# Practical and Efficient Model Extraction of Sentiment Analysis APIs

Weibin Wu*, Jianping Zhang†, Victor Junqiu Wei‡, Xixian Chen§, Zibin Zheng*, Irwin King†, Michael R. Lyu†
*School of Software Engineering, Sun Yat-sen University
†Department of Computer Science and Engineering, The Chinese University of Hong Kong
‡Department of Computing, The Hong Kong Polytechnic University
§Tencent
{wuwb36, zhzibin}@mail.sysu.edu.cn, {jpzhang, king, lyu}@cse.cuhk.edu.hk
wjqjsnj@gmail.com, cxxnju@hotmail.com

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00054

***Abstract*—Despite their stunning performance, developing deep learning models from scratch is a formidable task. Therefore, it popularizes Machine-Learning-as-a-Service (MLaaS), where general users can access the trained models of MLaaS providers via Application Programming Interfaces (APIs) on a pay-per-query basis. Unfortunately, the success of MLaaS is under threat from model extraction attacks, where attackers intend to extract a local model of equivalent functionality to the target MLaaS model. However, existing studies on model extraction of text analytics APIs frequently assume adversaries have strong knowledge about the victim model, like its architecture and parameters, which hardly holds in practice. Besides, since the attacker's and the victim's training data can be considerably discrepant, it is non-trivial to perform efficient model extraction. In this paper, to advance the understanding of such attacks, we propose a framework, PEEP, for practical and efficient model extraction of sentiment analysis APIs with only query access. Specifically, PEEP features a learning-based scheme, which employs out-of-domain public corpora and a novel query strategy to construct proxy training data for model extraction. Besides, PEEP introduces a greedy search algorithm to settle an appropriate architecture for the extracted model. We conducted extensive experiments with two victim models across three datasets and two real-life commercial sentiment analysis APIs. Experimental results corroborate that PEEP can consistently outperform the state-of-the-art baselines in terms of effectiveness and efficiency.**

***Index Terms*—model extraction, sentiment analysis APIs, active learning, architecture search**

## I. Introduction

Recent years have witnessed the fantastic accomplishment of deep neural networks (DNNs) in tackling a growing array of challenging tasks [1]–[3]. It leads to an exploding demand for the deployment of deep learning models in products to embrace the technological advance [4]. Machine-Learning-as-a-Service (MLaaS) thus gains ever-increasing prevalence [5], [6]. On the one hand, MLaaS providers make their advanced trained models partially accessible by offering prediction APIs to users. These vendors can monetize their service on a pay-per-query basis. On the other hand, general subscribers with limited expertise in deep learning can simply upload data of interest via MLaaS APIs and directly obtain the final analysis results on these data, eliminating the burden of building an expensive model from scratch.

The confidentiality of the trained models underneath MLaaS APIs plays a central part in the prosperity of MLaaS [7], [8]. First of all, like traditional software, deep learning models represent the intangible assets of the developers. Developing deep learning models generally requires a tremendous investment of time, money, and human efforts, originating from collecting and annotating enormous training data to engineering and tuning model architectures and parameters. It is thus in the best interest of MLaaS providers to hold the copyright of their MLaaS model to profit from its commercial value and maintain a competitive edge. Besides, in security and privacy-sensitive applications, exposing white-box access to the MLaaS models can facilitate evasion attacks [9]–[11] and incur a disastrous privacy breach [12].

Unfortunately, model extraction can pose a severe threat to the confidentiality of MLaaS models. Model extraction, also known as functionality stealing, is a kind of attack where hackers endeavor to extract a functionality-equivalent local model from the victim API [13]–[15]. More concretely, functional equivalence means that the output of the extracted model can match that of the target MLaaS model in the original problem domain of the victim [16], [17]. As such, legitimate users cannot differentiate the extracted model from the victim one, when only given query access to both of them [18], [19].

However, it is challenging to mount model extraction attacks against real-world MLaaS APIs, especially those for natural language processing (NLP) tasks. First, since MLaaS providers encapsulate the trained deep learning models into black-box APIs, attackers usually have limited knowledge about the target model, such as its architecture, parameters, and training data. Second, the admissible action of the adversaries is heavily restricted. They should behave like legitimate users. Therefore, only query access to the target MLaaS model is permitted. Third, query efficiency is also a fundamental criterion for a successful model extraction attack, since excessive queries may incur a prohibitive expense and make model extraction more detectable. Fourth, compared to models tailored for computer vision tasks, the discrete input nature of NLP models further complicates model extraction, since attackers cannot arbitrarily manipulate their query samples to

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00054

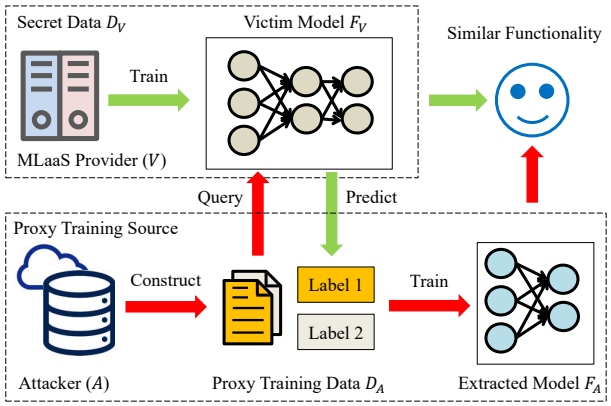


Fig. 1: Illustration of our model extraction scheme. In each iteration, we first adaptively select unlabeled samples from the proxy training source to query the victim model. We then employ the resultant proxy training data to learn an extracted model.

extract useful information about the target model [20].

Existing studies on model extraction of NLP APIs have two primary deficiencies. First, some suffer from limited practicality. They assume that adversaries know the architecture of the target MLaaS model and sometimes even the parameters of the pre-trained encoder employed by the victim. Therefore, adversaries can exploit the same architecture and pre-trained encoder as the victim [16], [20], [21]. As such, the model extraction task is reduced to extracting one feedforward layer's parameters of the victim. However, since MLaaS providers scarcely disclose the internals of their models, such assumptions hardly hold in practice. Second, query efficiency is not yet satisfactory. Most existing proposals apply random sampling to select query data for training the extracted model [20], [22], [23]. Some borrow the sampling strategy from the active learning literature [24], [25] to improve query efficiency [17]. Nevertheless, these sampling approaches are geared to standard model training, where the query data source belongs to the problem domain of interest. In contrast, attackers cannot obtain the proprietary training data of the victim during model extraction. Consequently, the training data of the attackers and victims can be substantially disparate, which renders these sampling strategies inefficient for model extraction.

Therefore, in this work, we attempt to further expose the realistic threat of such attacks. Since sentiment analysis APIs have a broad spectrum of applications in practice [26], [27] and are popular offerings on diverse MLaaS platforms, we focus on extracting sentiment analysis APIs in this paper. To this end, we propose PEEP, a learning-based framework for model extraction of sentiment analysis APIs.

Figure 1 illustrates the overall pipeline of PEEP, which is an iterative query-and-training procedure. Specifically, since the victim model's training data (the **secret data**) are not accessible, we first gather out-of-domain but publicly available corpora as the **proxy training source**. Then in the query phase of each iteration, with a novel sampling strategy that pursues an exploration-exploitation trade-off, we adaptively select instances from the proxy training source to query the victim model. The resultant input-output pairs constitute the **proxy training data**. In the subsequent training phase, we exploit the proxy training data to learn an **extracted model**, which aims to gradually extract the target model's knowledge. Besides, the architecture of the extracted model is settled by a greedy search algorithm, where we greedily search for a structure that possesses an appropriate capacity to efficiently imitate the functionality of the target model.

We conducted extensive experiments to evaluate the effectiveness and efficiency of PEEP. Concretely, we first launched model extraction attacks in simulated black-box settings. We targeted two top-performing victim models across three datasets, which are tailored to analyze the sentiment of text data from different domains. Experimental results show that PEEP can markedly surpass the state-of-the-art baselines in terms of effectiveness and efficiency. More importantly, we validated PEEP with two real-life commercial sentiment analysis APIs. Under a budget of only $1, we can extract local replicas that obtain over 87.9% agreement with the corresponding commercial APIs.

In summary, we would like to underscore the following contributions of this work:

- We propose a framework called PEEP for **P**ractical and **E**fficient model **E**xtraction of sentiment analysis A**PI**s. PEEP features employing out-of-domain public corpora and a novel query strategy to construct proxy training data for model extraction. Moreover, it applies a greedy search algorithm to settle an appropriate architecture for the extracted model.
- We performed extensive experiments to assess the effectiveness and efficiency of PEEP with two cutting-edge victim models across three datasets. PEEP can consistently outperform the state-of-the-art benchmarks in terms of effectiveness and efficiency.
- We investigated the effect of possible defenses on the performance of PEEP. Experimental results confirm that only returning label predictions is not an effective defense against PEEP.
- We further validated PEEP with two real-life commercial sentiment analysis APIs. The results corroborate that our model extraction attack can translate well to real-world scenarios.

## II. Problem Description

In this section, we define the task that we endeavor to tackle in this paper, namely, extracting the underlying model from real-world sentiment analysis APIs. As depicted in Figure 1, model extraction involves two parties: a victim $V$ and an attacker $A$. We elaborate on the victim's system model and the attacker's threat model in Section II-A and Section II-B, respectively.

**Algorithm 1** Model Extraction of Sentiment Analysis APIs.

**Require:** Proxy training source $\mathcal{X}_A$, sampling strategy $\pi$, and candidate structure pool $\{F_{Am}\}$

**Require:** Iteration number $N$, query budget $B$, and seed sample size $k_0$

1: $F_A \leftarrow$ GREEDYARCHITECTURESEARCH$(\{F_{Am}\})$ ▷ settle the architecture of $F_A$ via greedy architecture search

2: $S^{(0)} \leftarrow k_0$ instances randomly sampled from $\mathcal{X}_A$

3: $D_A^{(0)} \leftarrow \{(\mathbf{x}, F_V(\mathbf{x})) : \mathbf{x} \in S^{(0)}\}$ ▷ query the victim model to label the selected samples

4: $k \leftarrow \dfrac{B - k_0}{N - 1}$

5: **for** $i = 1$ to $N - 1$ **do** ▷ iterative query-and-training routine

6: $F_A^{(i-1)} \leftarrow$ SUPERVISEDTRAINING$(D_A^{(i-1)})$ ▷ train the extracted model on $D_A^{(i-1)}$

7: $\mathcal{X}_A \leftarrow \mathcal{X}_A - S^{(i-1)}$ ▷ remove already queried samples

8: $\bar{D}_A^{(i)} \leftarrow \{(\mathbf{x}, F_A^{(i-1)}(\mathbf{x})) : \mathbf{x} \in \mathcal{X}_A\}$ ▷ use the extracted model to label the remaining samples

9: $S^{(i)} \leftarrow \pi(\bar{D}_A^{(i)}, k)$ ▷ select $k$ samples with strategy $\pi$

10: $D_A^{(i)} \leftarrow D_A^{(i-1)} \cup \{(\mathbf{x}, F_V(\mathbf{x})) : \mathbf{x} \in S^{(i)}\}$

11: **end for**

12: **return** $F_A \leftarrow$ SUPERVISEDTRAINING$(D_A^{(N-1)})$

### A. The Victim's System Model

The victim $V$ is the provider of the target sentiment analysis model $F_V$. To develop $F_V$, $V$ first needs to gather sufficient problem-specific data $\mathcal{X}_V$. Then $V$ resorts to human experts to label these data, which results in a secret dataset $D_V := \{(\mathbf{x}_i, y_i)\}$. Here $y_i$ denotes the ground-truth label of $\mathbf{x}_i$. $D_V$ is further divided into disjoint training and test partitions, denoted as $D_V^{train}$ and $D_V^{test}$, respectively. The victim finally trains different models on $D_V^{train}$ and selects the one that attains the best performance on $D_V^{test}$ for deployment.

### B. The Attacker's Threat Model

*1) The Knowledge of the Attacker:* We assume that attackers are aware of the target task of the victim model $F_V$ (i.e., detecting sentiments in text), its expected input $\mathbf{x}$ (i.e., sentences within a certain length limit), and the semantic meaning of the returned output $y$ (i.e., the sentiment label for the provided sentence). Without offering such necessary information, it may even disable legitimate use, which violates the motivation for publishing an MLaaS API.

Nonetheless, attackers do not have access to the secret dataset $D_V$ of the victim. In general, these data may be proprietary and confidential, for example, patients' health records. Moreover, attackers know nothing about the internals of $F_V$, including its architectures, parameters, and input feature representations.

*2) The Admissible Action of the Attacker:* We assume that the attacker $A$ interacts with the target MLaaS API like a legitimate user. Therefore, attackers are only endowed with query access to the victim model. Specifically, attackers can only upload valid inputs (i.e., sentences) to $F_V$, and acquire predictions on them (i.e., sentiment labels). According to the predefined interface of the target API, victims may offer extra information about their label predictions on a given input, e.g., the estimated probability of an input sentence belonging to each sentiment class. However, to make our attack strategy more general, we propose to exploit the minimum information that is indispensable for legitimate use in this paper. That is, only sentiment label predictions are available to the adversary.

*3) The Goal of the Attacker:* The goal of the attacker is to extract a functionality-equivalent local model $F_A$ such that given the same input in the problem domain of the victim ($\mathcal{X}_V$), the label predictions of the extracted and victim models can match [16], [17]. Since the secret test set ($D_V^{test}$) is a representative collection of data from $\mathcal{X}_V$, we employ $D_V^{test}$ to estimate the functional similarity between $F_V$ and $F_A$ (i.e., their agreement $S_{VA}(D_V^{test})$):

$$S_{VA}(D_V^{test}) = \frac{1}{|D_V^{test}|} \sum_{\mathbf{x} \in D_V^{test}} \mathbf{1}(F_A(\mathbf{x}) = F_V(\mathbf{x})). \quad (1)$$

Here $\mathbf{1}(\cdot)$ is the indicator function, and $|D_V^{test}|$ returns the cardinality of the set $D_V^{test}$. Higher agreement values signify that $F_V$ and $F_A$ are more functionally similar.

## III. METHODOLOGY

In this section, we detail our framework, PEEP, for model extraction of sentiment analysis APIs. Algorithm 1 describes the complete procedure of PEEP. The core part is an iterative query-and-training routine to gradually refine the extracted model, which alternates the query and training phases in each turn. We explain the central query-and-training process in Section III-A. In the query phase, we need to construct a proxy training dataset $D_A$, which we detail in Section III-B. Then in Section III-C, we elucidate the training phase.

### A. Query-and-Training Routine

The query-and-training routine of PEEP corresponds to Lines 5–11 of Algorithm 1, which is inspired by the active learning methodology [24], [28]. In summary,

1) We first randomly select some unlabeled seed instances $S^{(0)}$ from the proxy training source $\mathcal{X}_A$. We then query the target API to label these seed samples, which results in a proxy training dataset $D_A^{(0)}$.
2) In the $i$-th iteration ($i = 1, ..., N-1$), we train an extracted model $F_A^{(i-1)}$ with $D_A^{(i-1)}$.
3) We employ $F_A^{(i-1)}$ to annotate all instances in $\mathcal{X}_A$, excluding those already labeled by the victim model $F_V$. It results in a candidate dataset $\bar{D}_A^{(i)}$.
4) We apply a sample strategy $\pi$ to select a subset of samples from the candidate dataset $\bar{D}_A^{(i)}$, which constitutes the unlabeled proxy training data $S^{(i)}$.
5) We query the victim model $F_V$ to label all samples in $S^{(i)}$. We add the resultant annotated instances to the previous proxy training dataset $D_A^{(i-1)}$.

6) We repeat Steps 2–5 to gradually refine the extracted model.

### B. Proxy Training Data Construction

Like the victim, attackers need to build a training dataset (the proxy training dataset $D_A$) before they can train an extracted model. Since annotating is performed by querying the victim model, the remaining tasks are gathering adequate unlabeled samples (i.e., determining the proxy training source $\mathcal{X}_A$) and selecting informative instances therein for query efficiency (i.e., devising the sampling strategy $\pi$). We present our solutions as follows.

*1) Proxy Training Source $\mathcal{X}_A$:* In practice, victims scarcely release the problem domain data $\mathcal{X}_V$ that they curated for model development, since these data may be confidential, e.g., patients' health records. Therefore, we propose to employ publicly available corpora as the proxy training source $\mathcal{X}_A$, which can be of different domains than $\mathcal{X}_V$.

*2) Sampling Strategy $\pi$:* We introduce a novel sampling strategy $\pi$ to select instances in $\mathcal{X}_A$, which are the most informative for extracting models from the target API. We then prioritize querying them to construct $D_A$. It facilitates reducing the number of queries we need before achieving the desired agreement.

Our idea is to quantify the informativeness of each candidate example in $\mathcal{X}_A$. As such, we can conveniently incorporate and combine different metrics that can boost the query efficiency of model extraction in our informativeness formula. Based on the feedback information from the current extracted model, we focus on two metrics in this work: uncertainty and diversity, which we detail as follows.

Uncertainty champions samples that are close to the decision boundary of the current extracted model, which represents an exploitation strategy. Such instances are the ones on which the extracted model is most uncertain about its decisions. Therefore, learning with these samples can quickly mold the decision boundary of the extracted model. We adopt the following entropy-based formula [24] to calculate the uncertainty of a sample $\mathbf{x} \in \bar{D}_A^{(i)}$:

$$R^{uncertainty}(\mathbf{x}) = -\hat{\mathbf{p}}_A(\mathbf{x}) \cdot \log \hat{\mathbf{p}}_A(\mathbf{x}). \tag{2}$$

Here $\hat{\mathbf{p}}_A(\mathbf{x})$ represents the output probability vector of the extracted model when given $\mathbf{x}$.

However, since the victim's problem domain data $\mathcal{X}_V$ and the attacker's proxy training source $\mathcal{X}_A$ may be materially discrepant, there are numerous out-of-domain data in $\mathcal{X}_A$ from the perspective of the victim model $F_V$. Without being trained on these out-of-domain data, $F_V$ may be uncertain about its predictions on these samples, even though their ground-truth class probabilities can be high [29]. As a result, solely banking on uncertainty sampling can over-exploit out-of-domain data and impede the extraction of the victim's knowledge in the original problem domain.

Therefore, to complement uncertainty sampling, we introduce another criterion when calculating a sample's informativeness: diversity. Diversity advocates samples that are distinct from the ones we have queried so far, which accounts for the principle of exploration. We introduce diversity to prevent the pure exploitation of a specific category of samples, e.g., uncertain examples that may be out-of-domain for $F_V$. We characterize the diversity of an instance $\mathbf{x} \in \bar{D}_A^{(i)}$ as:

$$R^{diversity}(\mathbf{x}) = \frac{1}{|D_A^{(i-1)}|} \sum_{(\mathbf{x}_m, y_m) \in D_A^{(i-1)}} ||L_A^{(k)}(\mathbf{x}) - L_A^{(k)}(\mathbf{x}_m)||_2. \tag{3}$$

Here $L_A^{(k)}(\mathbf{x})$ is the feature representation of $\mathbf{x}$ output by the $k$-th layer of $F_A$. Compared to uncertainty sampling, diversity sampling excels at mining data that may be closer to the problem domain of $F_V$. As a result, it can help the extracted model to imitate the victim's behavior in the original problem domain.

We combine these two metrics to balance exploitation and exploration during sampling, leading to the informativeness formula of a sample $\mathbf{x} \in \bar{D}_A^{(i)}$ that we employ:

$$R(\mathbf{x}) = R^{uncertainty} + \lambda \cdot R^{diversity}. \tag{4}$$

Here $\lambda$ is a weight parameter to balance the contribution of both terms. Therefore, in each iteration, the sampling strategy $\pi(\bar{D}_A^{(i)}, k)$ will return $k$ samples with the top informativeness values from $\bar{D}_A^{(i)}$.

### C. Extracted Model Training

After obtaining the proxy training data, we learn an extracted model on them to approximate the decision boundary of the victim model. Specifically, we need to determine the architecture and training algorithm of $F_A$, since these design choices of the victim model are unknown to the attacker. We describe our remedies as follows.

*1) Greedy Architecture Search:* We settle the structure of $F_A$ via a greedy architecture search algorithm, which is motivated by the studies on knowledge distillation [30]. We expect that, for a successful model extraction attack, $F_A$ and $F_V$ can be of different model types, e.g., XLNet [31] vs. BERT (Bidirectional Encoder Representations from Transformers) [32], as long as their capacities are comparable with respect to the target task of $F_V$.

Therefore, we first determine the model types of $F_A$ (e.g., XLNet and BERT), which are suitable for the task of $F_V$ as per domain knowledge. Then we engineer a candidate structure pool $\{F_{Am}\}$, which consists of $M$ candidate structures. Besides, the candidate architectures of each model type possess increasing complexity. As such, we simplify the architecture search task into choosing a structure from $\{F_{Am}\}$ for $F_A$, which owns the most appropriate capacity for quickly duplicating the functionality of $F_V$.

To efficiently address this problem, we devise a greedy architecture search algorithm, which is detailed in Algorithm 2. In short,

1) We first prepare a proxy validation set $D_A^{val}$, which is used to compute and compare the performance of different candidate structures on model extraction.

**Algorithm 2** Greedy Architecture Search.

**Require:** Proxy training source $\mathcal{X}_A$, sampling strategy $\pi$, and candidate structure pool $\{F_{Am}\}$
**Require:** Iteration number $C$, proxy validation set $D_A^{val}$, and sample size $d$
$S^{(0)} \leftarrow d$ instances randomly sampled from $\mathcal{X}_A$
Initialize $S_{Am}^{(0)} \leftarrow S^{(0)}$, for all $m \leq M$
Initialize $D_{Am}^{(0)} \leftarrow \{(\mathbf{x}, F_V(\mathbf{x})) : \mathbf{x} \in S^{(0)}\}$, for all $m \leq M$
Initialize $\mathcal{X}_{Am} \leftarrow \mathcal{X}_A$, for all $m \leq M$
$h \leftarrow \dfrac{M-1}{C}$ ▷ the number of candidates discarded in each iteration
**for** $i = 1$ to $C$ **do**
  **for** each $m$ in the index set of $\{F_{Am}\}$ **do**
    $F_{Am}^{(i-1)} \leftarrow$ SUPERVISEDTRAINING$(D_{Am}^{(i-1)})$ ▷ train the candidate model on $D_{Am}^{(i-1)}$
    $S_{VAm}^{(i-1)} \leftarrow \frac{1}{|D_A^{val}|} \sum_{\mathbf{x} \in D_A^{val}} \mathbf{1}(F_{Am}^{(i-1)}(\mathbf{x}) = F_V(\mathbf{x}))$ ▷ calculate the agreement
    **if** $i < C$ **then**
      $\mathcal{X}_{Am} \leftarrow \mathcal{X}_{Am} - S_{Am}^{(i-1)}$
      $\bar{D}_{Am}^{(i)} \leftarrow \{(\mathbf{x}, F_{Am}^{(i-1)}(\mathbf{x})) : \mathbf{x} \in \mathcal{X}_{Am}\}$
      $S_{Am}^{(i)} \leftarrow \pi(\bar{D}_{Am}^{(i)}, d)$ ▷ select samples with strategy $\pi$
      $D_{Am}^{(i)} \leftarrow D_{Am}^{(i-1)} \cup \{(\mathbf{x}, F_V(\mathbf{x})) : \mathbf{x} \in S_{Am}^{(i)}\}$
    **end if**
  **end for**
  $\{F_{Am}\} \leftarrow$ REMOVE$(\{F_{Am}\}, h)$ ▷ discard $h$ candidate models with the lowest $S_{VAm}$ values
**end for**
**return** $F_A \leftarrow \{F_{Am}\}$

2) We separately train each candidate model structure from $\{F_{Am}\}$ for one iteration. The training process of a candidate structure follows the query-and-training routine introduced in Section III-A.
3) We compute the agreement between the victim and the candidate models on the proxy validation set $D_A^{val}$. We will discard $h$ candidate models from $\{F_{Am}\}$ that attain the lowest agreement values.
4) We iterate Steps 2–3 until there is only one structure left in $\{F_{Am}\}$. We will employ it as the selected architecture of $F_A$.

*2) Training Algorithm:* Although some particulars of the training protocol may vary across different types of models, we consistently adopt the cross-entropy loss as the training loss function, which is a standard of practice for categorization problems [33]. For model extraction, the extracted model is trained to mimic the behaviors of the victim model $F_V$. Therefore, unlike regular model training, we modify the standard cross-entropy loss as:

$$\mathcal{L}_{CE}(\hat{\mathbf{p}}_V, \hat{\mathbf{p}}_A) = -\hat{\mathbf{p}}_V \cdot \log \hat{\mathbf{p}}_A. \tag{5}$$

Here $\hat{\mathbf{p}}_V$ and $\hat{\mathbf{p}}_A$ are the probability vectors predicted by the victim and the extracted models, respectively. Since we assume that the victim only returns final label predictions in this paper, we exploit the one-hot encoding of its label prediction $y_V$ to construct $\hat{\mathbf{p}}_V$. As such, different from standard model training, we treat the prediction of the victim model $F_V$ as the ground-truth label.

## IV. Experimental Setup

### A. Evaluation Metrics

As mentioned in Section II-B, we employ agreement to assess the extent to which model extraction attacks are successful. For PEEP, there are mainly three building blocks: the design of the proxy training source, the sampling strategy to select instances to be labeled, and the architecture search algorithm to settle the structure of the extracted model. Therefore, to evaluate these building blocks, we adopt the following metrics based on agreement:

**Effectiveness.** Effectiveness aims to evaluate the performance of a design of the proxy training source. We define the effectiveness of a proxy training source as the obtained agreement when running PEEP with the proxy training source, under the maximum query budget considered in this paper (i.e., 1K).

**Efficiency.** Efficiency aims to evaluate the performance of a sampling strategy. Attackers apply a sampling strategy to select the most informative query samples to maximize the obtained agreement under a preset query budget. Therefore, we define the efficiency of a sampling strategy as the obtained average agreement across different query budgets (i.e., 0.2–1K), when running PEEP with the sampling strategy. We set the range of the query budgets to 0.2–1K so that we can achieve results comparable to those reported in the state-of-the-art baselines [17], [20].

**Search time and the corresponding obtained agreement.** Search time and the corresponding obtained agreement aim to evaluate the performance of an architecture search algorithm. We define the search time of an architecture search algorithm to be its execution time to settle a structure for the extracted model. The corresponding obtained agreement is the obtained agreement on the proxy validation set, when running PEEP with the structure settled by the architecture search algorithm.

### B. Research Questions

We structure the evaluation of PEEP into four research questions as follows.

**RQ1: What is the performance of our design of the proxy training source and greedy architecture search algorithm?** In this RQ, we endeavor to evaluate the performance of our design of the proxy training source and greedy architecture search algorithm. The crucial hurdle for model extraction is that attackers cannot access the secret data and internal

| Dataset | Classes | Training Samples | Test Samples | Domain |
|---|---|---|---|---|
| MR | 2 | 8530 | 1066 | Movie |
| SST2 | 2 | 67349 | 872 | Movie |
| GitHub | 3 | 4985 | 2137 | GitHub |

TABLE I: Statistics of the adopted datasets.

| Victim Model | Accuracy |
|---|---|
| MR-BERT | 84.3 |
| MR-XLNet | 87.1 |
| SST2-BERT | 92.4 |
| SST2-XLNet | 93.6 |
| GitHub-BERT | 92.0 |
| GitHub-XLNet | 92.0 |

TABLE II: Accuracy (%) of the victim model on the corresponding secret test set.

particulars of the victim model. Therefore, a practical model extraction attack should first overcome this challenge.

**RQ2: What is the performance of the proposed sampling strategy?** Attackers may also attempt to minimize the number of queries to reduce the extraction cost and evade detection [13]. Therefore, in this RQ, we assess the performance of our sampling strategy by calculating its efficiency.

**RQ3: What is the effect of possible defenses on the performance of PEEP?** In this RQ, we aim to gain further insight into the robustness of PEEP. Therefore, we investigate whether PEEP is still effective against possible defenses. It can shed light on the development of future defenses.

**RQ4: Can PEEP be applied to extract models from real-life commercial sentiment analysis APIs?** In this RQ, we seek to further validate the practical applicability of PEEP. By evaluating the performance of PEEP with real-life commercial sentiment analysis APIs, we can answer whether our framework and findings can generalize well to real-world situations.

### C. Victim Models

In this work, we target both simulated black-box sentiment analysis APIs and real-world commercial sentiment analysis APIs. We conduct all our experiments employing a server with one NVIDIA GeForce RTX 2080 Ti GPU of 11GB memory.

*1) Simulated Black-Box Sentiment Analysis APIs:* In the simulated black-box setup, we employ three characteristic public datasets: Movie Review (MR) [34], Stanford Sentiment Treebank (SST2) [35], and GitHub pull request and commit comments (GitHub) [36]. Table I summarizes their statistics. They are all tailored for the task of sentiment analysis but come from different application domains. Specifically, MR and SST2 consist of movie reviews, while GitHub comprises sentences from the GitHub pull request and commit comments.

We exploit a prevailing paradigm in modern NLP systems to build target models [20]. The target models' structures are a composition of a pre-trained language model (e.g., BERT) and a task-specific output layer [32]. We consider two state-of-the-art pre-trained and publicly available language models: BERT-Base [37] and XLNet-Base [31]. Therefore, for each secret dataset, we build two victim models, resulting in a total of six victim models in the simulated black-box setting.

Specifically, we denote these victim models in the form of secret dataset-model architecture. We employ the publicly available trained parameters[1] for these victim models except SST2-XLNet, GitHub-BERT, and GitHub-XLNet. For SST2-XLNet, GitHub-BERT, and GitHub-XLNet, we fine-tune the pre-trained language model (BERT-Base[2] or XLNet-Base[3]) on the corresponding dataset. During fine-tuning, we apply an AdamW optimizer [38] with default training configurations to minimize the cross-entropy loss [33]. We fine-tune the models for three epochs. Table II shows the accuracy of these victim models on the corresponding secret test set.

*2) Real-World Commercial Sentiment Analysis APIs:* As for real-world commercial sentiment analysis APIs, we target two APIs provided by Google Cloud[4] and Microsoft Azure[5], respectively. They are representative APIs for sentiment analysis and possess cutting-edge performance. Both APIs take raw text as input and outputs the corresponding sentiment label for users. Note that both APIs also return confidence scores for their decisions. Nevertheless, as in the simulated black-box setting, we only exploit the top-1 label predictions for the generality of our work.

### D. Implementation of Our Framework

*1) Proxy Training Source $\mathcal{X}_A$:* For all of our experiments, we randomly select and fix 30K samples from the publicly available WikiText-103 corpus (Wiki) [39] as the proxy training source $\mathcal{X}_A$. Therefore, our proxy training source $\mathcal{X}_A$ and the secret datasets of the victims come from different domains.

*2) Extracted Model Architecture:* In our candidate structure pool $\{F_{Am}\}$ for the extracted model, we consider three model types: Bidirectional Long-Short Term Memory (BiLSTM) [40], XLNet [31], and BERT [32], since such models are adept at sentiment classification [41]. For each model type, we engineer different candidate structures by changing their capacities, e.g., increasing the number and size of hidden layers.

Table III enumerates the candidate structure pool $\{F_{Am}\}$ we adopt during the greedy architecture search. Particularly, for XLNet-based models, we employ three publicly available pre-trained models with varying complexity: XLNet-Tiny[6], XLNet-Base[3], and XLNet-Large[7]. For BERT-based models, we also employ three publicly available pre-trained models with varying complexity: BERT-Tiny[8], BERT-Base[2], and BERT-Large[9]. We configure and randomly initialize the last fully connected layer of these pre-trained models based on the

[1]https://huggingface.co/textattack
[2]https://huggingface.co/bert-base-cased
[3]https://huggingface.co/xlnet-base-cased
[4]https://cloud.google.com/natural-language/
[5]https://azure.microsoft.com/en-us/services/cognitive-services/language-service/
[6]https://huggingface.co/sshleifer/tiny-xlnet-base-cased
[7]https://huggingface.co/xlnet-large-cased
[8]https://huggingface.co/prajjwal1/bert-tiny
[9]https://huggingface.co/bert-large-cased

| Structure | Hidden Size | Layers |
|---|---|---|
| BiLSTM-1 | 150 | 1 |
| BiLSTM-2 | 150 | 2 |
| BiLSTM-3 | 150 | 3 |
| BERT-Tiny | 128 | 2 |
| BERT-Base | 768 | 12 |
| BERT-Large | 1024 | 24 |
| XLNet-Tiny | 128 | 2 |
| XLNet-Base | 768 | 12 |
| XLNet-Large | 1024 | 24 |

TABLE III: Candidate model structures considered during our greedy architecture search.

| Victim Model | Use Random Text | Our Method |
|---|---|---|
| MR-BERT | 56.4 | **87.1** |
| MR-XLNet | 59.3 | **90.1** |
| SST2-BERT | 60.8 | **88.1** |
| SST2-XLNet | 65.8 | **86.1** |
| GitHub-BERT | 44.9 | **76.3** |
| GitHub-XLNet | 45.2 | **75.8** |

TABLE IV: Agreement (%) between the extracted and the victim models when employing different proxy training sources. We denote each victim model in the form of secret dataset-model architecture.

class number of the target model. For BiLSTM-based models, we randomly initialize all the model parameters.

For simplicity, we determine the final structure of $F_A$ for each secret dataset by conducting greedy architecture search with MR-BERT, SST2-BERT, and GitHub-BERT as victim models, respectively. We then employ the same architecture of $F_A$ for all victims of the same secret dataset without model-specific architecture search. We set the iteration number $C = 4$ and the sample size $d = 100$ for our greedy architecture search algorithm. The proxy validation set $D_A^{val}$ contains 30K samples randomly selected from the WikiText-103 corpus. We note that $D_A^{val}$ and the proxy training source of the attacker do not overlap.

*3) Training Configurations:* We experimentally set $k_0 = 0.1B$ and $N = 10$ without fine-tuning for simplicity. We apply an AdamW optimizer [38] with default training configurations and a batch size of 32. We train the extracted model for three epochs. The weight parameter $\lambda$ is settled by a grid search. We employ the logit layer of the extracted model to compute the feature representations of samples in Eq. (3).

### *E. Benchmark Strategy*

We compare our framework with state-of-the-art baselines [17], [20] in terms of the design of the proxy training source $\mathcal{X}_A$ and the sampling strategy $\pi$. Specifically, Krishna et al. [20] proposed to employ random text as the proxy training source $\mathcal{X}_A$. They will randomly sample words from the WikiText-103 vocabulary [39] to construct query sentences. The sampling strategy of Pal et al. [17] is to prioritize querying instances with higher uncertainty values (Eq. (2)), which corresponds to a pure exploitation scheme during sampling.

| Candidate Structure | Step | | | | Stop Step |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| BiLSTM-1 | 54.9 | 56.5 | 62.1 | 61.7 | 1 |
| BiLSTM-2 | 55.8 | 63.2 | 63.7 | 63.2 | 3 |
| BiLSTM-3 | 55.8 | 58.8 | 61.4 | 61.4 | 2 |
| BERT-Tiny | 54.7 | 57.6 | 59.2 | 62.8 | 1 |
| BERT-Base | 66.1 | 74.1 | 76.0 | 75.1 | 4 |
| BERT-Large | 65.7 | 74.6 | 76.0 | 74.8 | 4 |
| XLNet-Tiny | 55.8 | 57.8 | 58.3 | 58.4 | 2 |
| XLNet-Base | 67.7 | 69.3 | 75.6 | 76.8 | - |
| XLNet-Large | 55.7 | 62.8 | 64.4 | 65.8 | 3 |

TABLE V: Agreement (%) between the extracted model and the victim model (MR-BERT) on the proxy validation set after each query-and-training iteration during the grid/greedy architecture search procedure. "Stop Step" indicates the step after which the candidate structures are removed from $\{F_{Am}\}$ during our greedy architecture search process.

To validate our greedy architecture search algorithm, we consider a naive grid search algorithm as the baseline. The baseline will first completely train all the candidate model structures. Then it selects the top performer among them as the final extracted model. Specifically, we directly adopt each candidate model structure in Table III as the structure of $F_A$ and then run PEEP to conduct model extraction. We note that the grid search baseline is capable of discovering the best architecture in the considered candidate structure pool. For a fair comparison, we employ the grid search algorithm to settle the structure of $F_A$ under the same setting of our greedy search algorithm. Specifically, we set the query-and-training iteration number $C = 4$ and the sample size in each iteration $d = 100$. The grid search algorithm also performs model extraction against the same victim models as our greedy search algorithm.

## V. Experimental Results

### *A. RQ1: What is the performance of our design of the proxy training source and greedy architecture search algorithm?*

We first evaluate our design of the proxy training source $\mathcal{X}_A$. To this end, we compare the effectiveness of different proxy training sources. Specifically, given a proxy training source, we run PEEP under a fixed query budget of 1K. We then compare the obtained agreement of the models extracted with different proxy training sources.

Table IV reports the model extraction results when employing different proxy training sources. We denote each victim model in the form of secret dataset-model architecture. "Use Random Text" corresponds to the technique of Krishna et al. [20]. We can consistently achieve higher agreement in **all cases**. Notably, in terms of the obtained average effectiveness across all victims, our construction of the proxy training source $\mathcal{X}_A$ exceeds the benchmark design [20] by over 28.5%. The advance confirms that PEEP is significantly more effective than the state-of-the-art baseline [20], in terms of the design of the proxy training source.

| | | MR | SST2 | GitHub |
|---|---|---|---|---|
| Settled | Grid | XLNet-Base | XLNet-Base | XLNet-Base |
| Structure | Greedy | XLNet-Base | XLNet-Base | XLNet-Base |
| Search | Grid | 3.4 | 3.4 | 2.6 |
| Time | Greedy | **2.8** | **2.9** | **2.2** |

TABLE VI: The final structure settled by different architecture search algorithms and their search time (in hours).

We then validate our greedy architecture search algorithm. Table V demonstrates the grid/greedy architecture search procedure when the victim model is MR-BERT. Table VI summarizes the final structure settled by different architecture search algorithms and their search time. The results show that the final structures settled by the grid search and our greedy search algorithms are the same. It confirms that our greedy search algorithm can discover the architecture that can obtain the highest agreement on the proxy validation set among all the candidate ones. Besides, in terms of the search time, it costs about 3.1 hours on average to settle the best structure with the grid search algorithm. In contrast, our greedy search scheme only takes about 2.6 hours on average, which is about 1.2 times faster than the naive grid search algorithm.

### *B. RQ2: What is the performance of the proposed sampling strategy?*

In this RQ, we compare the performance of different sampling strategies. To this end, we run PEEP with different sampling strategies: random sampling, diversity sampling, uncertainty sampling [17], and our sampling strategy. Specifically, random sampling will randomly select samples to query. Diversity sampling prioritizes querying samples with higher diversity values (Eq. (3)), while uncertainty sampling prioritizes querying samples with greater uncertainty values (Eq. (2)). For each victim model, we vary the query budget $B$ from 0.2K to 1K and compare the obtained agreement of the models extracted with different sampling strategies.

Table VII presents the model extraction results when applying different sampling strategies. We can consistently achieve higher agreement under the same query budget in **all cases**. Remarkably, in terms of the obtained average efficiency across all victims, our sampling technique outperforms the baseline [17] by over 2.1%. The performance gain verifies that we are considerably more efficient than the state-of-the-art benchmark [17], in terms of the sampling strategy.

To examine the significance of the performance difference between PEEP and the state-of-the-art baseline [17], we repeat the experimental evaluation as in Table VII and conduct two-tailed $t$-tests, which are widely used to determine if there is a statistically significant difference between the means of two groups [42]. The null hypothesis is that the mean agreements obtained by PEEP and the baseline are equal. Experimental results show that $p$-values are less than 0.05 for all tests. Therefore, the null hypothesis is rejected, which confirms that there is a significant difference between the performance of PEEP and the state-of-the-art baseline.

In terms of the achieved average agreement across different query budgets, we make the following observations when we zoom in on each secret dataset: First, our method is the most efficient sampling strategy across all secret datasets. We outshine the state-of-the-art baseline [17] by around 3.5%, 1.5%, and 1.3% on the MR, SST2, and GitHub datasets, respectively. Second, random sampling is the least efficient sampling strategy over all datasets. It verifies that both the uncertainty and diversity metrics in our informativeness definition (Eq. (4)) can boost query efficiency. Therefore, combining both of them, as we do, can achieve the best performance. Third, comparing different secret datasets, extracting models tailored for the GitHub dataset seems to be the most arduous task for all sampling methods.

To investigate the reason for these sampling methods' discrepant performances on different datasets, we display some example sentences from each dataset in Table VIII. We can see that the samples from the GitHub dataset are sentences from GitHub pull request and commit comments, which are colloquial and often contain code snippets. By contrast, the sentences from the MR and SST2 datasets are movie reviews, which are more formal and comprise natural language text. Recall that we employ samples from the Wiki dataset as the proxy training source, which also consists of natural language sentences. Since the sentences in the GitHub dataset are considerably different from those in the other datasets, the performance of model extraction on the GitHub dataset is relatively lower than that under the other situations.

### *C. RQ3: What is the effect of possible defenses on the performance of PEEP?*

In this RQ, we examine the effect of possible defenses on the performance of PEEP. We note that for the generality of our work, we assume that only the top-1 label predictions of the victim are available in previous experiments. Actually, popular MLaaS providers, such as Google Cloud and Microsoft Azure, also offer confidence scores for their label predictions. Therefore, reducing the returned information can be a possible defense deployed by the victim to prevent model extraction [13]. It motivates us to further investigate whether such a strategy can be an effective defense against PEEP.

To this end, we measure the performance variations of PEEP induced by this defense methodology. Specifically, when the victim also outputs a full probability vector, we can directly feed it to the training loss function (Eq. (5)) to guide the learning of the extracted model. We then compare the performance of the resultant extracted models with that in our previous experiments.

Table IX shows the effect of employing different outputs of the victim model on PEEP, under a fixed query budget $B = 1$K. We can see that only employing the label predictions actually helps to improve the performance of PEEP across all victims.

Different from knowledge distillation [30], in the setting of model extraction, the attacker's proxy training dataset and the victim's secret dataset are from different domains. Therefore,

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 74.9 | 75.5 | 78.2 | 80.2 | 80.5 |
| Uncertainty | 80.4 | 80.7 | 79.9 | 83.9 | 83.7 |
| Diversity | 74.1 | 78.1 | 78.3 | 80.0 | 82.5 |
| Our Method | **81.7** | **84.8** | **83.9** | **87.6** | **87.1** |

(a) MR-BERT

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 53.9 | 72.3 | 78.9 | 83.9 | 84.7 |
| Uncertainty | 71.9 | 81.9 | 82.1 | 85.9 | 86.9 |
| Diversity | 70.2 | 74.6 | 81.9 | 84.4 | 85.2 |
| Our Method | **78.4** | **85.4** | **84.9** | **88.4** | **90.1** |

(b) MR-XLNet

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 56.1 | 68.3 | 81.4 | 82.4 | 82.2 |
| Uncertainty | 80.7 | 82.6 | 84.2 | 87.1 | 87.3 |
| Diversity | 69.0 | 70.7 | 86.5 | 85.8 | 85.7 |
| Our Method | **82.1** | **84.2** | **88.0** | **87.8** | **88.1** |

(c) SST2-BERT

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 56.1 | 68.3 | 82.0 | 82.8 | 82.3 |
| Uncertainty | 79.7 | 81.2 | 83.2 | 85.7 | 85.2 |
| Diversity | 61.9 | 70.7 | 83.5 | 83.8 | 83.1 |
| Our Method | **81.0** | **82.5** | **86.1** | **86.4** | **86.1** |

(d) SST2-XLNet

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 58.7 | 68.3 | 68.9 | 68.2 | 70.1 |
| Uncertainty | 63.4 | 76.7 | 75.0 | 76.1 | 75.5 |
| Diversity | 62.3 | 72.5 | 70.5 | 72.9 | 74.9 |
| Our Method | **66.0** | **77.8** | **76.3** | **76.9** | **76.3** |

(e) GitHub-BERT

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 62.5 | 63.5 | 64.1 | 66.8 | 67.1 |
| Uncertainty | 71.5 | 72.4 | 73.2 | 73.5 | 73.0 |
| Diversity | 67.0 | 68.9 | 71.9 | 71.7 | 70.9 |
| Our Method | **72.9** | **73.8** | **73.7** | **74.0** | **75.8** |

(f) GitHub-XLNet

TABLE VII: Agreement (%) between the extracted and the victim models when applying different sampling strategies.

| Dataset | Example Sentence |
|---|---|
| Wiki | Lieberstein and Jennifer Celotta were named the series showrunners for the fifth season. |
| MR | a masterful film from a master filmmaker , unique in its deceptive grimness , compelling in its fatalist worldview . |
| SST2 | is pretty damned funny . |
| GitHub | Well, it would be safer to use ‘assoc = Hash.new { \|h,k\| h[k] = [] }‘ |

TABLE VIII: Example sentences from different datasets.

| Victim Model | Probability | Top-1 Label |
|---|---|---|
| MR-BERT | 80.3 | 87.1 |
| SST2-BERT | 83.7 | 88.1 |
| GitHub-BERT | 73.7 | 76.3 |

TABLE IX: Agreement (%) obtained by our framework when employing different outputs from the victim model.

during model extraction, the information provided by the victim model's full probability output may be noisier than that provided by its label prediction. As a result, employing the victim model's full probability output does not achieve better model extraction results than employing only the label prediction. Besides, when normally training a model, developers only need data annotated with labels. Therefore, as confirmed by the experimental results, it is not an effective defense against PEEP to reduce the returned information of the victim model (i.e., only returning the label prediction).

### *D. RQ4: Can PEEP be applied to extract models from real-life commercial sentiment analysis APIs?*

To further validate the practical applicability of PEEP, we perform model extraction against commercial sentiment analysis APIs offered by Google Cloud[4] and Microsoft Azure[5]. Both APIs charge users no more than $1 per 1K queries, when the total number of queries is below 1M. The price will then decrease as the number of queries increases.

Unlike the simulated black-box setting, we cannot access the internal test set $D_V^{test}$ of the victim model to evaluate the performance of our method. Therefore, we first conduct experiments to determine a proxy test set, which should be similar to the original test set of the victim. Specifically, we evaluate the accuracy of the API on several datasets and select the one on which the victim can obtain the highest accuracy as the proxy test set, since we expect that the victim model can perform well on the original test set. After comparisons, we choose to employ the test set of Yelp Reviews (Yelp) [43] as the proxy test set, which is distinct from our proxy training source.

The accuracy of Google Cloud API and Microsoft Azure API on the proxy test set is 94.8% and 90.2%, respectively. After settling the proxy test set, we perform model extraction as in the simulated black-box setting. The extracted model structure selected by our greedy search algorithm is XLNet-Base for both APIs.

We first compare the effectiveness of different proxy training sources. Specifically, we run PEEP under a fixed query budget of 1K with different proxy training sources to conduct model extraction. Table X shows the results. Again, we can consistently achieve higher agreement with both APIs. In

| Victim API | Use Random Text | Our Method |
|---|---|---|
| Google | 62.2 | **90.1** |
| Microsoft | 54.7 | **87.9** |

TABLE X: Agreement (%) between the extracted model and the commercial sentiment analysis API when employing different proxy training sources.

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 76.9 | 82.9 | 84.4 | 86.9 | 87.0 |
| Uncertainty | 81.2 | 85.7 | 86.7 | 88.3 | 87.2 |
| Diversity | 76.5 | 84.5 | 83.2 | 86.8 | 86.8 |
| Our Method | **87.8** | **88.0** | **89.0** | **88.6** | **90.1** |

(a) Google

| Sampling Strategy | 0.2K | 0.4K | 0.6K | 0.8K | 1K |
|---|---|---|---|---|---|
| Random | 83.0 | 84.0 | 85.4 | 85.6 | 85.7 |
| Uncertainty | 84.6 | 87.3 | 87.4 | 87.2 | 87.0 |
| Diversity | 85.8 | 87.5 | 86.7 | 86.4 | 86.6 |
| Our Method | **86.1** | **88.1** | **87.6** | **87.8** | **87.9** |

(b) Microsoft

TABLE XI: Agreement (%) between the extracted model and the commercial sentiment analysis API when applying different sampling strategies.

terms of the obtained average agreement across both APIs, our construction of the proxy training source $\mathcal{X}_A$ outshines the baseline design [20] by about 30.6%. The advance confirms that PEEP is significantly more effective than the state-of-the-art benchmark [20], in terms of the design of the proxy training source.

We then evaluate the efficiency of different query strategies. The results are presented in Table XI. We can consistently outperform the state-of-the-art baselines [17] in **all cases** by over 1.8% on average. Notably, under a relatively restricted budget (about $1 for 1K queries) for both APIs, we can succeed in extracting models with a remarkable performance (over 87.9% agreement). Therefore, our model extraction framework can translate well to real-world scenarios.

## VI. Threats to Validity

The first threat to validity is regarding the typicality of the considered NLP task and datasets. Sentiment analysis is a critical task for many NLP applications, such as social media monitoring and product analysis [44]. It is also a popular offering on diverse MLaaS platforms. Therefore, we believe that extracting models for sentiment analysis can serve as a representative task to validate the performance of our attack. Actually, sentiment analysis is a characteristic text classification task, and our framework only cares about the input-output behaviors of the target model. Therefore, PEEP should generalize well to extract models for other NLP tasks. As for the datasets, we have curated well-recognized datasets for sentiment analysis in different application domains [34]–[36]. Since we evaluate PEEP on all of them, our evaluation results should be generalizable to datasets from other application domains. In our future work, we also plan to extend PEEP to other NLP tasks and datasets.

The second threat to validity comes from the availability of our proxy training source. As observed in Section V-A, when there is a large domain gap between the attacker's proxy training source and the victim's secret data (e.g., random text vs. movie reviews), it will be difficult to perform model extraction. If we employ public corpora that come from similar domains with the victim's secret data as the proxy training source, the performance of model extraction can improve. However, although for some victim models (e.g., models for analyzing movie reviews), similar sentiment analysis corpora are publicly available, there are some victim models whose application domain data are not publicly available and low-resource, e.g., conversations between doctors and patients. Therefore, in this work, to ensure the availability of the proxy training source, we do not employ a victim-specific proxy training source that comes from a specific domain like movie reviews. Instead, we exploit general-purpose corpora that are publicly available. Based on our experimental results, although our proxy training source is out-of-domain, our PEEP can still effectively and efficiently extract models from the target APIs.

## VII. Discussion

The implications of our results for both practitioners and researchers are as follows:

(1) We assume a more realistic setting than that in previous studies, where attackers cannot access the particulars of the target NLP software. Therefore, we expose the realistic threat of model extraction to NLP software. Notably, we find that under a budget of only $1, we can extract local replicas delivering over 87.9% agreement with the victim APIs. Besides, we confirm that only returning the label prediction is not an effective defense. We thus call attention to this new security issue.

(2) We endeavor to make PEEP generally applicable to different application domains, which is validated by our experimental evaluations. Therefore, PEEP can be readily applied to assess the vulnerability of NLP software and the corresponding defenses to model extraction.

(3) This paper focuses on providing a strong benchmark for evaluating the vulnerability of NLP software and the corresponding defenses to model extraction in practice. Furthermore, we believe that PEEP can help to develop defenses against model extraction. For example, a potential defense would be applying PEEP to calculate the informativeness of a query sample and then blocking queries of highly informative samples. We leave further exploration to future work.

Particularly, the implications for the software engineering (SE) community are as follows:

(1) Like traditional software, developers also want to protect the copyright of their NLP software. To this end, PEEP can serve as a strong benchmark and help to develop defenses, as mentioned above.

(2) To facilitate the development of software, sentiment analysis software has been frequently used in the SE community to analyze the sentiment of various textual data [2], [26], [45] (e.g., GitHub commit comments, Stack Overflow posts, and Mobile app reviews). Therefore, the studied sentiment analysis software is also of interest to the SE community.

## VIII. Related Work

### A. Model Extraction

Like the code plagiarism issue in software engineering [46]–[48], deep learning software can be reproduced without permission by model extraction, which infringes the original developer's copyright. Model extraction, also called model stealing or functionality stealing, aims to extract a functionality-equivalent local copy of a victim model through only query access [13], [49].

Existing studies on model extraction generally focus on attacking image classifiers [8], [13]–[15], [18], [19]. These works often expect that attackers can freely manipulate query samples. As such, they can craft queries that are close to the victim classifier's decision boundary to extract useful information. However, such strategies cannot transfer to extract NLP APIs due to the discrete input nature of NLP models [20].

With the prevalent deployment of NLP models via MLaaS APIs, model extraction of NLP APIs has attracted growing interest [17], [20], [23]. These studies usually assume that attackers are informed of the architecture [22] and sometimes even the parameters of the pre-trained encoder [16], [20], [21] employed by the victim. Therefore, adversaries can adopt the same structure as the victim during attacks. However, such assumptions hardly hold in practice. In contrast, we explore more realistic settings in this paper, where attackers are not aware of the particulars of the target NLP model, such as its architecture, parameters, and training data.

Besides, existing proposals generally apply random sampling [20], [22], [23] or uncertainty sampling [17] to select query data. However, these sampling strategies overlook the glaring discrepancies between the attacker's proxy training data and the victim's secret data, rendering them inefficient for model extraction. To address this defect, we design a novel sampling strategy to boost the query efficiency of our framework. More crucially, we further validate our approach by applying it to extract the underlying model of two famous commercial sentiment analysis APIs. Therefore, we reveal the risk of such attacks in practice and call attention to this new security issue.

### B. Defenses against Model Extraction

Mainstream defenses against model extraction can be roughly categorized into two camps: detecting malicious queries [8] and prediction poisoning [50]. Most are tailored for protecting image classification models and are non-trivial to be adapted to safeguard NLP models. Nonetheless, we discuss the applicability of their ideas to defend against PEEP.

Detecting malicious queries aims to identify ongoing model extraction attacks, which enables blocking or misleading malicious users to prevent model extraction. Existing approaches of this category usually assume that attackers will modify natural data to synthesize query samples [8], [51], [52], which can extract maximal information from the victim. As such, the query samples from attackers and legitimate users are distinctly different [53], [54]. However, in our attack, we employ public and natural text as the proxy training source, which are similar to the query examples from legitimate users. Therefore, our attack can evade detection by these defenses.

Prediction poisoning works by perturbing the outputs of the victim model to disturb the training of extracted models. Representative proposals generally assume that the attacker will employ the full output probabilities [50], [55] or translation results [23] of the victim to train their extracted model. However, in our attack, we only exploit the label predictions from the victim. Therefore, these defense methods are also not effective against our attack.

## IX. Conclusion

In this work, we propose PEEP for practical and efficient model extraction of sentiment analysis APIs with only query access. PEEP is a learning-based framework, which employs public data as the proxy training source, and a novel sampling strategy to improve the query efficiency. Besides, we devise a greedy search algorithm to settle appropriate architectures for the extracted model. Experimental results corroborate the marked superiority of our technique over state-of-the-art benchmarks. We further demonstrate that we can successfully extract models from two famous commercial sentiment analysis APIs. Therefore, we faithfully expose the threat of such attacks in practice and call attention to this new security issue.

In future work, we plan to exploit PEEP to develop effective defenses against model extraction of NLP APIs. We also plan to extend our framework to extract models for other NLP tasks, such as question answering and machine translation. It will help to understand the threat of model extraction to a broader range of NLP software.

## Acknowledgment

The authors would like to thank the anonymous shepherd and reviewers for their detailed and insightful comments and suggestions, which have helped to improve this paper. This work was supported by the National Natural Science Foundation of China (Grant No. 62206318), the NSFC-Guangdong Joint Fund Project (Grant No. U20A6003), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund), and the Hong Kong RGC Research Impact Fund (RIF) with Project No. R5034-18 (CUHK 2410021).

## References

[1] Z. Chen, Y. Cao, H. Yao, X. Lu, X. Peng, H. Mei, and X. Liu, "Emoji-powered sentiment and emotion detection from software developers' communication data," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–48, 2021.

[2] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang, "Sentiment analysis for software engineering: How far can pre-trained transformer models go?" in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 70–80.

[3] K. Chen, Y. Li, Y. Chen, C. Fan, Z. Hu, and W. Yang, "GLIB: Towards automated test oracle for graphically-rich applications," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2021, pp. 1093–1104.

[4] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 750–762.

[5] M. Ribeiro, K. Grolinger, and M. A. Capretz, "MLaaS: Machine learning as a service," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 896–902.

[6] J. Weng, J. Weng, C. Cai, H. Huang, and C. Wang, "Golden grain: Building a secure and decentralized model marketplace for MLaaS," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[7] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta, "Model extraction warning in MLaaS paradigm," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 371–380.

[8] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "PRADA: Protecting against DNN model stealing attacks," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 512–527.

[9] W. Wu, Y. Su, X. Chen, S. Zhao, I. King, M. R. Lyu, and Y.-W. Tai, "Boosting the transferability of adversarial samples via attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1161–1170.

[10] W. Wu, Y. Su, M. R. Lyu, and I. King, "Improving the transferability of adversarial samples with adversarial transformations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9024–9033.

[11] W. Wu, Y. Su, X. Chen, S. Zhao, I. King, M. R. Lyu, and Y.-W. Tai, "Towards global explanations of convolutional neural networks with concept attribution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8652–8661.

[12] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 267–284.

[13] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 601–618.

[14] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt, "Model reconstruction from model explanations," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM, 2019, pp. 1–9.

[15] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," *arXiv preprint arXiv:1811.02054*, 2018.

[16] X. He, L. Lyu, L. Sun, and Q. Xu, "Model extraction and adversarial transferability, your BERT is vulnerable!" in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 2006–2012.

[17] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. Shevade, and V. Ganapathy, "ActiveThief: Model extraction using active learning and unannotated public data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, 2020, pp. 865–872.

[18] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4954–4963.

[19] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1345–1362.

[20] K. Krishna, G. S. Tomar, A. P. Parikh, N. Papernot, and M. Iyyer, "Thieves on Sesame Street! Model extraction of BERT-based APIs," in *International Conference on Learning Representations (ICLR)*, 2020.

[21] S. Zanella-Beguelin, S. Tople, A. Paverd, and B. Köpf, "Grey-box extraction of natural language models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 278–12 286.

[22] N. S. Keskar, B. McCann, C. Xiong, and R. Socher, "The thieves on sesame street are polyglots-extracting multilingual models from monolingual APIs," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6203–6207.

[23] E. Wallace, M. Stern, and D. Song, "Imitation attacks and defenses for black-box machine translation systems," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 5531–5546.

[24] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *SIGIR'94*. Springer, 1994, pp. 3–12.

[25] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[26] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 94–104.

[27] B. Lin, N. Cassee, A. Serebrenik, G. Bavota, N. Novielli, and M. Lanza, "Opinion mining for software development: a systematic literature review," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–41, 2022.

[28] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *International Conference on Learning Representations (ICLR)*, 2018.

[29] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," 2017.

[30] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[31] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[34] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, 2005, pp. 115–124.

[35] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1631–1642.

[36] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile, "Can we use se-specific sentiment analysis tools in a cross-platform setting?" in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 158–168.

[37] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962*, 2019.

[38] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[39] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations (ICLR)*, 2017.

[40] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[41] T. Abdullah and A. Ahmet, "Deep learning in sentiment analysis: A survey of recent architectures," *ACM Computing Surveys (CSUR)*, 2022.

[42] J. H. McDonald, *Handbook of biological statistics*. sparky house publishing Baltimore, MD, 2009, vol. 2.

[43] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2015, pp. 649–657.

[44] Z. Drus and H. Khalid, "Sentiment analysis in social media and its application: Systematic literature review," *Procedia Computer Science*, vol. 161, pp. 707–714, 2019.

[45] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 548–559.

[46] C. Liu, Z. Lin, J.-G. Lou, L. Wen, and D. Zhang, "Can neural clone detection generalize to unseen functionalities$f$," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 617–629.

[47] B. Liu, W. Huo, C. Zhang, W. Li, F. Li, A. Piao, and W. Zou, "$\alpha$diff: Cross-version binary code similarity detection with dnn," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 667–678.

[48] Y.-C. Jhi, X. Wang, X. Jia, S. Zhu, P. Liu, and D. Wu, "Value-based program characterization and its application to software plagiarism detection," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 756–765.

[49] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 2005, pp. 641–647.

[50] T. Orekondy, B. Schiele, and M. Fritz, "Prediction poisoning: Towards defenses against dnn model stealing attacks," in *International Conference on Learning Representations*, 2020.

[51] H. Zheng, Q. Ye, H. Hu, C. Fang, and J. Shi, "BDPL: A boundary differentially private layer against machine learning model extraction attacks," in *European Symposium on Research in Computer Security*. Springer, 2019, pp. 66–83.

[52] Z. Zhang, Y. Chen, and D. Wagner, "SEAT: Similarity encoder by adversarial training for detecting model extraction attack queries," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, 2021, pp. 37–48.

[53] S. Kariyappa and M. K. Qureshi, "Defending against model stealing attacks with adaptive misinformation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 770–778.

[54] B. G. Atli, S. Szyller, M. Juuti, S. Marchal, and N. Asokan, "Extraction of complex dnn models: Real threat or boogeyman?" in *International Workshop on Engineering Dependable and Secure Machine Learning Systems*. Springer, 2020, pp. 42–57.

[55] H. Ma, T. Chen, T.-K. Hu, C. You, X. Xie, and Z. Wang, "Undistillable: Making a nasty teacher that CANNOT teach students," in *International Conference on Learning Representations*, 2021.