7B) are included.

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Engineering</th>
<th>Architecture Engineering</th>
<th>Reproducible/Open-Sourced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retraining</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Finetuning</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>SISA [120]</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Revise &amp; Finetuning [14]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>EUL [15]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Ours*</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Unlearning Baselines.** We show the comparison of selected unlearning methods in Table 12. Only methods that can be reproducible on any dataset or LLM are considered in our experiments. However, we exclude SISA [120] because dividing the training data into disjoint shards and training a LLM on each shard will compromise the performance of LLMs, especially when the number of shards is large. As a result, in addition to the aforementioned unlearning Algorithm 2-4, we also consider retrain and finetuning as unlearning baselines:

- **Retraining:** We discard the current model and retrain the LLM using $D^+$ for 5 epochs. It is the benchmark for our evaluation of other baselines.

- **Finetuning:** We continue to finetune the current model on $D^+$ for 1 epoch. Intuitively, the model’s memory of $D^-$ will be weakened.

4.3.4 Hyperparameters

- For all the LLMs in our experiments, we set the coefficient for gradient ascent $l$ as $5e^{-5}$, which is the same as the learning rate. As for second-order methods, $\gamma$ is set as $2.5e^{-4}$. $\mu$ and $\sigma$ are both set to $1e^{-3}$.

- For empirical Fisher, $\lambda$ and $m$ are set to 1 and 1024, respectively.

- To implement exposure metrics, we refer to the European Court of Human Rights (ECHR) corpus\(^{11}\). The size of subspace SS is set to 10001000.

Note that $\gamma$, $\mu$ and $\sigma$ are chosen through grid search where $\gamma, \mu, \sigma \in [1e^{-4}, 1e^{-3}]$.

\(^{11}\)https://archive.org/details/ECHR-ACL2019
### Table 13: Unlearning Results on ARC Dataset

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unlearning</th>
<th>128 samples</th>
<th>256 samples</th>
<th>128 samples</th>
<th>256 samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NEO</td>
<td>OPT</td>
<td>NEO</td>
<td>OPT</td>
</tr>
<tr>
<td></td>
<td>∆ACC</td>
<td>∆EXP</td>
<td>∆ACC</td>
<td>∆EXP</td>
<td>∆ACC</td>
</tr>
<tr>
<td>Retraining</td>
<td></td>
<td>-0.50</td>
<td>-0.03</td>
<td>+1.14</td>
<td>-0.10</td>
</tr>
<tr>
<td>Finetuning</td>
<td>-0.50</td>
<td>+0.17</td>
<td>+0.27</td>
<td>+0.07</td>
<td>+1.20</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-3.50</td>
<td>-3.03</td>
<td>-10.29</td>
<td>-3.32</td>
<td>-12.9</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-0.36</td>
<td>-1.95</td>
<td>-5.62</td>
<td>-4.84</td>
<td>-2.65</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.07</td>
<td>-0.06</td>
<td>-0.17</td>
<td>-0.02</td>
<td>-0.85</td>
</tr>
<tr>
<td>125M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retraining</td>
<td>-0.35</td>
<td>-0.15</td>
<td>-1.52</td>
<td>-0.05</td>
<td>+1.15</td>
</tr>
<tr>
<td>Finetuning</td>
<td>+1.15</td>
<td>+0.13</td>
<td>-0.70</td>
<td>+0.05</td>
<td>+1.47</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-0.16</td>
<td>-0.14</td>
<td>-1.03</td>
<td>-0.03</td>
<td>-10.35</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>+0.22</td>
<td>-0.16</td>
<td>-5.13</td>
<td>-0.84</td>
<td>-2.28</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.22</td>
<td>-0.03</td>
<td>-0.41</td>
<td>-0.01</td>
<td>-0.27</td>
</tr>
<tr>
<td>1.3B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retraining</td>
<td>-0.33</td>
<td>-0.15</td>
<td>-0.49</td>
<td>-0.07</td>
<td>-0.76</td>
</tr>
<tr>
<td>Finetuning</td>
<td>-0.44</td>
<td>+0.01</td>
<td>-1.19</td>
<td>+0.12</td>
<td>-1.19</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-0.33</td>
<td>-0.28</td>
<td>-5.11</td>
<td>-0.27</td>
<td>-6.97</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-1.36</td>
<td>-0.79</td>
<td>-6.97</td>
<td>-1.12</td>
<td>-6.08</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.60</td>
<td>-0.00</td>
<td>-0.45</td>
<td>-0.01</td>
<td>-0.63</td>
</tr>
<tr>
<td>2.7B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 14: Unlearning Results on Piqa Dataset

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unlearning</th>
<th>128 samples</th>
<th>256 samples</th>
<th>128 samples</th>
<th>256 samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NEO</td>
<td>OPT</td>
<td>NEO</td>
<td>OPT</td>
</tr>
<tr>
<td></td>
<td>∆ACC</td>
<td>∆EXP</td>
<td>∆ACC</td>
<td>∆EXP</td>
<td>∆ACC</td>
</tr>
<tr>
<td>Retraining</td>
<td></td>
<td>-0.50</td>
<td>-0.03</td>
<td>+1.14</td>
<td>-0.10</td>
</tr>
<tr>
<td>Finetuning</td>
<td>-0.50</td>
<td>+0.17</td>
<td>+0.27</td>
<td>+0.07</td>
<td>+1.20</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-3.50</td>
<td>-3.03</td>
<td>-10.29</td>
<td>-3.32</td>
<td>-12.9</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-0.36</td>
<td>-1.95</td>
<td>-5.62</td>
<td>-4.84</td>
<td>-2.65</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.07</td>
<td>-0.06</td>
<td>-0.17</td>
<td>-0.02</td>
<td>-0.85</td>
</tr>
<tr>
<td>1.3B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retraining</td>
<td>-0.35</td>
<td>-0.15</td>
<td>-1.52</td>
<td>-0.05</td>
<td>+1.15</td>
</tr>
<tr>
<td>Finetuning</td>
<td>+1.15</td>
<td>+0.13</td>
<td>-0.70</td>
<td>+0.05</td>
<td>+1.47</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-0.16</td>
<td>-0.14</td>
<td>-1.03</td>
<td>-0.03</td>
<td>-10.35</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>+0.22</td>
<td>-0.16</td>
<td>-5.13</td>
<td>-0.84</td>
<td>-2.28</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.22</td>
<td>-0.03</td>
<td>-0.41</td>
<td>-0.01</td>
<td>-0.27</td>
</tr>
<tr>
<td>2.7B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.4 Experimental Results

In this section, we evaluate all unlearning baselines on four datasets respectively and further highlight the overall performances of each baseline on efficacy, fidelity, and efficiency.

To improve the interpretability of the results, we utilize color coding\(^\text{12}\) to emphasize on distinctive values in Table 13, 14, 15. Due to space constraints, the results of the Lambada dataset (Table ??) can be found in the Appendix.

\(^{12}\)Blue cells stand for that corresponding methods are superior to retraining in the current settings while gray cells mark catastrophic unlearning outcomes, i.e. the accuracy drops significantly (> 10%) or the exposure metric increases instead of decreasing (∆EXP > 0).
### Table 15: Unlearning Results on MathQA Dataset

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unlearning</th>
<th>128 samples</th>
<th>256 samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NEO</td>
<td>OPT</td>
</tr>
<tr>
<td></td>
<td>ΔACC</td>
<td>ΔEXP</td>
<td>ΔACC</td>
</tr>
<tr>
<td>Retraining</td>
<td>+0.40</td>
<td>-0.03</td>
<td>-0.31</td>
</tr>
<tr>
<td>Finetuning</td>
<td>-0.40</td>
<td>+0.31</td>
<td>-0.61</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-1.90</td>
<td>-0.23</td>
<td>+0.59</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-0.56</td>
<td>-1.13</td>
<td>+0.63</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>+0.11</td>
<td>-0.03</td>
<td>+0.83</td>
</tr>
<tr>
<td>Retraining</td>
<td>+1.94</td>
<td>-0.02</td>
<td>+0.47</td>
</tr>
<tr>
<td>Finetuning</td>
<td>+1.25</td>
<td>+0.32</td>
<td>+0.07</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-1.31</td>
<td>-0.11</td>
<td>-1.34</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-0.30</td>
<td>-0.12</td>
<td>-0.36</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>-0.37</td>
<td>-0.02</td>
<td>-0.10</td>
</tr>
<tr>
<td>Retraining</td>
<td>-0.77</td>
<td>-0.04</td>
<td>+0.10</td>
</tr>
<tr>
<td>Finetuning</td>
<td>+1.13</td>
<td>+0.33</td>
<td>+0.20</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>-1.58</td>
<td>-0.08</td>
<td>+0.07</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>-3.49</td>
<td>-0.71</td>
<td>-0.50</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>+0.23</td>
<td>-0.00</td>
<td>-0.16</td>
</tr>
</tbody>
</table>

#### 4.4.1 Efficacy Evaluation.

With retraining as the benchmark, we observe that Fisher Removal can effectively reduce exposure values. For example, in Table 14, Fisher Removal achieves a stronger efficacy guarantee than retraining in all the scenarios, as indicated by blue cells. Fisher Forgetting is less aggressive in terms of erasing the samples, which is consistent with its design.

However, gradient ascent cannot provide an efficacy guarantee as robust as Fisher Removal. In Table 13, the removal effects of gradient ascent are apparently insufficient for GPT-NEO models compared with Fisher Removal. Meanwhile, finetuning frequently causes the samples that should be removed to be more exposed to the adversary, especially in Table 15. It shows that finetuning, which is an intuitively plausible unlearning approach, actually does not erase the information from the samples steadily.

#### 4.4.2 Fidelity Evaluation

In the case of a single unlearning cycle (128 samples), finetuning and Fisher Forgetting are relatively closer to retraining in accuracy. Fisher Removal causes slightly more degradation due to a more aggressive parameter update strategy. However, on closer inspection, gradient ascent causes multiple catastrophic outcomes as shown in Table 13 & 14, suggesting gradient ascent is the least robust regarding retaining the model fidelity.

As for the case of two unlearning cycles (256 samples), finetune and Fisher Forgetting still maintain the accuracy as well as retraining. While Fisher Removal is relatively less utility-preserving compared with Fisher Forgetting. As discussed in Section 4.2.4, Fisher Forgetting has the property of supporting more unlearning cycles. Therefore, we further extend the cycles of unlearning and report the fidelity in Section 4.8.2.
Table 16: Time complexity of each unlearning baseline

<table>
<thead>
<tr>
<th>Method</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retraining</td>
<td>$O(r)$</td>
</tr>
<tr>
<td>Finetuning</td>
<td>$O(r)$</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>$O(t)$</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>$O(tm)$</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>$O(tm)$</td>
</tr>
</tbody>
</table>

4.4.3 Efficiency Evaluation.

We first formally present the time complexity of each method. Let the size of $D^-$ be $t$ and the size of $D^+$ be $r$ respectively. Besides, recall that $m$ is the number of recursions to estimate second-order information in Section 4.2.1.

For simplicity, we assume the time cost for one backpropagation pass is $O(1)$. The time complexity of each method is shown in Table 16. It is noticeable that gradient ascent is the most efficient method among them given $t << r$ can be easily satisfied. Moreover, the complexity of second-order methods is $O(tm)$. In fact, if the dataset is small such that $tm > r$, second-order methods are likely to be expensive. In contrast, when the dataset is large (which is more realistic for LLM applications), second-order methods can be relatively more efficient.

Runtime Comparison. In addition to the time complexity, we also show the runtime of GPT-NEO-125M on various datasets in Figure 8. Note that ARC ($\sim 2k$) and Lambada ($\sim 5k$) datasets are small, while Piqa ($\sim 16k$) and MathQA ($\sim 30k$) datasets are relatively larger.

In general, the results are consistent with previous analysis. Although second-order methods are almost as expensive as retraining on ARC and Lambada datasets, their runtime is close to finetuning on the MathQA dataset.

4.4.4 Overall Evaluation.

To understand the performances more comprehensively, we aggregate the results on individual datasets by: 1) calculating the rank of each method in each scenario\textsuperscript{13}, and 2) reporting the mean and standard deviation of the ranks for each method. Their overall rankings on different dimensions are shown in Table 17. Note that their performances on the two target LLMs are similar. Therefore, we conclude the behaviors of each unlearning approach as follows:

- **Retraining** achieves the optimal balance between efficacy and fidelity as the benchmark for unlearning. However, it is the least efficient baseline as expected.
- **Finetuning** cannot provide any efficacy guarantee, therefore it is not suitable for the purpose of unlearning.
- **Gradient Ascent** is unstable with high variations in both efficacy and fidelity.

\textsuperscript{13}The range is within 1 to 5, with the best method ranked 1st.
Table 17: Overall Rankings of all Unlearning Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>NEO</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Efficacy(↓)</td>
<td>Fidelity(↓)</td>
</tr>
<tr>
<td>Retraining</td>
<td>2.54+-0.91</td>
<td>2.21+-1.08</td>
</tr>
<tr>
<td>Finetuning</td>
<td>4.75+-0.52</td>
<td>2.00+-0.96</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>2.29+-1.06</td>
<td>4.17+-2.69</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>1.33+-0.47</td>
<td>4.08+-1.08</td>
</tr>
<tr>
<td>Fisher Forgetting</td>
<td>3.71+-0.61</td>
<td>2.79+-1.12</td>
</tr>
</tbody>
</table>

Figure 8: The runtime of each unlearning method on GPT-NEO-125M.

For example, its fidelity rank on the GPT-NEO models yields a standard deviation of 2.69. However, it can still serve as a baseline due to its extreme efficiency.

- **Fisher Removal** demonstrates the strongest efficacy by outperforming both retraining and gradient ascent. However, its fidelity score is compromised because of the aggressive removal effect.

- **Fisher Forgetting** is on contrary to fisher removal. It reduces the strength of erase in exchange for more robust accuracy on the main task.

### 4.5 Unlearning for Unintended Memorization

The phenomenon where LLMs tend to memorize certain training samples is called *Memorization.* Carlini et al. [121] extracted hundreds of GPT-2 training samples, including sensitive information such as address, email, phone, etc. The memorization can be troublesome if LLMs are trained on highly sensitive domain data, e.g. medical. Previous work studied the memorization of canaries and unlearned canaries from the training set of an LSTM model [19]. However, the canaries can be viewed as outliers compared with the distribution of the original training set, which makes the removal easier.
To evaluate the effectiveness of the aforementioned unlearning methods against general memorization, we conduct two case studies on LLMs trained on a real-world medical dataset and an email dataset.

### 4.5.1 Datasets

**Medical Transcription (MT):** MT dataset\(^{14}\) contains sample medical transcriptions for various medical specialties, including surgery, consultation, etc. There are 5k long medical transcriptions in total. We further split each transcription into shorter segments such that their lengths are below the input limit.

**Enron Email:** This dataset contains approximately 500k emails generated by employees of the Enron Corporation\(^{15}\). It was obtained by the Federal Energy Regulatory Commission during its investigation of Enron’s collapse. There exist sensitive entities such as phone numbers and passwords in those emails.

### 4.5.2 Training and Inference

**Training.** For each dataset mentioned above, we first randomly select a subset of 100k samples and continue to train a pre-trained GPT-NEO-125M/OPT-125M on it for an extra 15 epochs.

**Inference.** We randomly sample a batch of 128 texts containing more than 200 tokens from each dataset for the purpose of inference. After applying an unlearning method to the trained LLM once, we perform inference on the model using the prefixes of 10 tokens. Top-k [122] sampling is applied and each prefix is repeatedly inferred 10 times.

### 4.5.3 Results

Table 18 displays how effective unlearning methods are at reducing memorization. Since the memorized samples are part of the real training set instead of canaries, the complete forgetting of them is more challenging. Although retraining can guarantee the complete removal, it takes 10X longer time than other methods. Among other unlearning baselines, Fisher Removal is the most effective, while Fisher Forgetting and gradient ascent are relatively weaker. Their performances are consistent with their efficacy scores in Table 17.

### 4.5.4 Qualitative analyses

To understand the reason behind indelible memorizations, we manually inspect the training samples in the MT dataset. As shown in Table ??, these training samples have highly similar substrings. It has been proven that the rate at which LLMs regenerate training sequences is superlinearly related to a sequence’s count in the training set \([123]\). In order to handle these strong memorizations, data deduplication methods

---

\(^{14}\)https://www.kaggle.com/datasets/tboyle10/medicaltranscriptions

\(^{15}\)https://www.kaggle.com/datasets/wcukierski/enron-email-dataset
### Table 18: Unlearning on MT and Enron Email Dataset

<table>
<thead>
<tr>
<th>Methods</th>
<th>NEO</th>
<th>MT</th>
<th>OPT</th>
<th>Enron</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of Mem</td>
<td>Runtime (s)</td>
<td># of Mem</td>
<td>Runtime (s)</td>
<td># of Mem</td>
</tr>
<tr>
<td>No Unlearning</td>
<td>16</td>
<td>5.69 × 10^3</td>
<td>15</td>
<td>5.81 × 10^3</td>
<td>20</td>
</tr>
<tr>
<td>Retraining</td>
<td>0</td>
<td>3.79 × 10^3</td>
<td>15</td>
<td>3.87 × 10^3</td>
<td>20</td>
</tr>
<tr>
<td>Finetuning</td>
<td>8</td>
<td>4.00 × 10^3</td>
<td>7</td>
<td>4.00 × 10^3</td>
<td>9</td>
</tr>
<tr>
<td>Gradient Ascent</td>
<td>4</td>
<td>1.57 × 10^3</td>
<td>4</td>
<td>1.56 × 10^3</td>
<td>5</td>
</tr>
<tr>
<td>Fisher Removal</td>
<td>8</td>
<td>1.97 × 10^3</td>
<td>7</td>
<td>1.56 × 10^3</td>
<td>8</td>
</tr>
</tbody>
</table>

should be involved before the training stage. However, data deduplication falls in the category of pre-processing not unlearning.

### 4.6 DP-SGD vs. Unlearning

![Figure 9: Privacy-utility trade-offs of DP-SGD and unlearning approaches on (a) Lambda and (b) Piqa Dataset. The model under evaluation is OPT-125M.](image)

Differential privacy (DP) guarantees a bound of how much an individual sample can influence the output of a specific function. In the context of deep learning, DP-SGD [124] is usually applied during the training phase. We aim to study to what extent a DP-SGD-trained LLM can protect the training set as well as compare DP-SGD against unlearning.

#### 4.6.1 DP-SGD

We follow the standard definition in [124]. A randomized mechanism \( \mathcal{M} : \mathcal{D} \rightarrow \mathcal{R} \) with domain \( \mathcal{D} \) and range \( \mathcal{R} \) satisfies \( (\epsilon, \delta) \)-differential privacy if any two adjacent inputs \( d, d' \in \mathcal{D} \) for any subset of outputs \( S \subseteq \mathcal{R} \) it holds that

\[
P[\mathcal{M}(d) \in S] \leq e^\epsilon P[\mathcal{M}(d') \in S] + \delta
\]

To investigate whether DP-SGD offers guaranteed privacy in practice, we audit DP-SGD under varying privacy budget \( \epsilon \) values. Specifically, we observe the exposure scores of the 128-sample subset by querying the DP-SGD trained models. Finally, the privacy-utility trade-off is obtained.
4.6.2 Privacy-Utility Trade-off

Likewise, we apply the previous unlearning methods to remove the same 128 samples from the model trained without DP-SGD and report the final exposure scores. By varying the coefficients (in Table 11), we can generate a privacy-utility trade-off for each method. To this end, the trade-offs of DP-SGD as well as the unlearning approaches are summarized in Figure 9.

First, DP-SGD does not guarantee an equally optimal/suboptimal trade-off on different datasets. Although DP almost dominates other methods on the Lambada dataset, its behavior is not robust on the Piqa dataset. Second, DP-SGD introduces additional computational costs into the trivial training process, making the development of LLMs more expensive. Therefore, DP-SGD may serve as a general solution to protect training samples, but it cannot play the same role of unlearning.

4.7 Related Work

4.7.1 Cost of LLMs

The capacities of LLMs are usually accompanied by tremendous GPU hours. Recently, Touvron et al. [125] revealed that training a LLaMA-7B model requires 82432 hours on Nvidia A100 80GB GPU, not to mention larger models such as OPT-175B [21]. Given the enormous amount of resources required for a single training session, erasing target data by training from scratch is impractical and economically prohibitive for LLMs. For example, retraining a model like OPT-175B would entail not only substantial computational power but also significant time and energy costs, translating into substantial financial burdens.

For service providers, efficient unlearning methods maybe a practical solution to significantly reduce computational overhead while still observing the data regulations.

4.7.2 Machine Unlearning

Cao et. al [126] firstly introduced machine unlearning for statistical query learning. They proposed a layer of summations between the learning algorithm and the training data. The learning algorithm relied only on these summations, each of which was the sum of some transformations of the training data samples. The summations were computed during the training phase and used to update the model during the unlearning phase. To forget a specific data sample, the approach simply updated a small number of summations that depended on that sample and then updated the model by resuming the learning algorithm for a few more iterations.

Certified data removal [17] was later proposed, which provided a theoretical guarantee for removal from all convex models without the need to retrain them from scratch. The key method involved using Newton update to modify model parameters and reduce the influence of deleted data points. This process was complemented by masking any residual information with random perturbations, ensuring that removed data cannot be inferred. The method was shown to be practical in various settings, providing
a balance between data removal efficacy and model performance.

As for more general removal that is compatible with deep neural networks, Bourtoule et. al [120] proposed a sharding approach by diving the training data into multiple disjoint shards and training a model on each shard. When the request for unlearning arrived, only the constituent model associated with the specific data shard needed to be retrained. While effectively reducing the computational overhead of unlearning in machine learning models without significantly impacting the accuracy for simpler learning tasks, it introduces several challenges while working with LLMs. Assuming that each shard adequately represents the overall data distribution is not realistic. Moreover, training multiple models on large shards while tuning the hyperparameter for each constituent model also demands significant computational resources. This involves not just the processing power but also memory requirements, as each constituent model needs to be stored, trained, and optimized separately before integration.

More recently, the unlearning of LLMs has attracted increasing attention. Yao et al. [127] applied gradient ascent to unlearn undesirable behaviors in LLMs by aligning the model using negative examples. Wang et al. [128] presented a novel approach that focused on aligning the knowledge gap between models trained on different datasets, suggesting that unlearned models should treat the deleted data as unseen data. However, acquiring high-quality data from the same distribution can be particularly challenging, especially when the availability of such data is limited. In addition, we aim to generally remove the effects of unlearning subsets, rather than focusing on aligning LLMs with human preferences or knowledge gaps.

Chen and Yang [15] introduced an efficient unlearning framework designed to update LLMs effectively without the need for full retraining. This was achieved through the integration of lightweight unlearning layers into transformers, which were specifically trained on a set of data designated for forgetting, enabling it to effectively discard specific knowledge. Elden et al. [14] proposed to replace sensitive tokens of Harry Potter-related text with generic counterparts and finetune the LLMs on the modified samples. Nevertheless, the replacement process is non-trivial and does not easily generalize to other domains. Jang et al. [13] showed that applying gradient descent directly can make LLMs forget about target samples, while no guarantee is provided for model utility.

The aforementioned work has only explored the unlearning task with first-order information, which is reasonable due to its extreme efficiency. However, most of them either require data engineering [14,127,128] or specific architectural design [15], which limits the generalizability. We explore novel unlearning approaches for LLMs relying on second-order information. We show that Hessian can be estimated with affordable computational cost and our methods demonstrate superior robustness with respect to gradient ascent.
4.8 Discussions

4.8.1 Interpreting Unlearning via Weight Distributions

Recall that the distribution of the model parameters after an optimal unlearning algorithm should be indistinguishable from a model that has never seen the unlearning subset (eq. 4.1.1). Specifically, the KL quantity $KL(P(S_1(\theta, D)))||P(S_2(\theta, D^+)))$ should be small (or close to zero in ideal setting). To better illustrate the effects of unlearning, we visualize the weight of the last layer in GPT-NEO 125M after performing different unlearning algorithms in Figure 10.

First, it is noticeable that the weight distribution after applying the gradient ascent deviates far from that of retraining, leading to a large KL divergence of 1176792. However, our second-order-based methods achieve unlearning without pushing the weight distribution too far from retraining. The quantitative KL divergence values by Fisher Removal and Fisher Forgetting are merely 425 and 4174 respectively, which are consistent with the histograms.

4.8.2 Extended Unlearning Cycles

We have discussed the property of Fisher Forgetting in Section 4.2.4, here we compare the performance of each baseline when unlearning cycles are further extended. The GPT-NEO-125M performance curves after consecutive unlearning cycles are displayed in Figure 11. As indicated by the theoretical analysis, Fisher Forgetting maintains the model utility robustly on both the Lambada and Piqa datasets. However, it is not guaranteed that other methods will retain a similar level of utility. Specifically, we observe that gradient ascent causes model performance to drop rapidly as the unlearning subset grows in both scenarios.

4.8.3 Onion Effect

Carlini et al. [129] revealed and analyzed an Onion Effect of memorization: removing the “layer” of outlier points that are most vulnerable to a privacy attack exposes a
new layer of previously safe points to the same attack. In the context of machine unlearning, we explore how the removal of certain data samples impacts the rest of the training data.

We conduct a control experiment on the ARC dataset using GPT-NEO-125M. Specifically, we first obtain the exposure score of each training sample by querying the model before unlearning. Second, samples with exposure scores higher than 6 are removed. Finally, we retrain the model using the modified training dataset and recalculate the exposure scores. As shown in Figure 12, some initially “safe” samples become more exposed after the most exposed samples are removed, which validates the existence of the onion effect.

The results imply that machine unlearning may contradict the goal in the case of eliminating the most vulnerable samples. However, in general, in cases where we assume the unlearning subset is randomly sampled, no significant shift of exposure distributions is observed after unlearning.

4.9 Limitations & Future Work

4.9.1 Limitations

Calculating Hessian is Relatively Costly We have shown that the time complexity of estimating Hessian can be reduced to the product between the size of the unlearning subset and the number of recursions. However, it is still relatively costly compared to the calculation of first-order information. For LLMs, extra GPU space may be required to store the Hessian matrices. Both the time complexity and space complexity could be improved by optimizing the current estimation formula.

Erasing Unintended Memorization is Challenging Although previous work showed that canaries inserted into the training set can be easily erased from an LSTM model [19], it should be considered a special case of memorization since canaries are outliers in the regular training set. Our evaluation of general memorization reveals that even Fisher Removal cannot guarantee complete forgetting in one shot. However, repeatedly applying the unlearning algorithm may compromise the model’s utility.
Figure 12: Onion effect on GPT-NEO-125M. In the ARC dataset, we set the cutoff threshold as 6 and remove all the samples with higher exposure scores. After retraining the model, some left samples have their exposure scores increase past the threshold.

Therefore, it requires further investigation such as data duplication [123] to address indelible memorization.

4.9.2 Future Work

Towards Better Privacy-Utility Trade-off

If we consider that retraining represents the optimal trade-off between privacy and utility, Fisher Removal is biased toward the side of privacy, while Fisher Forgetting favors the side of utility. It still remains challenging to design an unlearning algorithm that strongly guarantees both properties. Future work can focus on improving the privacy-utility trade-off of unlearning. One possible direction is combining different unlearning strategies such as our Fisher Removal and Fisher Forgetting.

Robustness against Extended Unlearning Cycles

In practice, an unlearning algorithm should preserve the utility of the LLM even after multiple unlearning procedures. Otherwise, the service provider may still resort to retraining frequently due to the loss of utility, which contradicts the purpose of unlearning. In our evaluation, only Fisher Forgetting possesses this desired robustness. However, this property is imperative for any unlearning algorithm to be useful in real-world applications. Future work should focus on developing and refining unlearning methods that can reliably preserve the utility of LLMs across multiple unlearning cycles.

Unlearning Larger LLMs & Better Evaluation Metrics

Due to the hardware constraints, we have mainly performed experiments on GPT-NEO/OPT models up to 2.7B. Since our unlearning methods are model-agnostic, we encourage future work to
explore the unlearning process of larger LLMs such as LLama2 7B and Falcon 7B. In addition, we largely follow the previous work to measure how much an LLM still remembers the unlearning subset by exposure score. However, the setup of subspace and the process of iterating through all the samples can be time-consuming. Future work might investigate more efficient/elegant metrics to audit the outcome of unlearning.
References


