Towards Preserving Server-Side Privacy of On-Device Models

Akanksha Atrey[1], Ritwik Sinha[2], Somdeb Sarkhel[2], Saayan Mitra[2], David Arbour[2], Akash V. Maharaj[3], Prashant Shenoy[1]

[1]University of Massachusetts Amherst, [2]Adobe Research, [3]Adobe Inc.

aatrey@cs.umass.edu,{risinha,sarkhel,smitra,arbour,maharaj}@adobe.com,shenoy@cs.umass.edu

ABSTRACT

Machine learning-based predictions are popular in many applications including healthcare, recommender systems and finance. More recently, the development of low-end edge hardware (e.g., Apple's Neural Engine and Intel's Movidius VPU) has provided a path for the proliferation of machine learning on the edge with on-device modeling. Modeling on the device reduces latency and helps maintain the user's privacy. However, on-device modeling can leak private server-side information. In this work, we investigate on-device machine learning models that are used to provide a service and propose novel privacy attacks that can leak sensitive proprietary information of the service provider. We demonstrate that different adversaries can easily exploit such models to maximize their profit and accomplish content theft. Motivated by the need to preserve both client and server privacy, we present preliminary ideas on thwarting such attacks.

CCS CONCEPTS

• Computing methodologies → Machine learning; **Distributed artificial intelligence**; **• Security and privacy → Human and societal aspects of security and privacy**.

KEYWORDS

privacy, machine learning, on-device models, distributed systems, personalization

ACM Reference Format:
Akanksha Atrey[1], Ritwik Sinha[2], Somdeb Sarkhel[2], Saayan Mitra[2], David Arbour[2], Akash V. Maharaj[3], Prashant Shenoy[1]. 2022. Towards Preserving Server-Side Privacy of On-Device Models. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3487553.3524257

1 INTRODUCTION

The growth and ubiquity of Machine Learning (ML) models has changed how we conduct our lives. It affects everything from how we capture and store our images and videos, to what movies we decide to watch or what products we end up buying. This ubiquity coupled with the explosion in the number and types of consumer smart devices (i.e., laptops, mobile phones, smart watches, televisions) implies that ML inference can no longer be conducted in a centralized fashion. This has led to ML models being evaluated in a decentralized manner so that personalization and recommendation decisions can be made closer to where the content is served. For instance, a personalization model that chooses between several marketing offers for a particular customer may take as input the context of the user, and score several offers to choose the best. This inference can be performed on the person's mobile phone.

Performing the inference of the ML model locally, on the end-user's device provides several advantages [2]. First, performing on-device inference helps reduce the latency with which the content is served, latency being a critical factor in user experience. Second, evaluating ML models on user devices helps avoid moving user data to the cloud, providing a layer of privacy protection to the user. Third, on-device models may be processed without having the need to be connected to the cloud, thus can be processed even if the device is offline. Finally, not having to evaluate billions of decisions on the cloud can help reduce costs associated with compute and storage. Unfortunately, when an ML model is on the user's device, it lies outside of the security perimeter provided by the cloud and is open to possible exploitation by an adversary. We explore this aspect of on-device models in this work.

Example. Consider a trained ML model that is deployed on the device for inference. Assume this model is used by a bank's website to provide personalized financial incentives or offers to users. Note that the number and types of offers as well as who receives which offer are proprietary information for the bank. Thus, the bank expects that the offer recommended to the user is the only one they have access to. An adversary or curious user, in control of the device, can query the model to get an offer. With continuous and sufficient amount of querying, the adversary can learn the probability distribution of the potential offers. This is concerning because such information is proprietary and can be published or sold to coupon sites, competing brands, or price aggregators which can affect the client's business. Secondly, since such systems often have a feedback loop, the adversary can poison or bias the model by systematically searching through the feature space.

To this end, our contributions are as follows:

- **C1** We develop a taxonomy of on-device models focusing on models used in distributed services (e.g., web or mobile applications).
- **C2** We propose multiple privacy attacks on on-device models and evaluate their efficacy on a real-world dataset. Our results demonstrate that simple attacks can leak sensitive intellectual property of the service provider.
- **C3** We develop preliminary ideas on how to protect server-side privacy in on-device models.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524257

Table 1: Descriptions of white-box (WB) and black-box (BB) on-device models with the components of the ML model accessible by a user. "Rep" denotes representations.

Model	Feature Space		Output Space	Internals	
Type	All	Model Input	Model Output	Weights	Rep
WB	✓	✓	✓	✓	-
BB	✓	-	-	-	-

2 PRIVACY ATTACKS ON ON-DEVICE MODELS

2.1 Problem Setup

We focus on the multi-class classification task using one-vs-all models which are popular in many industry applications [7]. We describe the threat model and its entities as follows.

Service Provider. We consider a service provider $\mathcal{S}$ which is responsible for providing an arbitrary service to its users. The service provider requires the usage of machine learning-based predictions to provide the best service. We assume $\mathcal{S}$ trains a one-vs-all k-class classification model $M : X \to Y$, where X represents a user's contextual information and Y represents a service provided to the user (e.g., marketing offer). The final output is given by $y = \text{argmax}_{1...k} f_k(x) = M(x)$ where f_k represents a binary classification model of response for class k. This model is deployed on its users' devices. We assume Y and the representations learned by M are proprietary information.

User. We consider user $\mathcal{U}$ who employs the service provided by $\mathcal{S}$. We assume $\mathcal{U}$ keeps a deployed version of M on their local device and employs the service in an honest manner by accepting the service, $M(X_U) = y$, as provided by $\mathcal{S}$.

Adversary. We consider adversary $\mathcal{A}$ who intends to exploit the service provided by $\mathcal{S}$. Similar to $\mathcal{U}$, we assume $\mathcal{A}$ keeps a deployed version of M on their local device. Further, we assume $\mathcal{A}$ is treated like a regular user $\mathcal{U}$ and provided a service y based on their contextual data X_A. We consider $\mathcal{A}$ to be a curious adversary that attempts to learn $M(X'_A) \to Y$ (i.e., proprietary information).

Note, our focus is on privacy rather than security. Thus, we do not consider external security threats such as model or data theft. Instead, we focus on the privacy of the information available to, and business interests of the service provider $\mathcal{S}$.

2.2 Taxonomy of On-Device Models

The model space can be divided into three segments: feature space, prediction space and internals. Each segment can contain numerous features describing the input, output and internal representations. Additionally, we observe the common case where $\mathcal{S}$ collects multiple features from the customer and employs only a subset of them for training M.

We focus on two classes of on-device models, white-box and black-box. In the traditional user privacy setting, a white-box model is transparent where all its parameters are accessible and a black-box model is opaque with its model architecture and parameters hidden. Since on-device models reside on a user's device and have the ability to execute offline, certain aspects such as model input and output are automatically accessible.

In this work, we define white-box models as ones whose feature space and prediction space are visible to the user, yet the representations learned by the model are unknown. This implies that the K models are visible in a one-vs-all k-class model. This is representative of the setting where a model is running on a webpage by being embedded into the webpage's code. In such a setting, the model is accessible via methods such as browser's inspect element. On the other hand, black-box models refer to models where only the total set of features and the model output (i.e., offer) are visible to the user. This is representative of a model embedded into an application's binary interface, common in mobile applications. We describe these models in Table 1.

2.3 Privacy Attacks

In this work, we focus on server-side privacy attacks on on-device models which aim to recover the representations learned by model M. Attacks on on-device models can be divided into two categories, white-box and black-box. We present three attacks: model inversion, white-box random querying and black-box large-scale querying.

2.3.1 *Model Inversion Attack.*

As described in Section 2.2, in the white-box setting, adversary $\mathcal{A}$ has access to the K binary classification models in M. Model inversion attacks are plausible in such a setting for neural network based models. These attacks have historically been employed to learn sensitive attributes in the training data using a trained ML model [4]. In this work, we can use such an attack to exploit each binary classification model and reconstruct the input for each class using backpropagation.

Particularly, we exploit confidence information of these models. Since $f_k(x)$ is a binary classification model, $\mathcal{A}$ can recover the set of inputs by assigning a 100% probability to the class confidence. Thus, for each model f_k, the attack reconstructs input x_k by iteratively transforming a randomly generated input towards the value that maximizes class k. With sufficient amount of iterations, $\mathcal{A}$ can recover all potential $x_k \to y_k$ mappings.

2.3.2 *Random Querying Attack.*

Another white-box attack is randomly querying the known model input to gather all potential classes. Here $\mathcal{A}$ has access to the features used by the model and number of classes. The goal is to iteratively query the features used by the model to recover the k classes. The adversary can directly use M rather than the individual models to conduct the attack.

2.3.3 *Large-Scale Querying Attack.*

The black-box attack is more complex as $\mathcal{A}$ does not have access to the number of classes and set of features employed by the model. Here $\mathcal{A}$ only has access to all the features collected by the service provider. In such a case, similar to the white-box random querying attack, the adversary can randomly query all features at scale.

3 EVALUATION OF SERVER-SIDE PRIVACY LEAKAGE

3.1 Experimental Setup

Data. We use a real-world web experimentation dataset from a major telecommunications retailer for evaluation. This dataset consists of session level data where users are randomly exposed to one of multiple treatments via A/B tests. The treatment identifier

Table 2: Difference between traditional serialized models and ONNX models for random forests (RF) and deep neural networks (DNN).

		Size (KB)	Runtime (s)	Accuracy (%)
RF	Cloud	11300	0.2642	99.24
	ONNX	5895	0.4176	99.24
DNN	Cloud	564	0.2578	98.84
	ONNX	560	0.0989	98.84

along with profile and contextual features are captured at the time of treatment. Any subsequent conversions in the form of sign-ups for offers are then recorded, joined with the features vectors, and treated as positive samples. A lack of conversion is treated as a negative sample. The dataset consists of 30,000 samples each with $\sim$ 900 features ($\sim$ 17000 after processing and one hot encoding).

Models. We train two types of one-vs-all multi-class classification models, random forests and deep neural networks (DNN), particularly a two-layer multilayer perceptron. Here, a model is trained for each class and the final output is the argmax over the classes for the conversion prediction model.

To mimic the on-device setting, we use the Open Neural Network Exchange (ONNX) format [1]. ONNX is an open source format to represent machine learning models and enhance interoperability. Table 2 describes the difference between traditional serialized models versus ONNX models. Transforming trained random forests to the ONNX format reduces the storage size by more than 50% but increases runtime. For DNNs, we see that the size of the models stay around the same but there is a huge improvement in runtime complexity. In both cases, the prediction accuracy is unaffected.

Measures. The goal of the attacks is to recover M's input to output mappings. We evaluate the attack efficacy via the average percentage of prediction space (classes) recovered while learning input to output representations.

3.2 Results

We first identify the runtimes of the attacks described in Section 2.3. Table 3 contains runtimes of recovering the full set of input to output mappings in the DNN. All results are aggregated for 100 users. The model inversion attack performs the fastest; given the simplicity of the binary classification models in M, it is easy to recover input in less than five iterations. Random querying on the model input also performs fairly fast but slower than the model inversion method. However, we note that running large scale querying is much slower. With only 1000 unused (extra) features, large-scale querying was approximately 1045 and 36 times slower than the model inversion and random querying attacks, respectively.

We further evaluate the querying methods in the white-box and black-box setting. Figure 1a demonstrates the effect of differing query sample sizes on the average percentage of prediction space recovered. The adversary is successfully able to recover the DNN's output, that is the potential recommendation content, with a limited number of queries in both the white-box and black-box scenario. Attacking the random forest, however, is around 80% successful with 5000 queries. We hypothesize this is due to random forests being more robust to outliers than DNNs [3], since they are based on decision trees as base learners. The response surface learnt by a random forest is less smooth than that of a DNN and random queries are not bounded to the same distribution as the training data. Thus the DNN ends up revealing more private information whereas the random forest is more robust to such inputs.

Table 3: Runtimes of privacy attacks to recover full set of input to output mappings.

Model Type	Attack	Runtime (s)
White-box	Model Inversion	0.0531
	Random Querying	1.5323
Black-box	Large Scale Querying	55.4894

Although the number of queries do not affect performance of white-box versus black-box models, the runtime is substantially greater for black-box attacks. We evaluate the impact of the number of unused features on the attack runtime. Based on the trends in Figure 1c, we hypothesize the impact grows exponentially. We argue in Section 4 that this aspect can be used to thwart such attacks.

Figure 1b shows the effect of varying the number of features we query on the prediction space recovered with 1000 queries. The results are averaged over ten samples. Interestingly, the adversary is quite successful in attacking both the random forest and DNN even with a limited subset of model input being queried.

Key Takeaways: Model inversion and limited querying attacks can recover the service provider's private information efficiently in white-box models. Although results demonstrate that black-box models are theoretically effective, they are not efficient due to higher runtimes. The simplicity of the attacks heighten the risk of server-side privacy leakage.

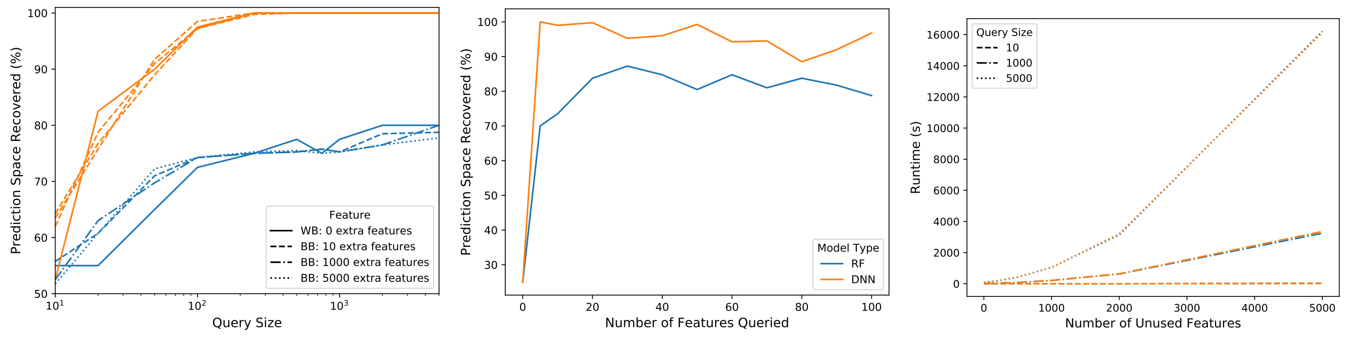
4 PRESERVING SERVER-SIDE PRIVACY

On-device models are more vulnerable to server-side privacy attacks as they have the ability of running offline. This reduces the service provider's authority of its usage and makes it difficult to track or identify adversarial actions. We present four preliminary ideas on protecting against server-side privacy attacks.

4.1 Countermeasures for White-Box Attacks

Inference on Encrypted Models. One solution for white-box models is using homomorphic encryption [6]. Doing inference on encrypted data can solve the problem of the adversary having access to too much information. In this case, $\mathcal{S}$ would hold the secret keys to decrypt the output and every time a unique output is produced, the user will have to connect with $\mathcal{S}$ to decrypt it. Thus, $\mathcal{S}$ will have more authority over what is revealed. Encryption will also protect against traditional gradient-based (i.e., model inversion) attacks. However, encryption induces additional computational overhead.

Distributing Service. Alternatively, we propose distributing the output to thwart against white-box attacks. The model output is often tied to some tangible service. If the inference can be conducted on the device but the mapping of model output to tangible service be kept on some central node, then each time the adversary executes a unique query, they will need to connect to a central node. Since this method does not require a continuous connection to the central node, the latency to conduct this is lower than doing full inference



(a) (b) (c)

Figure 1: Results of evaluating the white-box (WB) and black-box (BB) querying attacks on random forests (RF) and deep neural networks (DNN): (a) impact of varying query sizes on the attack efficacy; (b) impact of varying the number of features being queried in the white-box attack; and (c) impact of the number of unused features on runtime in the black-box attack.

on a centralized server. In this manner, the number of queries can be limited and adversarial queries can be more readily detected.

4.2 Countermeasures for Black-Box Attacks

Increasing Dimensionality. Prior work has suggested high data dimensionality as a preventative measure for black-box attacks [9]. Since the subset of features employed by the model are unknown to the adversary in this work, a simple yet effective method is collecting more data, regardless of whether it is used in model training. This will increase the runtime to leak proprietary information as seen in Figure 1c, making it more difficult to conduct the attack.

Query-based Model Degradation. Additionally, we propose a query-based model degradation method which degrades the weights of the model as more queries are conducted. Eventually the model will output random noise, protecting the hidden function and output from the adversary.

5 RELATED WORK

Most common attacks in the privacy preserving machine learning domain are concerned with user privacy. Membership inference attacks intend to identify the existence of a user in the training set [10] whereas model inversion attacks aim to extract sensitive features by reverse engineering a trained model [4]. Alternatively, property inference attacks focus on extracting hidden global patterns in the training data [5].

Closer to our work, model extraction attacks aim to learn information about the model itself. Model extraction attacks use a trained ML model to extract model parameters and learn an equivalent shadow model to poison with adversarial examples [8, 9] or monetize off of the model [11]. However, most applications of model extraction have been focused on user privacy.

Our work deals with server-side privacy of on-device models. There has been limited work on exploring privacy of ML models beyond user privacy. Although some concepts overlap with existing model extraction literature, such as recovering the model's representations, we argue that privacy from the service provider's point of view offers a unique set of challenges. To the best of our knowledge, this is one of the few works that evaluates the extent to which a service provider's intellectual property can be exploited and misused in on-device models. With the growing trend towards deploying ML models on the device, this is an important problem.

6 CONCLUSION

In this work, we established the importance of server-side privacy in on-device service models. We developed a taxonomy of white-box and black-box on-device models and proposed a number of privacy attacks. Our results demonstrated that such attacks can reveal a service provider's proprietary information with little effort in white-box settings whereas they are more inefficient, yet still feasible, in the black-box setting. To protect against such attacks, we discussed preliminary solutions.

REFERENCES

[1] 2019. Open Neural Network Exchange. https://onnx.ai/. Accessed on 01/18/2022.
[2] Google Developers. 2022. Why On-Device Machine Learning? https://developers.google.com/learn/topics/on-device-ml/learn-more. (Accessed on 01/18/2022).
[3] Yifan Ding, Liqiang Wang, Huan Zhang, Jinfeng Yi, Deliang Fan, and Boqing Gong. 2019. Defending against adversarial attacks using random forest. In *Proceedings of the IEEE/CVF Conference on CVPR Workshops*.
[4] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the ACM SIGSAC Conference on CCS*.
[5] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. 2018. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the ACM SIGSAC Conference on CCS*.
[6] Thore Graepel, Kristin Lauter, and Michael Naehrig. 2012. ML confidential: Machine learning on encrypted data. In *Proceedings of the International Conference on Information Security and Cryptology*.
[7] Rahul Gupta, Aman Alok, and Shankar Ananthakrishnan. 2019. One-vs-all models for asynchronous training: An empirical analysis. *Proceedings of the International Speech Communication Association INTERSPEECH* (2019).
[8] Seong Joon Oh, Bernt Schiele, and Mario Fritz. 2018. Towards reverse-engineering black-box neural networks. In *Proceedings of the International Conference on Learning Representations*.
[9] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the ACM on Asia Conference on CCS*.
[10] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of IEEE Symposium on Security and Privacy*.
[11] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *Proceedings of the USENIX Security Symposium*.