Timing Channels in Adaptive Neural Networks

Ayomide Akinsanya
Stevens Institute of Technology
aakinsan@stevens.edu

Tegan Brennan
Stevens Institute of Technology
tbrenna5@stevens.edu

***Abstract*—Current machine learning systems offer great predictive power but also require significant computational resources. As a result, the promise of a class of optimized machine learning models, called adaptive neural networks (ADNNs), has seen recent wide appeal. These models make dynamic decisions about the amount of computation to perform based on the given input, allowing for fast predictions on "easy" input. While various considerations of ADNNs have been extensively researched, how these input-dependent optimizations might introduce vulnerabilities has been hitherto under-explored. Our work is the first to demonstrate and evaluate timing channels due to the optimizations of ADNNs with the capacity to leak sensitive attributes about a user's input. We empirically study six ADNNs types and demonstrate how an attacker can significantly improve their ability to infer sensitive attributes, such as class label, of another user's input from an observed timing measurement. Our results show that timing information can increase an attacker's probability of correctly inferring the attribute of the user's input by up to a factor of 9.89x. Our empirical evaluation uses four different datasets, including those containing sensitive medical and demographic information, and considers leakage across a variety of sensitive attributes of the user's input. We conclude by demonstrating how timing channels can be exploited across the public internet in two fictitious web applications — Fictitious Health Company and Fictitious HR — that make use of ADNNs for serving predictions to their clients.**

I. Introduction

Machine learning services have quickly risen in popularity, allowing companies and individuals to reap the benefits of efficient and accurate deep learning models for tasks ranging from image recognition to machine translation. With the continued increase in the use of machine learning services, these systems are handling an increasing amount of data, including potentially sensitive user data, such as demographic or medical information. Cyber-attacks that steal such confidential information can be devastating, and, as more and more information is stored and manipulated in these systems, securing data against information leakage remains pivotal.

That machine learning systems are not immune to cyber-attacks aimed at recovering confidential information has already been effectively demonstrated. Membership inference attacks [44], [41], [63], for example, have been used against machine learning models to determine if an input sample was part of the target model's training set. Such attacks potentially reveal sensitive information about users whose data was used in training.

Side-channel vulnerabilities provide an avenue for an outsider to learn confidential information about a system by observing the system's non-functional side effects, such as power usage or execution time. Though side-channel vulnerabilities have been known for decades, they remain challenging to defend against. The susceptibility of machine learning models to side-channel vulnerabilities is a recent concern, and current proof-of-concept work demonstrates the potential of side-channel vulnerabilities as an avenue of information leakage in machine learning systems. Already, a variety of side channels have been exploited to retrieve sensitive information about deep learning systems [1], [33], [10].

Adaptive neural networks are a class of deep learning models that show promise in addressing the memory and workload requirements of DNNs through a wide-range of optimizations that trade accuracy for efficiency. Until now, no existing side-channel work has considered these adaptive optimizations as a avenue of potential side-channel leakage. In our work, we demonstrate that the optimizations made by ADNNs can introduce critical timing-channel vulnerabilities in machine learning models, allowing an observer to learn key attributes of the user's input.

While many kinds of ADNNs have been introduced, all share a similar high-level objective: to invest a variable amount of computation based on the received query in order to minimize computational overhead. This most commonly arises through dynamic decisions to invest less computation on input that can be predicted quickly with high accuracy. In making these decisions, we observe that the amount of computation performed, and hence likely the prediction time of the model, can be correlated with the user input. It is this correlation which introduces timing channels into ADNNs.

Our work is the first to demonstrate such timing-channel vulnerabilities. We consider the Machine-Learning-as-a-Service (MLaaS) model, an umbrella term for a set of cloud-based tools offered by service providers. The MLaaS model allows users to benefit from powerful ML models without needing to invest the resources to train and develop those models themselves. Pivotal to our study is that under such a model, multiple clients (whether individuals or employees of businesses) make queries and receive responses from the same ML model. We demonstrate how an adversary who is also a client of a target model $\mathcal{F}$ and who is able to take timing measurements of the response time of $\mathcal{F}$ to user queries is able to leverage timing information to infer sensitive properties of a victim user's input.

Our contributions are as follows:

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24125
www.ndss-symposium.org

- We present the first demonstration of timing-channel leakage due to optimizations made by ADNNs.
- We develop a strategy for leveraging timing channels in adaptive neural networks to learn sensitive user information under a clear and realistic threat model.
- We evaluate timing-channel leakage across six kinds of different ADNNs, considering four different datasets and three types of private information and demonstrate how to exploit timing channel leakage in two fictitious web applications that employ ADNNs.
- We explore the impact of the model hyperparameters such as exit thresholds and architecture depth on timing channel-leakage, accuracy, and performance across all six different ADNNs we consider.

The rest of the paper is organized as follows. In Section II, we present the required background on ADNNs and side-channel analysis. In Section III, we walk through a motivating example of how timing channels can arise in ADNNs. We present our problem definition, threat model, and approach to leveraging timing channels to learn sensitive attributes about user's input in Section IV. In Section V, we present the results of our experiments, and we further discuss some of our results in Section VI. In Section VII, we demonstrate the exploitation of these timing channels over the public internet through two case studies. We discuss the limitations of our work in Section VIII. Our related work is given in Section IX, and we present our conclusions and future work in Section X.

II. Preliminaries

Memory and workload requirements of DNNs are a major challenge in the adoption of deep learning techniques, especially on smaller devices, including smart phones, smart appliances and other embedded devices. There has been substantial research aimed at minimizing these requirements, and one recent, promising avenue is a class of networks called adaptive neural networks. These networks make dynamic decisions at runtime about the amount of computation to perform based on factors such as the available resources, deadlines, or, most commonly and also most critically for our work, the given input. This allows ADNNs to leverage the observation that not all input is equally challenging to process and invest a variable amount of effort based on the input difficulty. Ultimately, this results in a trade-off between accuracy and performance.

a) Adaptive Neural Networks: Multiple types of ADNNs have arisen over the past few years. **Early-exit neural networks** [48], [20], [13], [23] are neural networks in which a backbone classifier is augmented with additional exit nodes. These early exits may be taken when an input sample is deemed "easy" and able to be inferred with high confidence without being processed by the entire classifier. Related **dynamic neural networks** [30], [32], [49], [52], [56], [14], [43] likewise take advantage of the observation that inference might be expedited if different inputs take different computation paths and introduce various methods to dynamically select what part of the inference graph to use for given input. These techniques include skipping over parts of the inference graph, dynamically pruning the graph, or fractionally executing certain layers. **Model selection or cascades** [61], [47], [28], [24] are a type of adaptive inference where a family of models (with varying computational requirements) is trained for the same task. At inference time, either the appropriate model for the given input is selected or the input is propagated through more complex models until a certain criteria is met. Most current ADNNs are aimed towards tasks in computer vision, most commonly classification. Nevertheless, recent work [42], [58], [66], [46] explores the potential of early-exit neural networks to accelerate inference in DNNs for natural language processing.

b) Side Channels in Machine Learning Systems: Side-channel vulnerabilities in machine learning systems have been leveraged to compromise multiple types of data, including extracting sensitive model parameters [10], [15], [64], [5], [4], [55], [33], [53], [26], [19], [60] and learning class labels on unknown user input [1], [50], [54], [45]. Our work targets the problem of user privacy in a Machine-Learning as-a-Service (MLaaS) framework. In such a framework, a user is able to interact with a machine learning model through an API, securely sending their own data to the model and receiving the model's inference results on that data. Through this interaction, the user's input to the machine learning model and the output of the machine learning model should remain private.

Under such a framework, we formulate the side-channel problem as follows: Consider the inference performed by a neural network as a function which takes some sensitive user input x from input domain $\mathbb{X}$ and returns its inference result, along with a vector of side-channel observables o. These observables are non-functional characteristics of inference such as execution time, memory used, or packets transmitted over a network. A side channel exists if there are at least two mutually exclusive subsets of the input space $\mathbb{X}$ that can be partitioned using the observables o. In addition to properties of the neural network itself, whether such a partitioning is possible is also affected by what ability an observer has to take precise measurements as well as noise in the observations. While side channels can arise due to numerous observables, we focus on side channels in time, also called timing side channels or timing channels, in our work. Also note that while the existence of any two mutually exclusive subsets constitutes a side channel, not all partitions of the input space might result in a realistic security vulnerability, a point that we will further discuss as it pertains to side channels in machine learning systems.

III. Motivation

In dynamically choosing how much computation to perform based on the difficulty of the input, ADNNs introduce multiple possible computation paths for inference. It is possible that some of these inference paths can be correlated with some sensitive attribute of the input. Take, for example, BranchyNet[48], an early-exit network architecture that allows results for some input to exit the network early when results for those input can be inferred with high confidence. BranchyAlexNet, a simple BranchyNet architecture with early-exit paths added to the original AlexNet, is depicted in Figure 1.

Notice how the structure of this ADNN resembles the control flow graph of a program with three possible return statements where the chosen exit is input-dependent. This

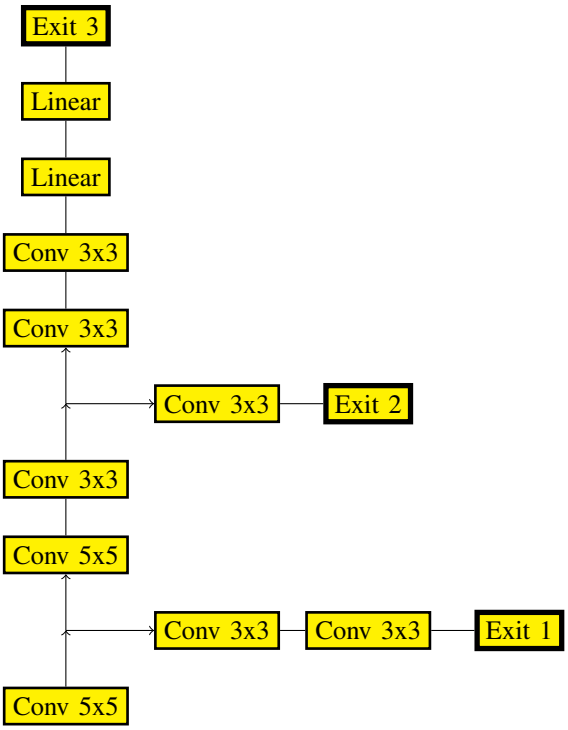

Fig. 1: Branchy-AlexNet: A simple BranchyNet architecture [48] with two branches added to the original AlexNet. Each bold "Exit" node denotes an exit point of Branchy-AlexNet.

creates a partitioning of the input domain into three different clusters according to which exit is taken. Should the execution time of inference be correlated with which exit is taken (a reasonable assumption given that more computation is done for later exits but which we experimentally validate in Figure 2), then there exists a timing channel partitioning the input space according to exit taken. An observer with the capability to make timing measurements would therefore be able to infer which partition or cluster an input belongs to based on the observed timing measurement. If one or more of these clusters correlates to a sensitive attribute (such as a unique class label), then this would result in leaking private information about the input (in this case the class it belongs to) and could be considered a privacy violation. An example of such a privacy violation arising is given in Figure 3. In this figure, we observe how the execution time for Branchy-AlexNet varies according to whether the input image is of a benign or malignant skin mole. When the execution time is low, it is much more likely that the input image is of a benign mole. An adversary can use thus use this timing information to make a more informed inference about the private input image than they would be able to by guessing simply according to input distribution. It is important to note; however, that our goal is not to find correlations between model execution time and every possible attribute as this is rarely, if ever, possible. Privacy violations occur even when there is a single characteristic timing profile that correlates with some sensitive attribute as in the case just discussed. The stronger the correlation, the more confident an adversary might be about inference on the input image when observing that timing profile.

Ultimately, just as how side channels in software arise due to an observable difference in resource usage across program paths, we hypothesize that so too can side channels in neural networks arise due to an observable difference in resource usage across inference paths. By construction, ADNNs introduce multiple input-dependent inference paths, which results in different model execution time distributions and thereby introduces the potential for side channels.

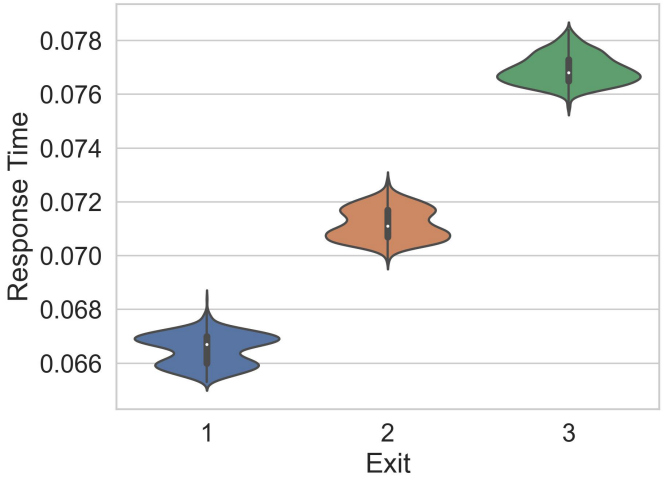

Fig. 2: Execution time (in seconds) for input from the CANCER dataset (introduced in our experimental setup) taking each of the three exits of Branchy-AlexNet

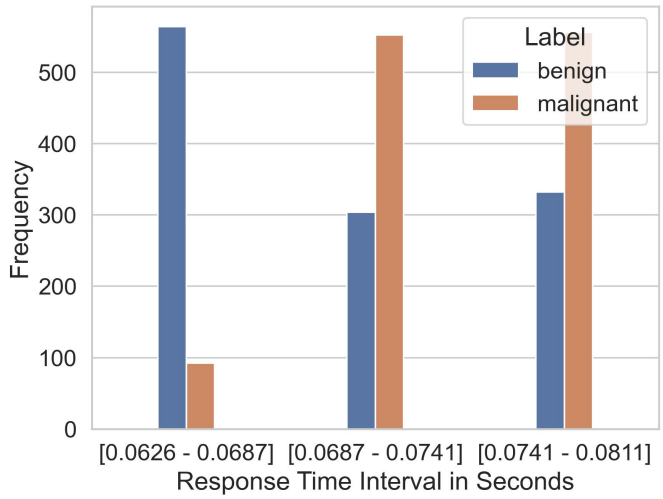

Fig. 3: Execution time ranges (in seconds) for input from the CANCER dataset (introduced in our experimental setup) taking each of the three exits of Branchy-AlexNet

IV. Analyzing Timing Channels in Adaptive Neural Networks

In this section, we present our approach to analyze timing channels in ADNNs. We begin with the problem definition. As previously introduced, a timing channel exists if there are at least two mutually exclusive, non-empty subsets of the input

space $\mathbb{X}$ that can be partitioned using the observable o. For a timing side channel, the observable is the collected timing measurements. Not all partitions of the input space result in realistic security vulnerabilities; however. For example, consider the partitioning of the input space of images arising from the three different exits in Figure 1. If there is no meaningful difference between the images that belong to each partition, then the attacker is not able to glean any meaningful information about the input image even if a timing channel leaks which partition it belongs to. Phrased in another way: if, for any pair of previously unseen images x_1, x_2, the probability that an unknown $x \in \{ x_1, x_2\}$ is x_1 remains the same after the attacker observes which partition x belongs to, then no meaningful information is leaked by the side channel. This naturally leads to the question of what information about images might be meaningful to an attacker.

A. Attributes

For the given input space, $\mathbb{X}$, an attribute $\mathbb{A}$ over $\mathbb{X}$ is a finite set of values such that every element of $x \in \mathbb{X}$ is deterministically mapped to an element $a \in \mathbb{A}$. We focus on attributes that characterize sensitive properties of the input domain. Considering user-provided images as input, such attributes might include class label, predicted class, whether an input is misclassified, whether an input is above a certain resolution, or whether an input is adversarial. Attributes might even consider whether an input has a certain feature related to a network layer.

B. Threat Model

The capabilities of the adversary also impact whether a timing channel poses a realistic security threat. In our threat model, the attacker monitors the timing characteristics of TLS/SSL encrypted network communication between the client (victim) and the Machine Learning as a Service (MLaaS) server. This monitoring can occur over a Local Area Network (LAN) or across the public internet. By measuring the response time of the ML server to the client's request, the attacker intends to utilize this observed timing information to infer a sensitive attribute of the user's input to the model. The attacker aims to achieve this by leveraging a pre-trained attack model $\mathcal{A}$ described in Section IV-C2.

As an example, let's consider a scenario where a hospital application utilizes a machine learning model $\mathcal{F}$ to provide responses (predictions) y_p to client queries x. It is assumed that all inputs to the application are valid images. An example application could be a web application for predicting skin cancer, where clients submit images of skin moles and receive predictions regarding the malignancy or benignity of the depicted mole. The resulting prediction from the underlying model should only be disclosed to the client, making the prediction highly confidential. We assume that the attacker has black box access to the model with no additional knowledge of the model internals. We also assume that the model owner (possibly the hospital/hospital's partner) that provides the MLaaS to its clients is trusted and would not try to breach the confidentiality of the model's prediction on their client's input. The attacker in this scenario is some malicious client of the hospital MLaaS who knows when another client's input would be queried against the hospital MLaaS. The assumptions about communication between attacker, client, and MLaaS server are summarised below:

1) The attacker and victim(s) are on the same network and within the same location.
2) The MLaaS server handles and responds to queries individually not in batches.
3) The communication between the client and the MLaas server is over a TLS/SSL encrypted network stream, hence trying to break the encryption itself is outside the scope of this work.
4) Despite the encryption of the network stream's contents (TCP/IP packet payloads), various visible metadata, such as packet sizes, timings, directions, and flags, can still be extracted from message headers using network analysis tools like Wireshark,Tcpdump.

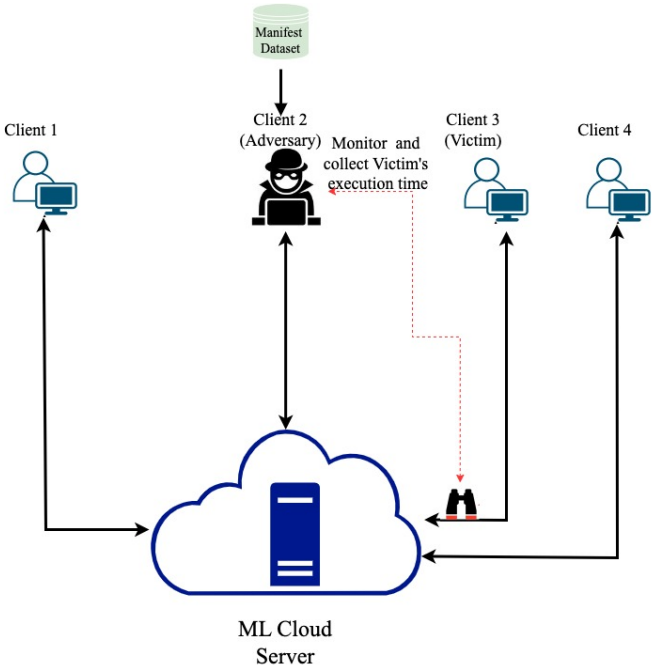

Fig. 4: Privacy Leakage Threat Model in a Client-Server MLaaS Framework

Adversary's Capabilities. We consider an adversary who is also a client of the application. They can submit queries to the model $\mathcal{F}$ and receive a prediction from $\mathcal{F}$. We assume that the adversary can submit as many queries as desired and will not be flagged for suspicious behavior. The adversary can observe and collect the response time of the application on both their own queries and those of other users, using packet capture tools such as Wireshark, Tcpdump etc. The adversary has access to a manifest dataset $DS_{\mathcal{M}}$. The manifest dataset is a dataset which is not necessarily a subset of the original training dataset of the target model $\mathcal{F}$, but belongs to the same input distribution as the training set.

Adversary's Knowledge. The adversary has no access to the model internals. The adversary does not know the model architecture, weights, or parameters. The adversary only knows when the victim's query x is submitted to the MlaaS model $\mathcal{F}$.

Adversary's Goals and Objectives. The adversary's objective is to infer a sensitive attribute a of the victim's input x to the model. This sensitive attribute a could be the class label y, the predicted class label y_p, or some other property of the input x. If there is any user input for which the attacker successfully determines its attribute, then a privacy violation exists.

C. Attacker's Strategy for Using Timing-Channel Leakage

Not all timing channels in ADNNs are strong enough to pose realistic security threats. A number of factors, such as noise in timing measurements or the degree to which the input space is partitioned, can impact whether the timing channel is dangerous and should be mitigated. Ultimately, we consider these channels to be dangerous if they increase an adversary's probability of correctly inferring a sensitive attribute about user input by a considerable margin. In order to determine whether a timing channel accomplishes this, we propose the following attack strategy for an adversary :

1) Obtain Timing Profile $\mathcal{T}$: To create the timing profile $\mathcal{T}$ of the ADNN, we query the target ADNN model $\mathcal{F}$ using data samples (x_i, a_i) from a manifest dataset $DS_{\mathcal{M}}$, and record the model's inference time (t_i) observed over the network into a one dimensional array T. The *manifest dataset* $DS_{\mathcal{M}}$ is a set of data samples that is available to the attacker. It is not necessarily a subset of the original training dataset of the target model $\mathcal{F}$, but belongs to the same input distribution as the training set. The timing profile $\mathcal{T}$ is completed by pairing each recorded timing measurement t_i with the attribute a_i of the corresponding input x_i, which we formally define as $\mathcal{T} = (t_1, a_1), ..., (t_k, a_k)$ where k is the number of data samples present in $DS_{\mathcal{M}}$.

2) Train Attack Model $\mathcal{A}$: The next step is to train an attack model $\mathcal{A}$ that takes as input t_i and outputs a prediction for the sensitive attribute a_i for any given input x_i to the target ADNN model $\mathcal{F}$. We train the attack model $\mathcal{A}$ using our timing profile $\mathcal{T}$. The *key goal* of this step is to have the attack model learn how the timing channel information t_i correlates the input x_i with the sensitive attribute a_i. To evaluate how well our attack model $\mathcal{A}$ is able to learn this correlation, we adopt the model's accuracy, i.e., attack success rate (denoted as ASR). This is similar to work on membership inference attacks [44], [41], [63], except in this case we are interested in inferring the sensitive attribute a_i of an input x_i to the target ADNN model $\mathcal{F}$, given it's observed inference time t_i over the network, as opposed to trying to infer whether x_i was used to train the target ADNN model $\mathcal{F}$.

a) Attack Model Parameters: Our attack model $\mathcal{A}$ is a simple deep neural network (DNN) with two hidden layers comprising of sixty-four and thirty-two neurons respectively, and using ReLU as an activation function. Other models, such as logistic regression or decision trees, might also serve as an attack model, though our exploratory experiments showed the simple deep neural network to outperform them in terms of accuracy. Given the simple nature of our attack model, it can be trained very easily and quickly usually only taking minutes to train. The attacker can then leverage the model to infer the sensitive attribute of a new input.

3) Cluster Timing Observations T: ASR provides us with a metric to understand the probability of correctly guessing the attribute over the space of all input images given their respective timing observations. We observe; however, that this probability is not necessarily uniform: there might be some input for which the probability of correctly inferring the attribute is high given their timing observations and others for which it is low given their timing observations. We introduce a clustering-based partitioning algorithm in order to address the more nuanced question: are there certain time partitions of the input space for which the ASR is high?

To do this we leverage Kernel Density Estimation (KDE) [39] an unsupervised statistical method for clustering one dimensional data, to partition this one dimensional array T into different time clusters C. We adopt KDE over other popular clustering algorithms such as K-means [35], kNN [31], and DBSCAN [11] because KDE is well poised for clustering one-dimensional data, while the other algorithms are better suited for multidimensional data. Figure 5 shows the clustering of the recorded inference times of the Branchy-AlexNet architecture over the CANCER dataset. We observe that KDE fits T with a smoothed line, which can be partitioned into three different clusters based on the different minimas present in the curve. In the case of Branchy-AlexNet, the number of clusters found by the KDE algorithm corresponds to the number of exits in the ADNN, but this not always the case. The closer the inference times of different exits are to each other, the more likely KDE is to put them all together in one single cluster. The number of clusters is impacted by the KDE algorithm's bandwidth parameter. We record the accuracy of the attack model on each cluster, defining this accuracy as the ASR/cluster.

In the context of the timing variability (timing channels) of ADNNs, the time clusters C serve as representations of this variability. When a time cluster c_j exhibits a high cluster ASR, it signifies a strong correlation between that specific cluster c_j and a particular sensitive attribute a^*. This correlation suggests that any timing observation t belonging to cluster c_j reveals information about the sensitive attribute a associated with a given input x. We view this as a privacy violation, recognizing the potential breach of privacy when timing observations fall into such clusters. We summarize our analysis approach in Algorithm 1.

Algorithm 1 Analyzing Timing Channel Leakage

Require: Manifest dataset $DS_{\mathcal{M}} = \{(x_1, a_1), ..., (x_k, a_k)\}$
1: **for all** (x_i, a_i) in $DS_{\mathcal{M}}$ **do**
2: query target model $\mathcal{F}$ with (x_i, a_i) to obtain $T = [t_i, ..., t_k]$
3: pair t_i with a_i of the corresponding input x_i to obtain timing profile $\mathcal{T} = (t_1, a_1), ..., (t_k, a_k)$
4: **end for**
5: train attack model $\mathcal{A}$:$\{t_i\} \rightarrow a_i$ using $\mathcal{T}$
6: partition T using KDE into $C = \{c_1, ..., c_n\}$ where n is number of time clusters in T
7: calculate $\mathcal{A}$ ASR and $\mathcal{A}$ ASR_j for each cluster for each cluster c_j
8: return $\mathcal{A}$ ASR and $\mathcal{A}$ ASR_j for each cluster c_j

Deploying the Adversary's Strategy. In order to successfully infer the private attributes of other user's input, the

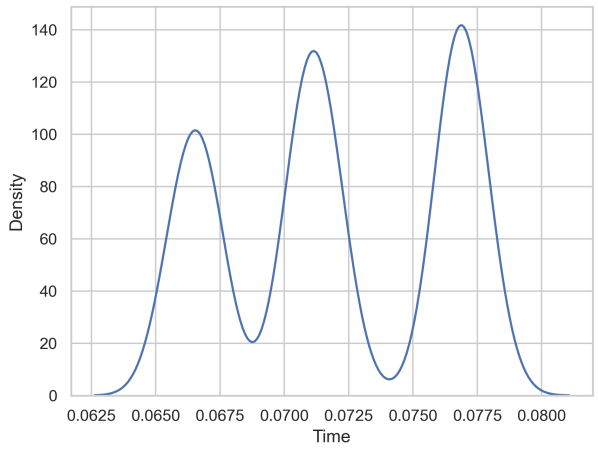

Fig. 5: Execution time clusters (in seconds) for Branchy-AlexNet over the CANCER dataset.

adversary implements a two-stage approach. In the first stage of this approach, the adversary trains an attack model as described in Section IV-C2. Because timing measurements are nondeterministic, the attacker may choose to repeatedly query the model with the same input to train a more robust attack model.

With the trained attack model $\mathcal{A}$ in hand, the attacker is prepared to infer attributes of the user's input. When a target user makes a query to the model $\mathcal{F}$ with some unknown input x^* and receives a response, the attacker will collect the appropriate timing observation t^*. For every t^* collected, the attacker will then use their attack model $\mathcal{A}$ to infer the sensitive attribute a of the user input x^*, using their timing observation t^*.

While the training stage is performed in advance of the inference stage, the attacker also has the ability to update their attack model $\mathcal{A}$ should it be deemed beneficial. For example, should the attacker somehow determine that the input distribution has diverged significantly from that of the manifest dataset, the attacker might want to update their attack model with observations based on inputs from this new distribution. Similarly if some change in the underlying model is detected (for example, it has been increasingly optimized), the attacker might wish to collect new timing measurements to create a new attack model for this more optimized model.

V. Experimental Evaluation

A. Experimental Setup

a) Local Area Network (LAN) Setup: For all experiments, we hosted the target machine learning model on our server machine using the Python Webservice Framework Flask v1.1.2 and timed the responses on our client machine connected over a local area network. Our server machine was a MacBook Pro 2021 (Apple M1 Pro, 16GB RAM), and our client machine was a Dell XPS 11th Gen (2.4GHz 8-Core Intel core i5, 16GB RAM). Both machines are on our Ethernet LAN via a WAVLINKAC 1200 router. Another three computers

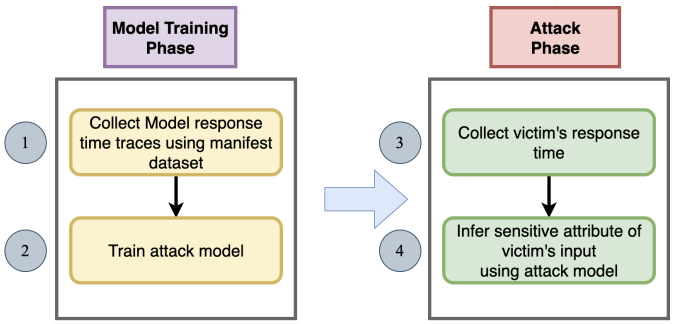

Fig. 6: Two-Phase Adversary Strategy

are on the LAN. Under low load, the typical round-trip time between the client and the server machines through the LAN was 1.35 msec (min 0.540, mean 1.347, max 1.849).

b) Machine Learning Models: For our experiments we consider six different types of ADNNs, namely; the early-exit networks Branchy-AlexNet and Shallow Deep Network, the model cascade networks RANet and MSDNet, and the dynamic networks BlockDrop and SkipNet. All six of these ADNNs are aimed at the task of image classification. Some additional notes about each follow below.

Early-Exit Networks.

Branchy-AlexNet [48], a network architecture developed by Teerapittayanon et al. was introduced in our motivation. The network consists of two early exit classifiers attached to the main network.

Shallow-Deep Networks (SDNs) [23] another example of early exit networks, modify traditional Deep Neural Networks (DNNs) by adding internal classifiers, known as early exits, at various locations within the network. SDNs prevent *overthinking* in DNNs, where accurate predictions can be made before the final network layer. This enhances computational efficiency while maintaining accuracy. SDNs and Branchy-AlexNet differ in their heuristic approach for early exits. SDNs utilize confidence scores and a metric called the confusion metric, while Branchy-AlexNet relies on entropy as a measure of early exit classifier confidence. In our experiments we make use of the VGG and ResNet-56 variants.

Model Selection/Cascade networks.

Multi-Scale Dense Networks (MSDNets) [20], an example of a model selection/cascade network, are designed to capture multi-scale information and exploit dense connectivity patterns. MSDNets address the challenge of effectively modeling both local and global contexts in complex data. In MSDNets, the network is structured into multiple scales or levels, with each level consisting of dense blocks. Dense blocks are densely connected layers where each layer receives inputs from all preceding layers. This dense connectivity promotes feature reuse and enables effective information flow throughout the network. The multi-scale aspect of MSDNets allows them to capture information at different levels of granularity. Each scale focuses on a specific receptive field size, allowing the network to analyze fine-grained details at lower scales and capture broader context at higher scales. By integrating infor-

mation from multiple scales, MSDNets can effectively capture both local details and global relationships of different inputs based on their perceived complexity and classify them using the most minimal network.

RANet (Resolution Adaptive Networks) [61], an improvement on MSDNet, is another example of a model selection/cascade network. This network is an ensemble of models at three different scales and inputs are propagated from lower scales where the computational requirements are low to higher scales which have higher computational demands. Inputs exit the network once the exit criteria at the classifier is met. When it is not met, they are propagated to the next higher scale alongside the feature maps learned from the lower scales, which improve the ability of the classifiers at higher scales to infer the input. RANet uses the intuition that we can classify some inputs (easy inputs) at low scales/resolution, while other inputs which are referred to as hard inputs can be classified using higher scales or resolutions. Inputs deemed as easy contain very similar pixel features when the object in the image is large enough The hard images contains less similar pixel features when the object in the image is of a very small size.

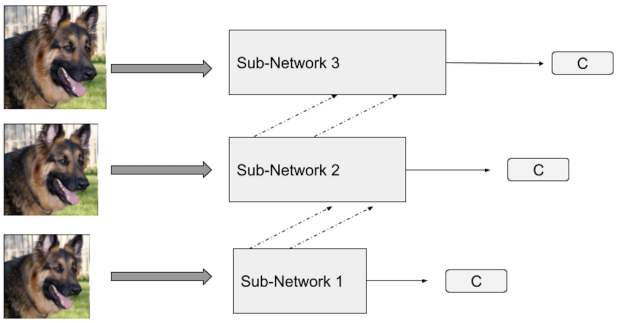

Fig. 7: RANet input being fed to sub-networks with different scales [61]

Dynamic Networks.

BlockDrop [56] is an example of a dynamic network. BlockDrop aims to reduce the total amount of computation of an original ResNet [17] by dynamically selecting layers of blocks in the original ResNet to execute while ignoring the others. This dynamic selection is based on user inputs, hence BlockDrop produces different computation paths when evaluating different inputs. BlockDrop consists of a policy network (an agent) which takes in an input image and outputs the minimal configuration of blocks that is needed to correctly classify it. Thus the policy network outputs different block configurations for different inputs, allowing for them to take different computations paths through the main network.

SkipNet [52], another example of a dynamic neural network, incorporates a gating network, which acts as a decision-maker, to selectively determine whether to skip certain convolutional blocks based on the activations observed in the previous layer. This dynamic skip mechanism allows the network to adaptively adjust its architecture to focus on relevant features, while preserving accuracy and improving overall computational efficiency.

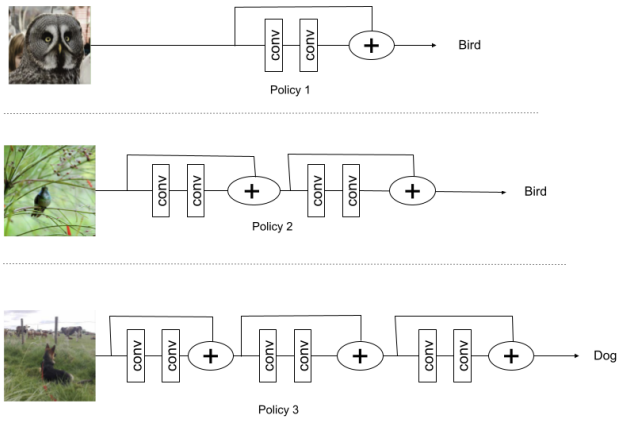

Fig. 8: BlockDrop policy network outputing different block policies for different inputs [56]

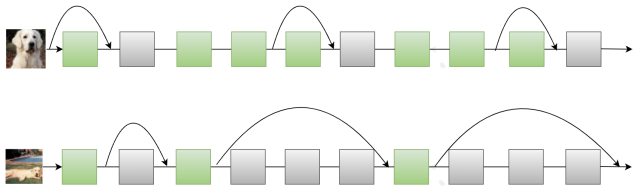

Fig. 9: SkipNet skipping different convolutional layers based on the difficulty of the input [52]

c) Attributes: For our experiments, we consider three different, potentially sensitive attributes.
CLASS LABEL: the actual class of the input.
GENERALIZED LABEL: the super category of the class of the input. For example, the CIFAR100 dataset can be super categorized into twenty (20) super classes, where the classes bear, leopard, lion, tiger, wolf belong to the super class large carnivores and the classes orchids, poppies, roses, sunflowers, tulips belong to the super class flowers.
ADVERSARIAL INPUT: whether or not a particular input is adversarial.

d) Datasets: We consider four datasets in our experiments: CIFAR10, CIFAR100, CANCER, and FAIRFACE. Branchy-AlexNet, SDN , MSDNet, RANet, BlockDrop and SkipNet were each trained on the CIFAR10 and the CIFAR100 datasets[25] which contains 32 × 32 RGB natural images, corresponding to 10 and 100 classes respectively. The CIFAR10 and CIFAR100 dataset both contain 50,000 training and 10,000 testing images. Branchy-AlexNet, SDN(VGG and ResNet-56 variant), MSDNet, RANet, and SkipNet were also trained on two additional datasets. The first is the publicly available Skin Cancer dataset from the ISIC archive [12]. This dataset consists of 224 × 224 images of benign and malignant skin moles. The dataset originally contains 2637 training and 660 testing images. We applied standard data augmentation strategies [18] to further increase the size of our training and testing dataset to 10548 training images and 2400 testing images equally split between benign and malignant classes. The second is

the FAIRFACE dataset [27]. This dataset consists of 108,501 split into 96336 training images and 12165 testing images. The dataset is made up of 224 × 224 images and labelled with race, gender and age groups. For this dataset, we focus on training the different ADNNs to predict just the age group which contains 3 respective classes, and we ensured that all images in the training and test dataset were equally split between all age classes. We were unable to train the BlockDrop model on the CANCER and FAIRFACE datasets because of problems with replicating the training script in the newest PyTorch version.

e) Training: For these experiments, we made use of the publicly available implementation of the Branchy-AlexNet architecture [6] based on PyTorch and proceeded to train our network consisting of three exits, following the configurations from the original paper [48] by first pre-training the backbone network on the datasets for 100 epochs and then jointly training the backbone together with the exits using weights 1.0,1.0,0.3 on the 3 exits respectively. In our experiments on SDN, MSDNet, RANet and SkipNet, we followed the configurations from the original papers [23], [22], [20], [21], [61], [62], [52], [51], and for our experiments on BlockDrop, we used the released pre-trained model weights built with ResNet 110 on the CIFAR10 dataset and for the CIFAR100 dataset we use the pre-trained model weights built with ResNet 32 [57].

f) KDE Bandwidth Parameter: For our experiments, we use the default KDE bandwidth parameter of 1, which provides a balanced trade-off between capturing local details of the continuous timing observations and smoothing the density estimate.

g) Running Our Experiments: To create the timing profile, we use the test set of the respective datasets as our manifest dataset. We train our attack model for a total of 100 epochs, using adam as an optimizer. We repeat each timing measurements over the LAN 5 times and record the average to generate our timing profile.

B. Metrics

We report two metrics along with a baseline metric in all our experimental results, which we describe below:

a) ASR and ASR/Cluster: ASR denotes the accuracy of the attack model $\mathcal{A}$ in predicting the sensitive attribute a given any timing observation t, while the ASR/cluster denotes the accuracy of the attack model $\mathcal{A}$ to predict the sensitive attribute a for all timing observation t that fall into cluster c_j. While we report both the ASR and the ASR/cluster metric in all our results, it is important to note that our goal is not find perfect/exact correlations between the sensitive attribute a and the timing observation t (ASR), as this is rarely, if ever possible, but rather to find individual time clusters c_j that highly correlate with some sensitive attribute a^* (ASR/cluster).

b) Random Guessing Attack (RandGA): As a baseline for comparison, we implement the Random Guessing Attack. In this attack, the adversary does not need to have query access to the target model nor the ability to make timing observations on other user's input. Instead, they make predictions for the sensitive attribute by assigning probabilities to each possible value according to any knowledge they have about the input distribution of the attribute. For our experiments, we assume an equal probability distribution for all possible values of the sensitive attribute across all datasets. A comparison between the probability of an attacker correctly inferring the attribute using RandGA versus ASR and ASR/Cluster is given in Table I.

C. Experimental Results

a) Leakage Results: Our results are reported in Table I. These results show that there are timing channels in ADNNs that leak sensitive attributes about the user's input. Consider the results of Branchy-AlexNet on the CANCER dataset. We see that the highest ASR/cluster is **82.61**%, an **1.65x** higher accuracy than the RandGA of 50%. For SDNet on the FAIRFACE dataset, we find a cluster with an ASR of **94.29**%, an improvement of **2.83x** over the RandGA of 33.33%, and for BlockDrop on the CIFAR10 dataset, we find a cluster with ASR of **98.94**% an improvement of **9.89x** over the RandGA of 10%. There are also a combination of model architectures and datasets for which we do not experience high leakage. For example, the cluster with the highest leakage for Branchy-AlexNet on the CIFAR10 Dataset has an ASR/cluster of 22.22%, an improvement of only 2.22**x** over the RandGA baseline.

b) What percentage of the input space is vulnerable?: Having shown significant leakage in some networks, a natural question to pose is: what percentage of the input space is contained within clusters with high ASR? We provide these results in the "Cluster Input Distribution" column of Table I. The proportion of input that falls into clusters with high ASR varies across datasets and architectures. For example, for SDNet on the FAIRFACE dataset, a cluster with the high ASR of 94.29% contains only **5.42**% of the input distribution, whereas the cluster with the highest ASR for Branchy-AlexNet on the CANCER dataset contains **27.33**%, over a quarter of the input distribution. Even when the portion of the input space in clusters with high ASR is low, model owners and clients should consider whether even leaking attributes about a small portion of inputs might be considered a major privacy issue.

VI. Discussion

a) Impact of the Number of Possible Attributes: The results from our experiments in Table I demonstrate a trend where early exit and model cascade networks are less vulnerable to leaking sensitive attributes via their timing channels on datasets where the sensitive attribute can take a larger number of possible values. For example, experiments on the CIFAR10 dataset where the sensitive attribute (Class Label) can take up to 10 different possible values show less leakage when compared to the CANCER or the FAIRFACE dataset where the sensitive attribute can only take 2 or 3 values respectively. In the case of the dynamic network variants, we still find clusters with high ASR for the CIFAR10 dataset even with the sensitive attribute of this dataset having 10 possible values. For SkipNet, we find a cluster with an ASR of **60**%, and for BlockDrop a cluster with an ASR of **98.94**%. This is compared to 10% ASR of the RandGA, indicating that the timing channels in dynamic networks can still leak some information about the sensitive attribute, even if the sensitive attribute can take a large range of possible values.

Arch Type	Architecture	Dataset	Attribute	No Clusters	ASR/Cluster	Cluster Input Distribution	RandGA	ASR
Non-Adaptive Networks	AlexNet	CANCER	Class Label	1	46.76	100	50	46.76
	AlexNet	FAIRFACE	Class Label	1	31.64	100	33.33	31.64
	VGG-16	CANCER	Class Label	2	[47.57, 56.1]	[72.96, 27.04]	50	50.0
	VGG-16	FAIRFACE	Class Label	2	[36.81, 33.81]	[26.06, 73.94]	33.33	34.57
	ResNet-110	CANCER	Class Label	1	46.53	100	50	46.53
	ResNet-110	FAIRFACE	Class Label	1	34.57	100	33.33	34.57
Early Exit Networks	Branchy-AlexNet	CIFAR10	Class Label	3	[13.52, 13.93, 22.22]	[44.68, 26.51, 28.81]	10	16.11
	Branchy-AlexNet	CIFAR100	Generalized Label	4	[16.67, 13.04, 8.05, 5.21]	[0.84, 6.2, 2.62, 88.34]	5	6.00
	Branchy-AlexNet	CANCER	Class Label	3	[**82.61**, 69.01, 60.53]	[27.33, 35.66, 37.00]	50	70.37
	Branchy-AlexNet	FAIRFACE	Class Label	3	[**78.26**, 64.52, 41.46]	[4.20, 11.08, 84.72]	33.33	45.71
	SDNet	CIFAR10	Class Label	5	[19.22, 10.83, 14.71, 21.74, 19.61]	[21.22, 42.98, 29.32, 2.75, 3.73]	10	14.44
	SDNet	CIFAR100	Generalized Label	3	[5.64, 7.62, 9.63]	[66.98, 12.67, 20.35]	5	6.67
	SDNet	CANCER	Class Label	3	[64.5, 66.67, 62.58]	[57.58, 2.84, 39.58]	50	63.89
	SDNet	FAIRFACE	Class Label	5	[**94.29**, 42.0, 36.04 36.88, 43.3]	[5.42, 13.06, 17.75, 22.19, 41.58]	33.33	43.21
Model Cascade Networks	RANet	CIFAR10	Class Label	3	[25.0, 17.23, 10.58]	[2.23, 14.71, 83.06]	10	12.06
	RANet	CIFAR100	Generalized Label	3	[10.34, 9.47, 6.29]	[2.77, 18.72, 78.51]	5	7.06
	RANet	CANCER	Class Label	3	[**95.05**, 62.07, 66.39]	[23.35, 21.55, 55.09]	50	72.26
	RANet	FAIRFACE	Class Label	3	[**81.44**, 34.84, 45.45]	[12.83, 20.17, 67.0]	33.33	43.98
	MSDNet	CIFAR10	Class Label	3	[12.94, 13.51, 20.47]	[67.90, 23.76, 8.34]	10	13.61
	MSDNet	CIFAR100	Generalized Label	3	[9.28, 6.37, 7.27]	[19.98, 37.62, 42.4]	5	7.33
	MSDNet	CANCER	Class Label	2	[**89.32**, 61.4]	[20.83, 79.17]	50	68.06
	MSDNet	FAIRFACE	Class Label	2	[**77.38**, 36.17]	[10.92, 89.08]	33.33	41.51
Dynamic Networks	BlockDrop	CIFAR10	Class Label	2	[**98.94**, 21.57]	[4.80, 95.20]	10	25.93
	BlockDrop	CIFAR100	Generalized Label	3	[**100**,66.67,4.75]	[0.04, 0.41, 99.56]	5	5.06
	BlockDrop	CIFAR10	Adversarial Input	2	[**100**,61.16]	[4.17, 95.83]	50	62.71
	SkipNet	CIFAR10	Class Label	5	[0, **60.98**, 16.57, 16.51, 0]	[0.01, 1.66, 55.67, 42.64, 0.01]	10	17.56
	SkipNet	CIFAR100	Generalized Label	3	[0, 8.36, 6.54]	[0.1, 30.93, 68.97]	5	7.11
	SkipNet	CANCER	Class Label	5	[**77.08**, 45.83, 58.73, 51.79, 50.0]	[10.29, 31.62, 30.04, 27.88, 0.17]	50	54.63
	SkipNet	FAIRFACE	Class Label	4	[39.22, 38.06, 48.65, 0]	[46.25, 42.52, 11.19, 0.03]	33.33	39.81
	SkipNet	CIFAR10	Adversarial Input	5	[0, 66.35, 52.78, 55.23, 0]	[0.01, 17.94, 31.99, 50.04, 0.01]	50	56.51

TABLE I: LAN Results (clusters with significantly high ASR >=**75**% for CANCER, >**70**% for FAIRFACE, and >**50**% for CIFAR10 and CIFAR100 are in **bold font**)

b) Trends in "Easier" Input Images: We observe a trend in the CANCER and FAIRFACE datasets where certain attributes correspond to lower execution times regardless of the early exit or model cascade ADNN architecture. On the CANCER dataset, we observe that the early exits/ intermediate classifiers of early exit and model cascade networks learn to recognize the benign class of the dataset better than the malignant class, hence introducing a bias between the two possible class label values at the earlier exits or classifiers. All four different architectures (Branchy-AlexNet, SDNet, RANet and MSDNet) exhibit this behavior. Figure 10 shows this agreement. Although the difference in the input images of the CANCER dataset is imperceptible to a non-medically trained individual, we hypothesize that the features that the model learns at their earlier exits are more significant to the benign skin mole class. We also see the similar behaviour and agreement of all four models on the FAIRFACE dataset, where the models treats age classes [0-19] and [more than 50] as easier compared to the age class [20-49]. Figure 11 shows this agreement.

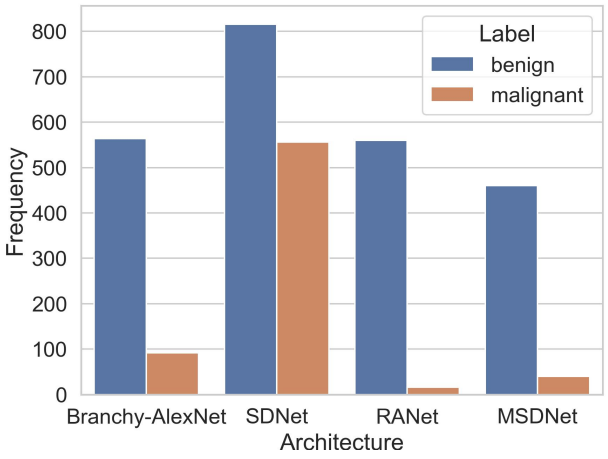

Fig. 10: Input Distribution of benign and malignant skin mole images across the first time cluster of Branchy-AlexNet, SDNet, RANet and MSDNet

Not all datasets exhibit such a trend; however. For example, the classes in CIFAR10 deemed "easy" by the two dynamic networks evaluated are not the same. This behavior is shown in Figure 12.

c) Impact of Exit Threshold Hyperparameter: An important part of developing and deploying early exit (Branchy-AlexNet and SDNet) and model cascade (RANet and MSDNet) variants of ADNNs is the choice of their exit thresholds hyperparameter. This threshold hyperparameter is greatly influenced by the type of setting the ADNN is to operate in. We

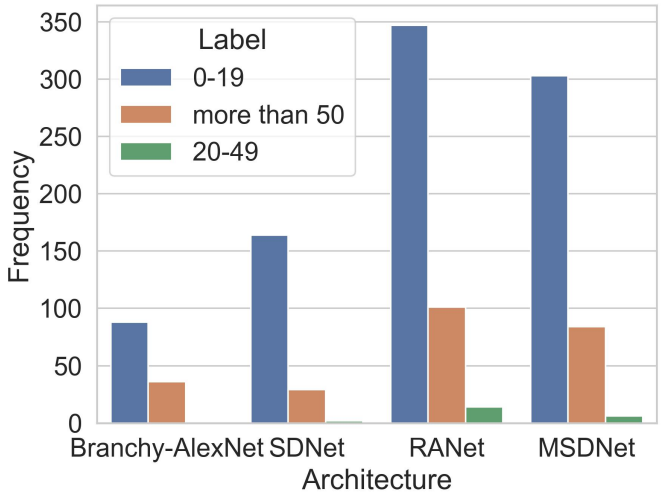

Fig. 11: Input Distribution of FAIRFACE age classes across the first time cluster of Branchy-AlexNet, SDNet, RANet and MSDNet

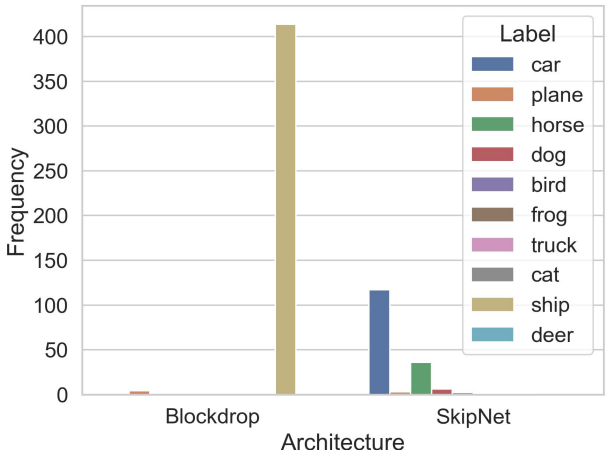

Fig. 12: Input Distribution of CIFAR10 classes across the first time cluster of BlockDrop and SkipNet

experiment with three different kinds of such settings:

1) Conservative Setting: In this setting, the thresholds of the earlier exits of the classifiers are chosen to be very strict to prioritize accuracy over inference time. The early exits are only taken when the classifiers are highly confident of the input's prediction, hence they aren't utilized much.
2) Balanced Setting: In this setting, we balance the accuracy and speed requirements of the networks. The goal here is to find a sweet spot that maximizes both accuracy and speed.
3) Relaxed Setting: In this setting, the strict requirement of the early exits thresholds is relaxed to favor speed over accuracy, and early exit classifiers are greatly utilized.

Table II provides our results. We see in the conservative setting that reducing the exit threshold for the first exit of Branchy-AlexNet on the FAIRFACE dataset increases the accuracy of the model from 74.81% to 75.03% but reduces the model performance from 45.2secs to 47.2secs. The best ASR/cluster increases from 78.26% in the balanced setting to **100**% in the conservative setting. This further corroborates our hypothesis that the model learns discriminative features particular to a specific value of the sensitive attribute at this exit which is more pronounced at stricter exit thresholds. The opposite can be observed in the case of RANet on the CANCER dataset in the relaxed setting. The highest ASR/Cluster observed drops from a value of **95.05**% in the balanced setting to 75% in the relaxed setting. This is as a result of the relaxed threshold of the earlier classifiers which allows more inputs with both possible values of the sensitive attribute to leave the network via the early exits. This relaxation of the thresholds also causes the model accuracy to drop from 87.0% to 84.13% and the model performance to increase from 49secs to 37.9secs.

d) Impact of Model Depth Hyperparameter: While dynamic networks do not rely on exit thresholds, model depth is a major hyperparameter. We experiment with different values of model depth and report the observed timing channel leakage in Table IV. We observe that the deeper the dynamic network, the more prone the network is to leak information via timing channels. For example, we consider two variants of the BlockDrop architecture trained on CIFAR10. For the BlockDrop variant built with ResNet-110 (54 blocks), there is a cluster with an ASR of **98.94**%. On the other hand, for the BlockDrop variant built with ResNet-32 (15 blocks), the largest ASR/cluster is **87.5**%. Similar results are observed for SkipNet. For SkipNet built with ResNet 110, the largest ASR/cluster is **60.98**%, while for a SkipNet built with ResNet-38 (18 blocks), the largest ASR/cluster is 39.58%.

In the case of BlockDrop, we hypothesize that this leakage behaviour is due to the policy network of the architectures. For larger networks, more unique block dropping policies, and hence computational paths, are generated (**469** in the case of the BlockDrop variant built on ResNet-110 versus only **53** in the case of the variant built on ResNet-32). Similarly, the gating network of SkipNet generates more computational paths in the case of larger networks. The greater number of computational paths potentially leads to greater partitioning of the input space, which results in higher values of ASR/cluster.

In general, searching for model hyperparameters such as exit thresholds, model depths, etc that satisfy privacy requirements while optimizing performance and accuracy for this types of ADNNs is a huge search problem given the size of the set of possible values. Further exploration of this challenge is left to future work.

e) Learning Adversarial Input: Adversarial machine learning has been a subject of study for quite some time now, and can be generally grouped into data poisoning, byzantine attacks, model evasion attacks and model extraction attacks. In our case, we consider the case of model evasion attacks, using the BlockDrop and SkipNet architectures trained on CIFAR10 dataset. We investigated if we could observe a timing difference between adversarial inputs and non-adversarial inputs. Our intuition here is that more of the adversarial inputs would use a larger number of blocks and hence have longer

Dataset	Architecture	Attribute	Exit Thresholds	Setting	Accurracy	Performance (secs)	No Clusters	ASR/Cluster	Input Distribution/Cluster	RandGA	ASR
FAIRFACE	Branchy-AlexNet	Class Label	[2.0e-03, 5.0e-02]	Conservative	75.03	47.2	3	[**100.0**, 57.63, 38.78]	[0.72, 15.28, 84.0]	33.33	42.59
FAIRFACE	Branchy-AlexNet	Class Label	[2.0e-02, 5.0e-02]	Balanced	74.81	45.2	3	[**78.26**, 64.52, 41.46]	[4.20, 11.08, 84.72]	33.33	45.71
FAIRFACE	Branchy-AlexNet	Class Label	[5.0e-01, 5.0e-01]	Relaxed	73.33	34.9	3	[36.84, 46.72, 42.03]	[45.67, 21.64, 32.70]	33.33	40.59
FAIRFACE	SDNet	Class Label	[0.99, 0.99]	Conservative	78.64	50.8	5	[**90.91**, 58.82, 57.47, 46.73, 43.77]	[1.47, 4.94, 11.44, 18.64, 63.5]	33.33	47.69
FAIRFACE	SDNet	Class Label	[0.95, 0.95]	Balanced	78.11	46.6	5	[**94.29**, 42.0, 36.04 36.88, 43.3]	[5.42, 13.06, 17.75, 22.19, 41.58]	33.33	43.21
FAIRFACE	SDNet	Class Label	[0.8, 0.8]	Relaxed	77.11	39.9	5	[56.49, 42.31, 40.69, 44.55, 56.63]	[21.67, 24.83, 22.83, 17.69, 12.97]	33.33	47.53

TABLE II: Early Exit Network Hyperparameter on FAIRFACE Dataset

Dataset	Architecture	Attribute	Exit Thresholds	Setting	Accurracy	Performance (secs)	No Clusters	ASR/Cluster	Input Distribution/Cluster	RandGA	ASR
CANCER	RANet	Class Label	[9.999942e-01, 9.9999730e-01]	Conservative	87.16	55.8	3	[**100.0**, 69.46, 70.18]	[0.5, 41.5, 58.00]	50	69.91
CANCER	RANet	Class Label	[9.9942e-01, 9.9773e-01	Balanced	87.0	49	3	[**95.05**, 62.07, 66.39]	[23.35, 21.55, 55.09]	50	68.06
CANCER	RANet	Class Label	[9.0642e-01, 9.0330e-01]	Relaxed	84.13	37.9	3	[65.11, **75.0**, 67.36]	[51.42, 1.42, 47.16]	50	66.2
CANCER	MSDNet	Class Label	[9.99642e-01, 9.99330e-01]	Conservative	87.0	48.6	3	[**100.0**, **77.42**, 69.47]	[3.67, 32.42, 63.92]	50	73.38
CANCER	MSDNet	Class Label	[9.9642e-01, 9.9330e-01]	Balanced	86.83	46.5	2	[**89.32**, 61.4]	[20.83, 79.17]	50	68.06
CANCER	MSDNet	Class Label	[9.0642e-01, 9.0330e-0]	Relaxed	83.33	37.1	2	[66.96, 61.54]	[49.33, 50.66]	50	64.35

TABLE III: Model Cascade Network Hyperparameter on CANCER Dataset

Architecture	Arch Depth	No of Blocks	Dataset	Attribute	No Clusters	ASR/Cluster	Input Distribution/Cluster	Random Guess	ASR
BlockDrop	ResNet 110	54	CIFAR10	Class Label	2	[**98.94**, 21.57]	[4.80, 95.20]	10	25.93
SkipNet	ResNet 110	54	CIFAR10	Class Label	5	[0, **60.98**, 16.57, 16.51, 0]	[0.01, 1.66, 55.67, 42.64, 0.01]	10	17.56
BlockDrop	ResNet 32	15	CIFAR10	Class Label	2	[**87.5**, 10.32]	[0.27, 99.72]	10	10.67
SkipNet	ResNet 38	18	CIFAR10	Class Label	6	[39.58, 21.57, 12.13, 11.74, 15.62, 23.64]	[2.57, 15.01, 26.20, 27.54, 21.80, 6.88]	10	15.83

TABLE IV: Dynamic Network Hyperparameter on the CIFAR10 Dataset

inference times compared to the non-adversarial inputs, leading to execution time correlated with whether or not an image is adversarial. To generate our adversarial examples for this experiments, we use the Fast Gradient Sign Attack (FGSM) method described by Goodfellow et.al [16]. This method involves adding imperceptible perturbations to an image, which in turns causes the machine learning model to misclassify it. The results from this experiments shown in Table I confirms that there exists time partitions for which we can distinguish between adversarial and non-adversarial inputs fed to the BlockDrop model trained on CIFAR10 dataset.

VII. Case Studies

To further exemplify the real-world applicability of exploiting timing channels in ADNNs, we perform two case studies on two different web applications that use different variants of adaptive neural networks on their backend. We solely focus on ADNNs that exhibit high ASR/cluster on the LAN when considering a specific dataset for these case studies.

A. Case Study Setup

For our case studies, we ran our experiments over the public internet. We hosted the Fictitious Health Company and Fictitious HR web application on a Google Cloud server located in the USA eastern region, which according to Google Cloud documentation [9] resolves to Moncks Corner, South Carolina, North America, while the client was located in our lab in Hoboken, New Jersey USA. The distance between the server hosted on Google Cloud and our remote client is approximately 700 miles. According to TRACEROUTE, the route comprises 7 hops. Round-trip time and noise vary depending on load, but typical RTT was around 13.1 msec (min 10.5, mean 13.12, max 14.8, std 1.1).

We first create our attack model using the method described in Section IV, then we had a fake user residing on the same network as us submit a query to the web application, we collected the timing results and proceeded to infer the sensitive attribute of the user's input.

B. Fictitious Health Company

Fictitious Health Company is a toy technology company we designed and implemented that utilizes ADNNs to provide accurate and timely skin cancer predictions to clients based on uploaded images of skin moles. It provides a user-friendly web application accessible over the public internet that allows healthcare professionals to upload images of skin moles and receive predictions immediately, helping the health care personnel to make faster treatment decisions. The assumptions in this case study are those enumerated in Section IV-B. Most importantly, we assume the adversary is on the same network as the victim.

Our results are given in Table V. We observe that for all architectures, some information is leaked about whether or not the input image is of a malignant skin mole. For three of the architectures, an ASR/cluster of over **80**% is observed (versus a baseline of 50%). In the case of RANet, almost a quarter of input images fall into a cluster with an ASR of **86.49**%. We observe lower ASR/Cluster for SDNet and SkipNet though both still show an overall ASR of over **60**%, meaning that the adversary has still improved their ability to infer the sensitive attribute with timing information.

Architecture	Dataset	Attribute	No Clusters	ASR/Cluster	Cluster Input distribution	RandGA	ASR
Branchy-AlexNet	CANCER	Class label	3	[**84.88**, 59.43, 70.18]	[26.56, 54.30, 19.14]	50	68.24
SDNet	CANCER	Class label	2	[67.0, 62.45]	[46.79, 53.21]	50	64.58
RANet	CANCER	Class label	2	[**86.49**, 62.62]	[23.46, 76.54]	50	68.75
MSDNet	CANCER	Class label	2	[**83.0**, 58.13]	[20.46, 79.54]	50	63.89
SkipNet	CANCER	Class label	5	[**75.0**, 54.24, 66.06, 0, 0]	[10.04, 63.5, 26.21, 0.17, 0.08]	50	59.72

TABLE V: Fictitious Health Company

C. Fictitious HR

Fictitious HR is a toy technology company we designed and implemented that utilizes ADNNs to provide accurate age estimation to its clients based on uploaded images. The company's objective is to revolutionize the hiring process by incorporating AI technology to assist organizations in assessing a candidate's age, ensuring fair and unbiased decision-making. It provides a user-friendly web application secured over the public internet that allows HR professionals to upload candidate images and receive age estimations. Again, our assumptions in this case study are those given in Section IV-B. Most importantly, we assume the adversary is on the same network as the victim, a job applicant uploading their image to the Fictitious HR web application as instructed by the hiring HR personnel.

Our results are given in Table VI. Again, we observe high ASR/Cluster for at least one cluster for each architecture, improving the accuracy from 33% to over **70**% in all cases. We observe very strong leakage **92.86**% for one cluster of SDNET, though only 3.48% of the submitted images fall into this cluster.

D. Additional Internet Experiments

We also perform experiments using the CIFAR10 dataset across the public internet. We report our results in Table VII, where we still find clusters with significantly high ASR, 76.09% in the case of BlockDrop and 60.0% in the case of SkipNet.

E. Impact of Noisy Observations

In our experiments across the public internet, we acknowledge the impact of noise factors such as network congestion, packet loss, and varying network conditions. These noise factors introduce unpredictability and inconsistency in network latency, posing challenges in accurately measuring and interpreting timing variations. To mitigate the effect of this noise on out timing profile, we employ several strategies. Firstly, we run each input of our manifest dataset ten times and record the average timing measurements. By collecting multiple samples, we aim to smooth out the impact of random fluctuations caused by noise, providing a more reliable estimate of timing variations. Additionally, we apply statistical techniques to identify and remove outlier measurements. This helps us eliminate any extreme values that may be influenced by noise, ensuring that our analysis focuses on more representative timing data.

Despite these efforts, we suffer a decrease in performance of our highest ASR for a given cluster when compared to measurements taken over a local area network environment due to the inherent noise and variability of the public internet. However, by accounting for these challenges and employing appropriate noise mitigation techniques, we strive to minimize its impact and draw reliable conclusions from our experiments.

VIII. Limitations

A. Quality of the Manifest Dataset $DS_{\mathcal{M}}$

The ability of an adversary to generate an accurate timing profile depends on the quality of the manifest dataset. The more divergence between the input distribution and the manifest dataset, the less reliable our attack model would be in inferring the sensitive attributes of unseen inputs, given their timing observations.

B. Normal Distribution of Timing Observations

Given our utilization of the KDE algorithm for clustering timing observations, it becomes crucial for the timing observations to approximate a normal distribution. This is because the KDE algorithm relies on a continuous probability density function for clustering. If our timing observations deviate significantly from the normal distribution, it may be advisable to consider an alternative clustering algorithm.

C. Impact of the KDE's algorithm bandwidth parameter

The effectiveness of KDE heavily relies on selecting an appropriate kernel bandwidth parameter as an improper selection may lead to oversmoothing or undersmoothing, resulting in inaccurate density estimates, misleading clustering outcomes, and a reduction in our attack model's performance. Hence it might be worthwhile to explore other bandwidth values, but given that this is a search space problem, we leave this for future work.

D. Noise over LAN and Public Internet

Considering the influence of noise in our experiments conducted over the public internet, we postulate that as the level of noise amplifies due to factors such as network instability, network congestion, or increased distance between the client and server, the reliability of our timing measurements and observations will diminish. This decrease in reliability directly affects the capability of our attack model to establish correlations between timing observations and the sensitive attributes of the input, both across the LAN (Local Area Network) and over the public internet. Additionally, this noise could also affect the quality of the timing observations collected for the target victim's input further decreasing the performance of our attack model.

Architecture	Dataset	Attribute	No Clusters	ASR/Cluster	Cluster Input distribution	RandGA	ASR
Branchy-AlexNet	FAIRFACE	Class label	4	[**71.43**, 34.22, 29.51, 42.31]	[2.59, 41.81, 33.09, 22.52]	33.33	35.51
SDNet	FAIRFACE	Class label	6	[**92.86**, 51.26, 34.09, 42.48, 45.2, 52.31]	[3.48, 17.53, 15.78, 24.87, 27.70, 10.64]	33.33	45.94
RANet	FAIRFACE	Class label	5	[**72.94**, 37.37, 35.45, 48.44, 52.94]	[12.47, 58.22, 9.75, 9.75, 2.83]	33.33	43.21
MSDNet	FAIRFACE	Class label	2	[**73.21**, 39.92]	[11.60, 88.40]	33.33	43.45

TABLE VI: Fictitious HR

Architecture	Dataset	Attribute	No Clusters	ASR/Cluster	Cluster Input distribution	RandGA	ASR
BlockDrop	CIFAR10	Class label	6	[27.27, **76.09**, 40.26, 11.15, 0, 0]	[0.61, 5.11, 7.18, 87.08, 0.01, 0.01]	10	17.06
SkipNet	CIFAR10	Class label	6	[0, **60.0**, 20.87, 16.28, 19.02, 0]	[0.01, 1.5, 47.76, 12.22, 38.50, 0.01]	10	20.39

TABLE VII: Additional Results over the Internet

IX. Related Work

a) Privacy in Adaptive Neural Networks: The work of Zheng Li et. al aims to investigate how multi-exit networks affect membership leakages [29]. While this work mainly concerns itself with the vulnerability of multi-exit networks to membership inference attacks, our work focuses on how timing channels in ADNNs may leak information about sensitive attributes of user's input to the model.

b) Side Channels in Machine Learning Frameworks: Existing research aims at studying security vulnerabilities in machine learning frameworks, including side-channel vulnerabilities. These attempts can be categorized along multiple axes: what type of side-channel observable is considered, what kind of information is leaked, and what attack scenario is considered. Previous research has demonstrated side channels due to micro-architectural events (generally cache hits/misses) [19], [60], [26], [1], [50], power consumption [55], [54], [4], memory [38], GPU [53], [33], electromagnetic emanations [5], [64], and, as in our work, execution time [15], [10], [34], [45]. A significant fraction of the above work has focused on side-channel attacks to reverse engineer proprietary machine learning models. Such attacks use side-channel information, such as cache hits and misses, to infer model parameters. Another line of inquiry more closely our work is using side-channel vulnerabilities to learn properties about user input. The class label is the most commonly targeted property though the work of Wei et al [54] uses a power side channel to recover the input image itself. Other work by Nakai et al [34] uses timing channels to craft adversarial examples. While some previous work considers an attack scenario similar to ours where the attacker has black-box or API-only access to the model, others consider the case where the adversary either has full or partial knowledge of the model internals. Measuring side-channel observables such as electromagnetic emanations or power consumption also requires physical access to the model. A characterization of related work along these axes can be found in Figure 13.

c) Automated Side-Channel Analysis: Extensive research has been devoted to the study of side-channel vulnerabilities in software and hardware. Both static [2] and dynamic [65], [40], [37] techniques to quantify software side-channel vulnerabilities using entropy-based metrics have been proposed; however, none of these metrics explicitly formulate attributes of the input domain that might be considered sensitive. This is likely partly due to differences in the size of the domain of secret input compared to the domain in our work. It also might arise as a result of how side channels in software tend to partition the secret input: according to branch conditions in the source code relating to predicates over that input [7]. As a result, side channels in software tend to partition the input space according to traits, removing the need to explicitly check the partitioning. Fuzzing is another widely used technique for side-channel analysis [36], [8], [3], [59], [37], usually aimed at efficiently determining if side-channel vulnerabilities are present in software. In future work, we plan to explore whether fuzzing might offer some benefits for efficiently exploring the space of hyperparameters to find values where no sensitive information is leaked.

Work	Side-Channel Observable	Objective
[1], [19], [26], [50]#, [60]	Micro-architectural events	○ □ □ ○ □
[4]*, [54]*, [55]*	Power consumption	□ ○ □
[38]#	Memory	□
[33], [53]	GPU	□ □
[5]*, [64]*	Electromagnetic emanations	□ □
[10], [15], [34], [45]	Time	□ □ △ ○

□ = Reverse engineer model ○ = Learn sensitive input information △ = Craft adversarial examples
* = Physical access required
# = Not black box

Fig. 13: Summary of Side-Channel Vulnerabilities in Deep Learning Models

X. Conclusions and Future Work

We have presented the first demonstration of information leakage via timing channels in ADNNs. We demonstrated how sensitive attributes of user inputs can be recovered through timing information across six different adaptive neural network architectures and across multiple datasets. Critically, this information can be learnt by an adversary with only black-box access to the model who is taking timing measurements across a network. The timing channels we observe are large enough in magnitude that different partitions can be observed even in the presence of noise. In future work, we intend to leverage our observation that model parameters such as exit

thresholds greatly impact the vulnerability of these networks to timing channel attacks by introducing an automated fuzzing-based approach to exploring the space of network parameters to generate models that are robust to timing channels, accurate and efficient. Because runtime factors such as caching can impact timing measurements, we also intend to introduce an online monitoring system to detect when timing channels have arisen in deployed models and introduce mitigation measures in the case of such an event. Some of the mitigation measures we plan to explore include; the impact of introducing random delays during model inference, and the effectiveness of removing block dropping or skipping policies that are linked to privacy violations.

AVAILABILITY

We share all the code and data needed to generate our results via https://github.com/akinsanyaayomide/ADNNTimeLeaks.git and also on Zenodo via https://zenodo.org/records/10164348

REFERENCES

[1] Manaar Alam and Debdeep Mukhopadhyay. How secure are deep learning algorithms from side-channel based reverse engineering? In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–2, 2019.

[2] Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S Păsăreanu, and Tevfik Bultan. String analysis for side channels with segmented oracles. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 193–204, 2016.

[3] Tiyash Basu, Kartik Aggarwal, Chundong Wang, and Sudipta Chattopadhyay. An exploration of effective fuzzing for side-channel cache leakage. *Software Testing, Verification and Reliability*, 30(1):e1718, 2020.

[4] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. Csi neural network: Using side-channels to recover your artificial neural network information. *arXiv preprint arXiv:1810.09076*, 2018.

[5] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, 2019.

[6] Benjamin Biggs. early-exit network(s). https://github.com/biggsbenjamin/earlyexitnet, 2022.

[7] Tegan Brennan, Nicolás Rosner, and Tevfik Bultan. Jit leaks: Inducing timing side channels through just-in-time compilation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1207–1222. IEEE, 2020.

[8] Tegan Brennan, Seemanta Saha, and Tevfik Bultan. Jvm fuzzing for jit-induced side-channel detection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1011–1023, 2020.

[9] Google Cloud. Regions and zones. https://cloud.google.com/compute/docs/regions-zones, 2023.

[10] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*, 2018.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[12] Claudio Fanconi. Skin Cancer: Malignant vs. Benign. https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign, 2019.

[13] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 84–95. IEEE, 2020.

[14] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 115–127, 2018.

[15] Cheng Gongye, Yunsi Fei, and Thomas Wahl. Reverse-engineering deep neural networks using floating-point timing side-channels. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[19] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv preprint arXiv:1810.03487*, 2018.

[20] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[21] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. msdnet. https://github.com/kalviny/MSDNet-PyTorch/tree/master, 2019.

[22] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. sdn. https://github.com/yigitcankaya/Shallow-Deep-Networks, 2019.

[23] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019.

[24] Alexandros Kouris, Stylianos I Venieris, and Christos-Savvas Bouganis. A throughput-latency co-optimised cascade of convolutional neural network classifiers. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1656–1661. IEEE, 2020.

[25] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[26] Bhargav Achary Dandpati Kumar, Sai Chandra Teja R, Sparsh Mittal, Biswabandan Panda, and C Krishna Mohan. Inferring dnn layer-types through a hardware performance counters based side channel attack. In *The First International Conference on AI-ML-Systems*, pages 1–7, 2021.

[27] Kimmo Kärkkäinen and Jungseock Joo. Fairface: Face attribute dataset for balanced race, gender, and age. *arXiv preprint arXiv:1908.04913*, 2019.

[28] Royson Lee, Stylianos I Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D Lane. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *The 25th annual international conference on mobile computing and networking*, pages 1–16, 2019.

[29] Zheng Li, Yiyong Liu, Xinlei He, Ning Yu, Michael Backes, and Yang Zhang. Auditing membership leakages of multi-exit networks, 2022.

[30] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[31] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[32] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018.

[33] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2139–2153, 2018.

[34] Tsunato Nakai, Daisuke Suzuki, and Takeshi Fujino. Timing black-box attacks: Crafting adversarial examples through timing leaks against dnns

on embedded devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 149–175, 2021.

[35] Subhas C. Nandy, Sandip Das, and Partha P. Goswami. An efficient k nearest neighbors searching algorithm for a query line. *Theoretical Computer Science*, 299(1):273–288, 2003.

[36] Shirin Nilizadeh, Yannic Noller, and Corina S Pasareanu. Diffuzz: differential fuzzing for side-channel analysis. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 176–187. IEEE, 2019.

[37] Yannic Noller and Saeid Tizpaz-Niari. Qfuzz: quantitative fuzzing for side channels. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 257–269, 2021.

[38] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174. IEEE, 2022.

[39] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956.

[40] Nicolás Rosner, Ismet Burak Kadron, Lucas Bang, and Tevfik Bultan. Profit: Detecting and quantifying side channels in networked applications. In *NDSS*, 2019.

[41] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[42] Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. The right tool for the job: Matching model and instance complexities. *arXiv preprint arXiv:2004.07453*, 2020.

[43] Jianghao Shen, Yue Wang, Pengfei Xu, Yonggan Fu, Zhangyang Wang, and Yingyan Lin. Fractional skipping: Towards finer-grained dynamic cnn inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5700–5708, 2020.

[44] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[45] Shubhi Shukla, Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Pabitra Mitra. On the evaluation of user privacy in deep neural networks using timing side channel. *arXiv preprint arXiv:2208.01113*, 2022.

[46] Luca Soldaini and Alessandro Moschitti. The cascade transformer: an application for efficient answer sentence selection. *arXiv preprint arXiv:2005.02534*, 2020.

[47] Ben Taylor, Vicent Sanz Marco, Willy Wolff, Yehia Elkhatib, and Zheng Wang. Adaptive deep learning model selection on embedded systems. *ACM SIGPLAN Notices*, 53(6):31–43, 2018.

[48] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[49] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[50] Han Wang, Syed Mahbub Hafiz, Kartik Patwari, Chen-Nee Chuah, Zubair Shafiq, and Houman Homayoun. Stealthy inference attack on dnn via cache-based side-channel attacks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1515–1520. IEEE, 2022.

[51] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. skipnet. https://github.com/ucbdrive/skipnet, 2018.

[52] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[53] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, 2020.

[54] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[55] Shaya Wolf, Hui Hu, Rafer Cooley, and Mike Borowczak. Stealing machine learning parameters via side channel power attacks. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 242–247. IEEE, 2021.

[56] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018.

[57] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. blockdrop. https://github.com/Tushar-N/blockdrop, 2022.

[58] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*, 2020.

[59] Fei Yan, Rushan Wu, Liqiang Zhang, and Yue Cao. Spider: Speeding up side-channel vulnerability detection via test suite reduction. *Tsinghua Science and Technology*, 28(1):47–58, 2022.

[60] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[61] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2369–2378, 2020.

[62] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Ranet-pytorch. https://github.com/yangle15/RANet-pytorch, 2022.

[63] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

[64] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 209–218. IEEE, 2020.

[65] Kehuan Zhang, Zhou Li, Rui Wang, XiaoFeng Wang, and Shuo Chen. Sidebuster: automated detection and quantification of side-channel leaks in web application development. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 595–606, 2010.

[66] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

APPENDIX A
ARTIFACT APPENDIX

A. Description & Requirements

1) How to access: Our artifact can be found via https://github.com/akinsanyaayomide/ADNNTimeLeaks/tree/main and also on Zenodo via https://zenodo.org/records/10164348

2) Hardware dependencies: If both client and server side are simulated on a single machine then all other tabs or background activities on the machine should be suspended prior to running the experiment for the most accurate results.

3) Software dependencies: Any commodity machine that can run Python 3.8.

4) Benchmarks: The Datasets required for using this artifact are the following CIFAR10, CIFAR100, CANCER, and FAIRFACE. The artifact provides a reliable means of obtaining all the datasets required for running the experiment.

B. Artifact Installation & Configuration

The artifact runs in a Python 3.8 environment, and the artifact contains a requirements.txt file which should be installed before running the artifact.

C. Experiment Workflow

The experiment workflow consists of two part listed below:

1) Profiling Stage: This stage describes how the client (adversary) obtains the timing profile of the target model hosted on the server side.
2) Evaluation Stage: This stage describes how the client (adversary) evaluates the obtained target model timing profile for information leakage.

D. Major Claims

- (C1): We show how timing channels in ADNNs observed over the LAN can increase an adversary's ability to infer sensitive user attribute, such as age or medical information, by up to a factor of **9.89x**. This is proven by experiment (E1) whose results are reported in Table I of our paper.

E. Evaluation

1) Experiment (E1): [LAN Experiments] [120 human-minutes + 2 compute-hour]:

[How to] Clone the artifact repository and follow the instructions in the README file.

[Preparation] You might want to setup a Python virtual environment to avoid any package conflict. You can do so by following the instructions here https://www.geeksforgeeks.org/creating-python-virtual-environment-windows-linux/

[Execution] Once the virtual environment has been setup follow the steps in the README file in the artifact repository, which consists of installing a requirements.txt file.

[Results] After generating the model timing profile in the profiling stage, follow the steps in the Evaluation stage in the README file to obtain results for each model architecture. The results obtained should be very similar to the ones presented in Table 1 of the paper, with maximum variations of $\pm 3\%$ in the cluster accuracies of clusters reported with very high accuracy (the ones in bold in the paper).

F. Customization

If running all experimental cases proves to be too laborious, example timing profiles to reproduce main results of the paper is given in the EvalTimeProfile directory in the artifact repository.