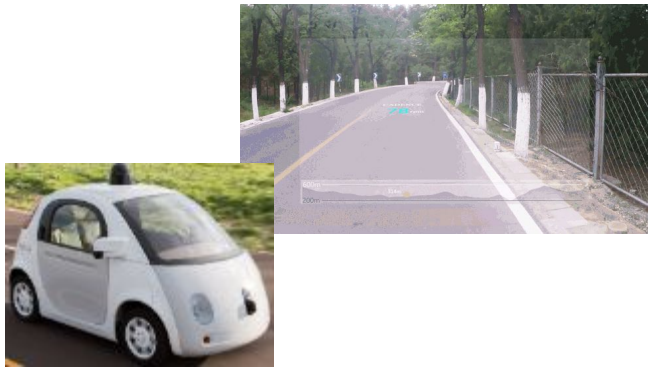
Machine Learning and Embedded Security

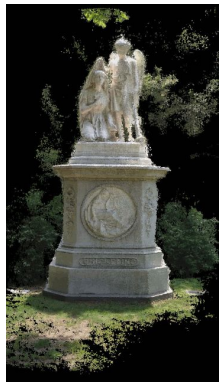
Farinaz Koushanfar
Professor and Henry Booker Faculty Scholar
Founder and Co-Director, Center for Machine-Integrated & Security (MICS)
University of California San Diego

Big data and automation revolution

Computer Vision



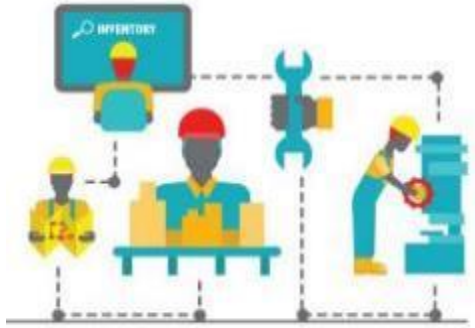
3D Reconstruction



Cyber-Physical Systems

Speech Recognition

Smart Manufacturing



Search Engines

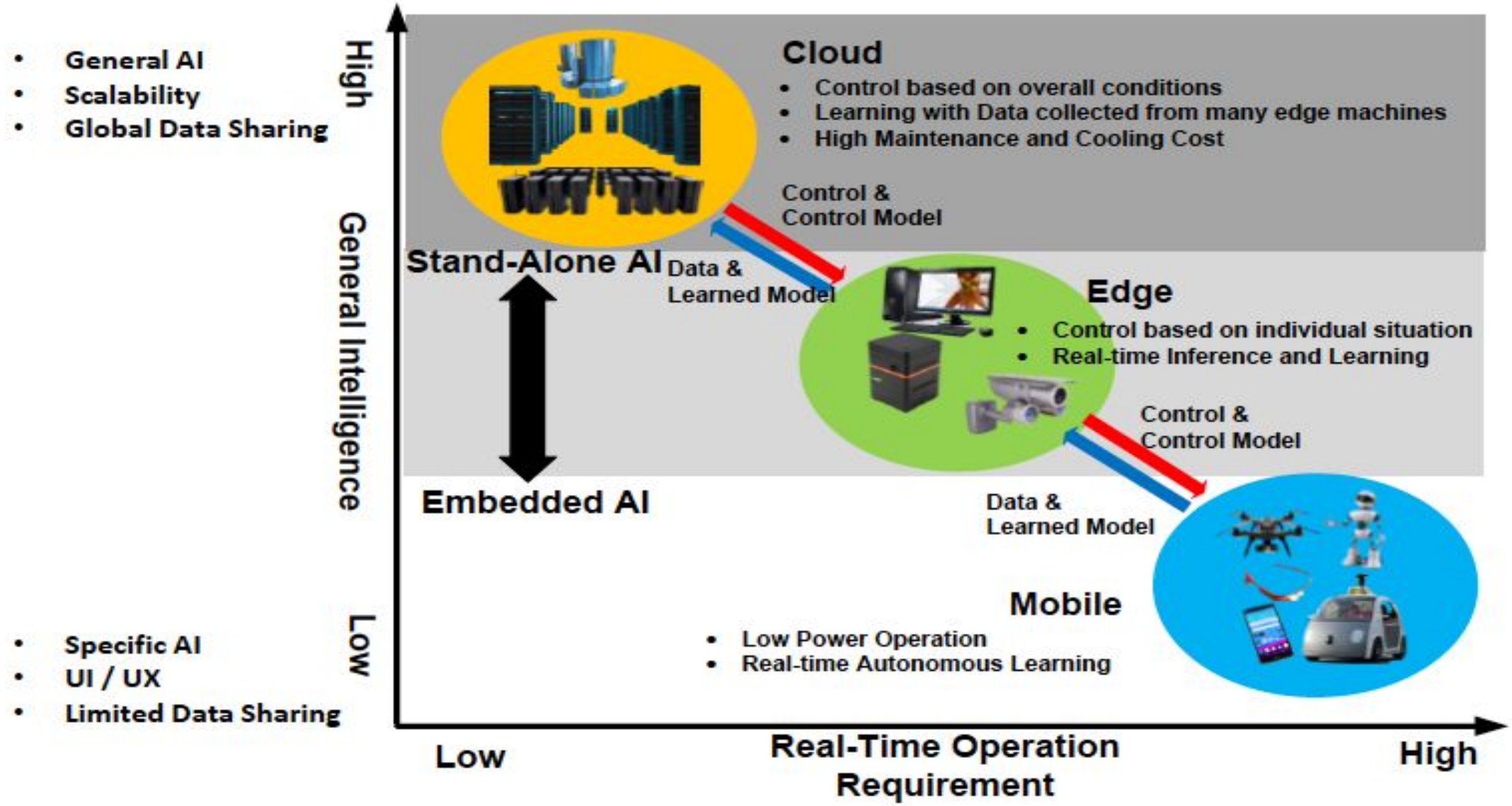
bing

Google

Microsoft

Baidu 百度

Machine learning on embedded devices



Example: Embedded vision applications

Security & Surveillance

Visual Perception & Analytics

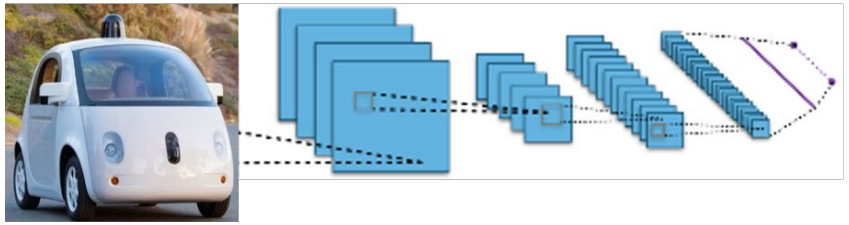
Drones

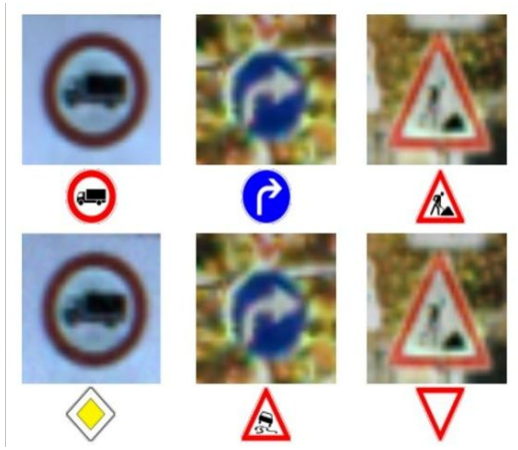
ADAS & Autonomous Cars

Augmented Reality

Reliability of ML on embedded devices

- Reliability of AI systems is one of the major obstacles for the wide-scale adoption of emerging learning algorithms in sensitive autonomous systems such as unmanned vehicles and drones
- Performance is the most widely pursued challenge now: yet to be solved!
- Some standing security challenges
  - Adversarial examples
  - IP vulnerabilities
  - Trusted execution
  - Privacy
    - Anonymity
    - Inference on encrypted data

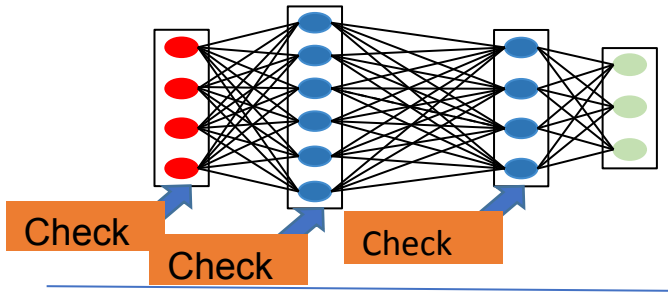




Safe embedded ML technologies in UCSD/MICS

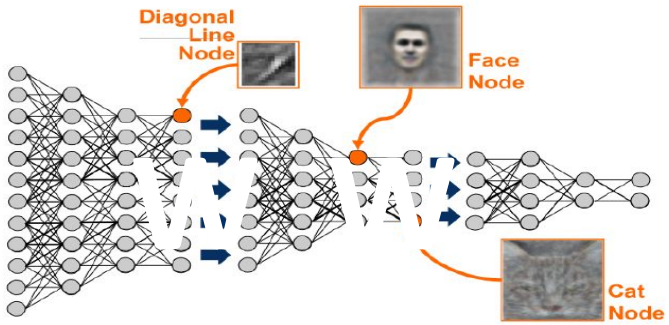
DeepFence

The first comprehensive defense
Against adversarial DL on ES



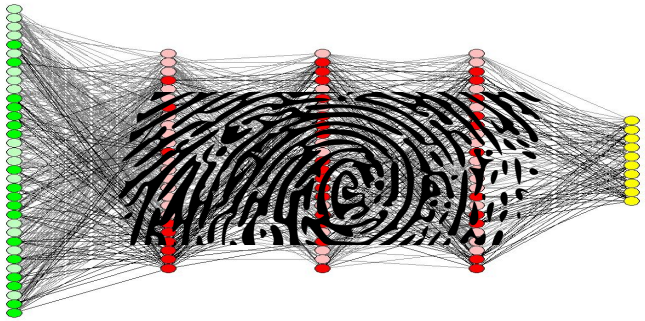
DeepMarks

The first unremovable DL watermarks



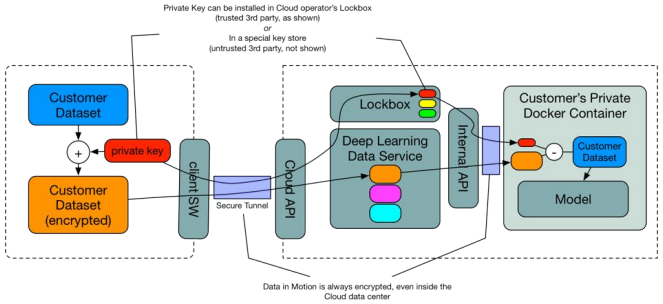
DeepSigns

The first unremovable DL fingerprints



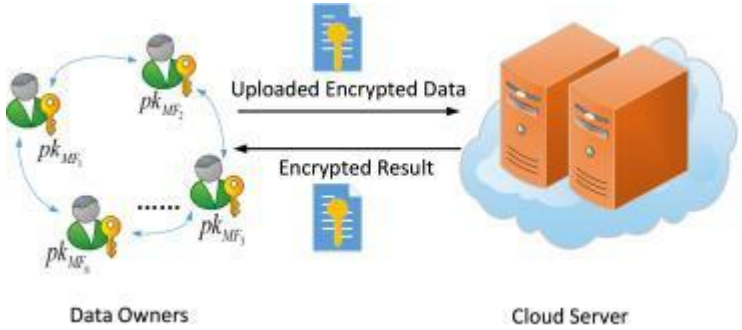
DeepIPTrust

The first hybrid trusted platform
& DL for IP protection



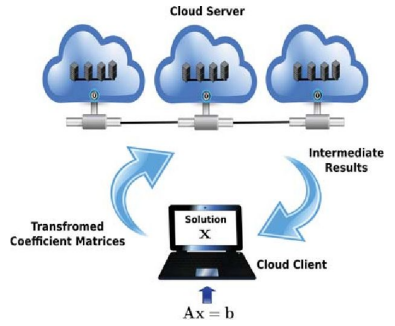
DeepSecure & Chameleon

The most efficient DL on encrypted data



Secure Federated ML

Efficient secure distribued&federated ML



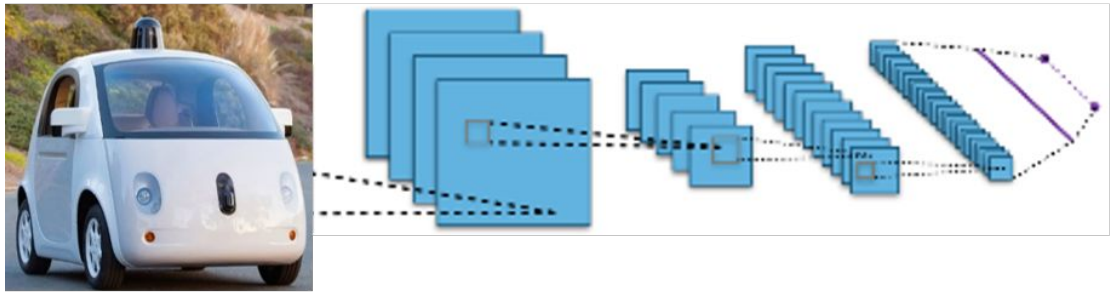
DeepFense

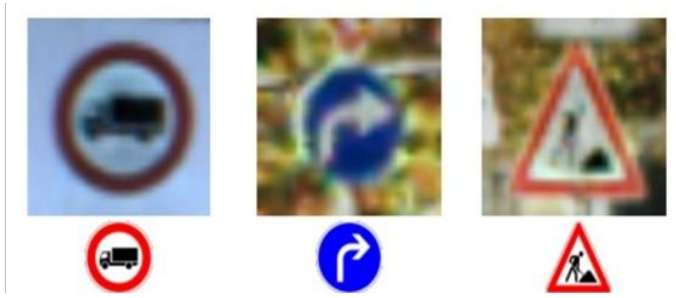
The First accelerated and automated defense against adversarial learning

Adversarial learning

Reliability is one of the major obstacles for the wide-scale adoption of emerging Deep Learning (DL) models in sensitive autonomous systems such as unmanned vehicles and drones

Consider an autonomous car which leverages a DL model to analyze front scene





DeepFense contribution

Unsupervised model assurance as well as defending against the adversaries

Model assurance by checkpointing DL models at intermediate points

- parallel models with various accuracy & robustness
- Hardware-acceleration for just-in-time response

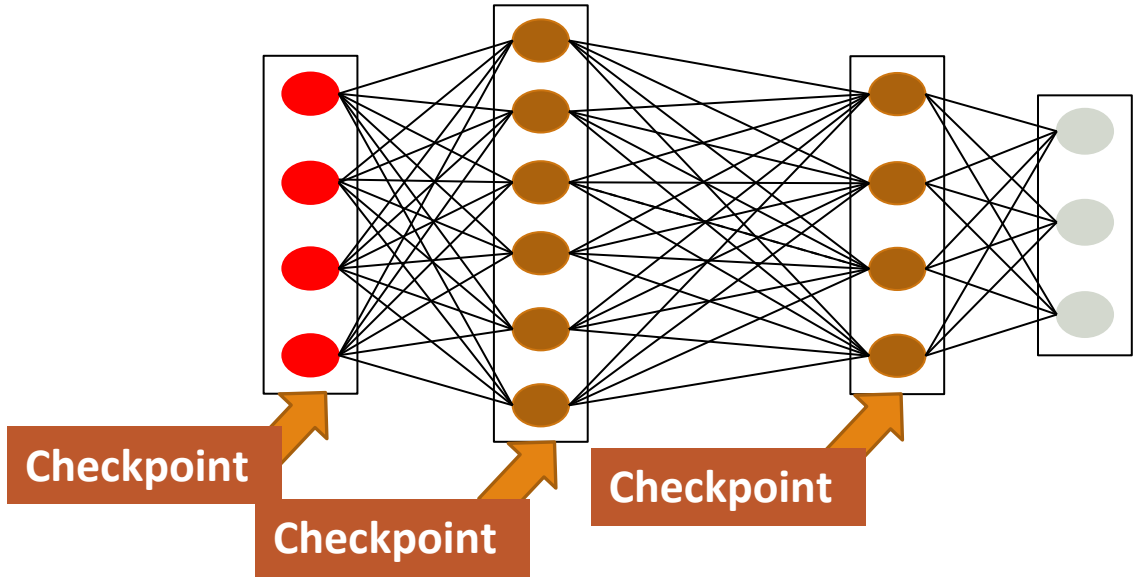
Proof-of-concept evaluation on various benchmarks and attacks

Automated accompanying API

DeepFense framework

Robustness and model accuracy are distinct objectives with a trade-off

We checkpoint the intermediate variables to find atypical samples

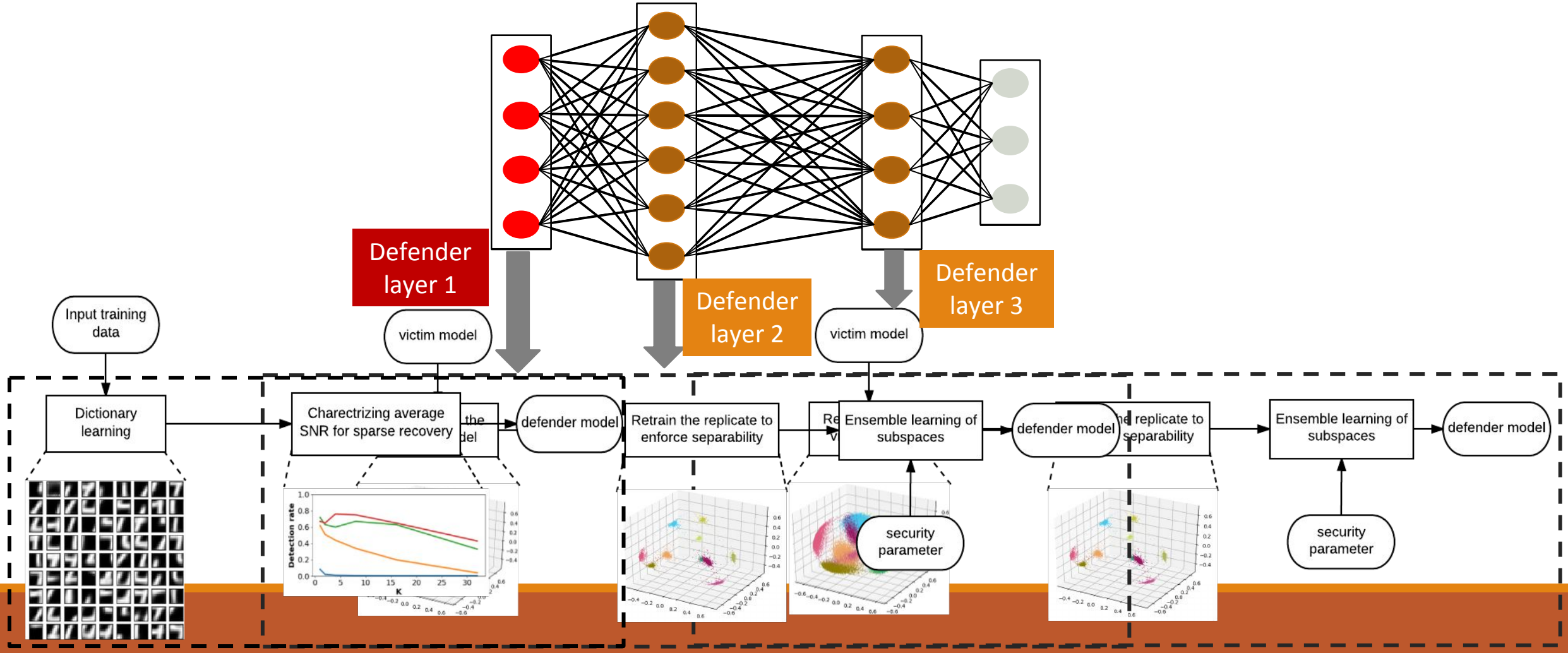


With the proposed defense methodology:

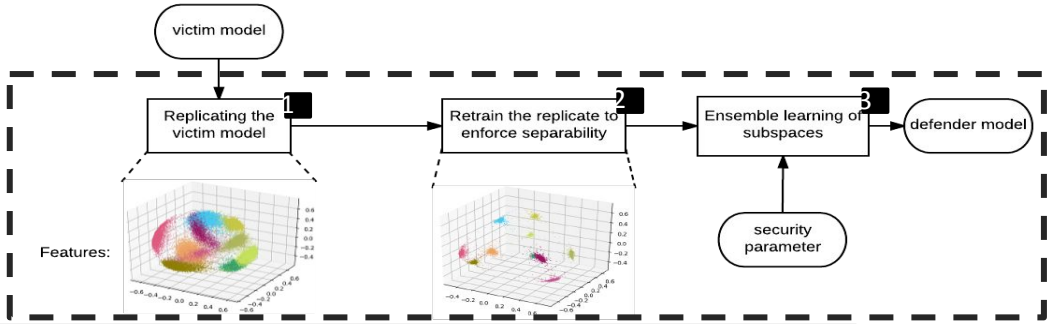
- The victim model is not altered
- The accuracy is not dropped
- The adversary would require to deceive all defenders to success

Global flow


victim model
Input sample
Victim model
Prediction
Defender modules
Is this prediction reliable?


Defender layer 1
Defender layer 2
Defender layer 3
Input training data
victim model
victim model
Dictionary learning
Charectrizing average SNR for sparse recovery
defender model
Retrain the replicate to enforce separability
Ensemble learning of subspaces
defender model
Ensemble learning of subspaces
defender model
security parameter
security parameter
Detection rate
K

Training latent defender



Training each redundancy module involves three main steps:

1 Replicating the victim neural network

2 Fine-tuning the replicated network with a modified loss function[1] to **condense** and **separate** the data features in the pertinent space

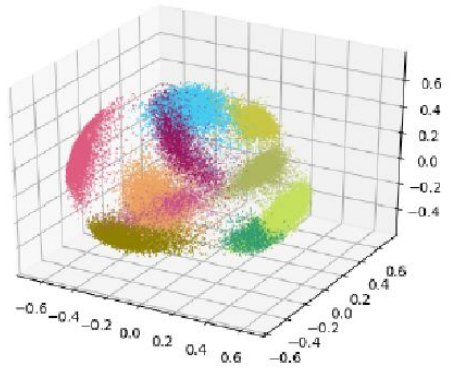
$$L = L_{victim} + L_{clustering}$$

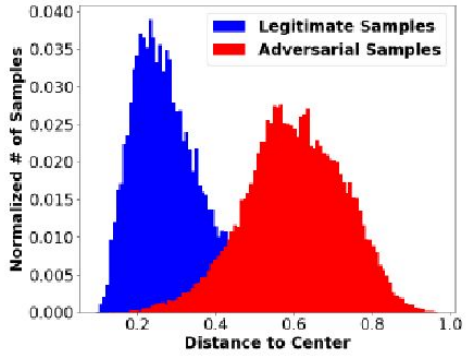
3 Learning the PDF of the data abstractions as a union of explored subspaces to be used later for validating the legitimacy of test samples

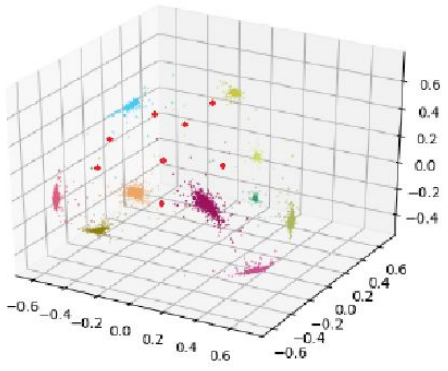
[1] Bita Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. “DeepFense: Online Accelerated Defense Against Adversarial Deep Learning”, ICCAD 2018

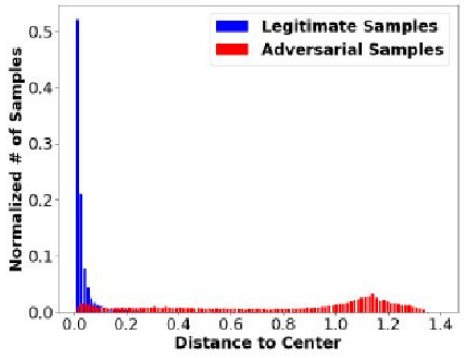
Statistical testing for detection

- Adversarial and legitimate samples differ in statistical properties
  - Even in the victim model (left), adversarial samples deviate from the PDF of legitimate samples
  - Our unsupervised defense mechanism (right) characterize the underlying space by data realignment and separation of the PDFs corresponding to adversarial and legitimate samples









Training Input defenders

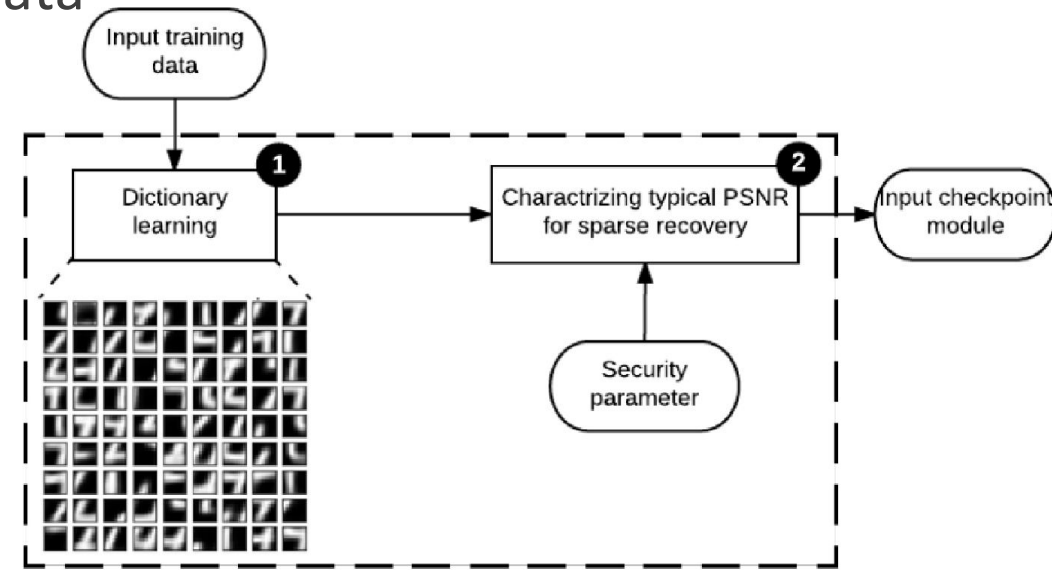
Training each input redundancy module involves two main steps:

1 Dictionary learning

- Learning separate dictionaries for each class of data

2 Characterizing typical PSNR in each category

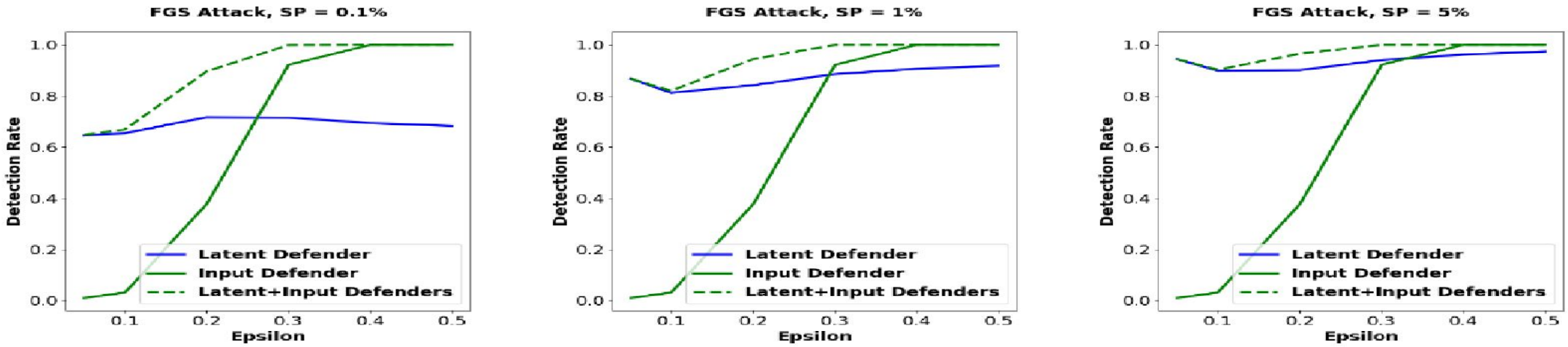
- Profiling PSNR of legitimate samples in each class



[1] Bita Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. “DeepFense: Online Accelerated Defense Against Adversarial Deep Learning”, ICCAD 2018

Input and latent defenders

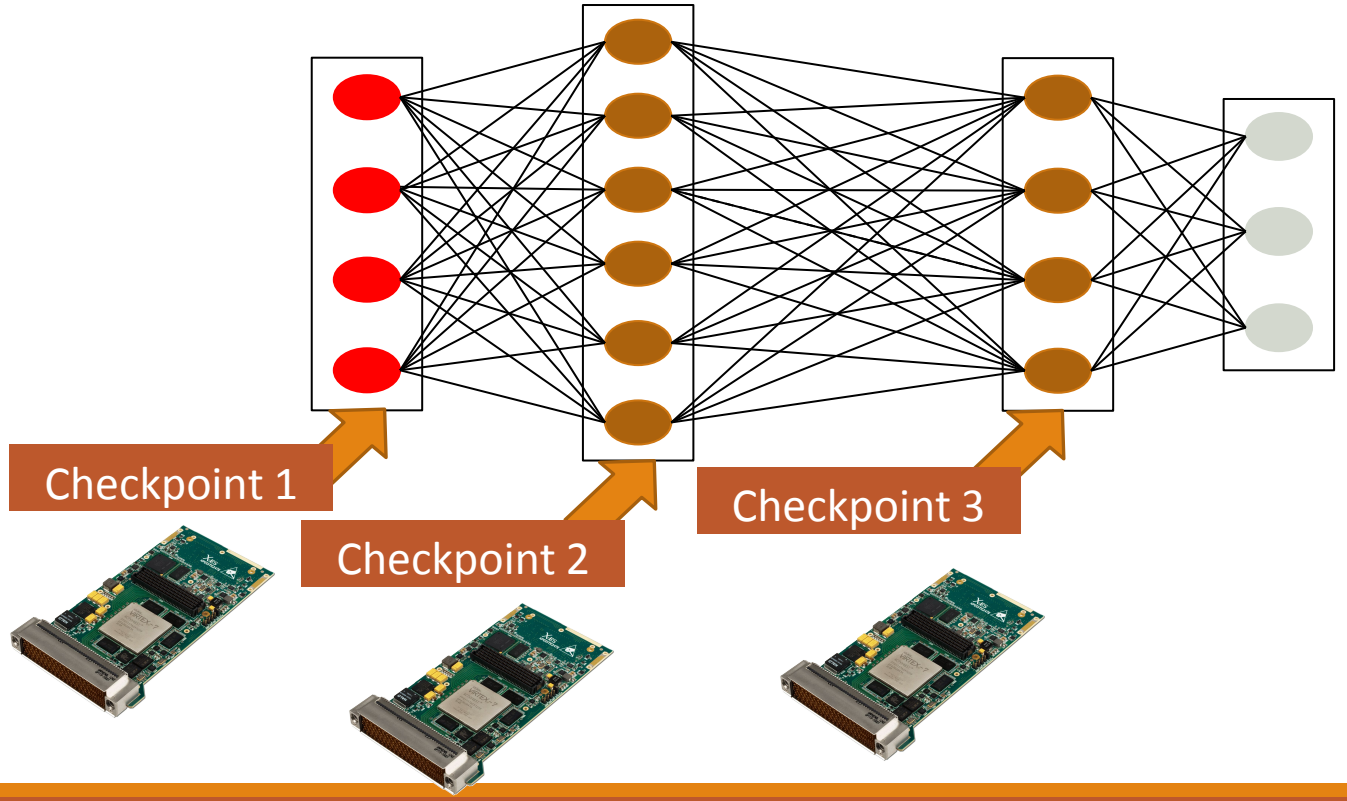
The impact of perturbation level on the pertinent adversarial detection rate for three different security parameters (cut-off thresholds) on MNIST benchmark



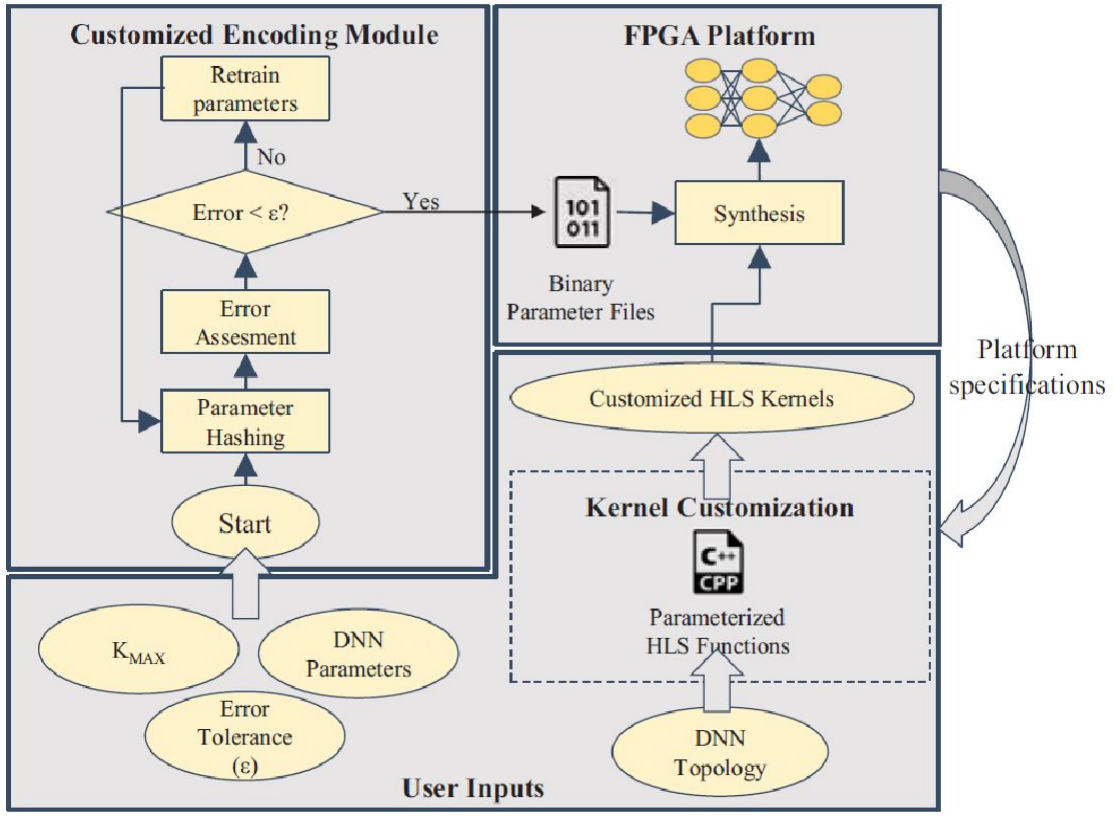
The use of **input dictionaries** facilitate **automated detection** of adversarial samples with relatively **high perturbation** (e.g., $\varepsilon > 0.25$) while the **latent defender module** is sufficient to distinguish malicious samples even with **very small perturbations**

Hardware acceleration for DeepFence

- Reducing runtime overhead by parallel execution of defender modules on FPGA



Hardware/Software co-optimized acceleration (e.g.,[1,2])



[1] Mohammad Samragh, Mohsen Imani, Farinaz Koushanfar, Tajana Rosing "LookNN: Neural network with no multiplication." DATE *2017*

[2] Mohammad Samragh, Mohammad Ghasemzadeh, Farinaz Koushanfar, " Customizing neural networks for efficient FPGA implementation" FCCM 2017

Automation and API

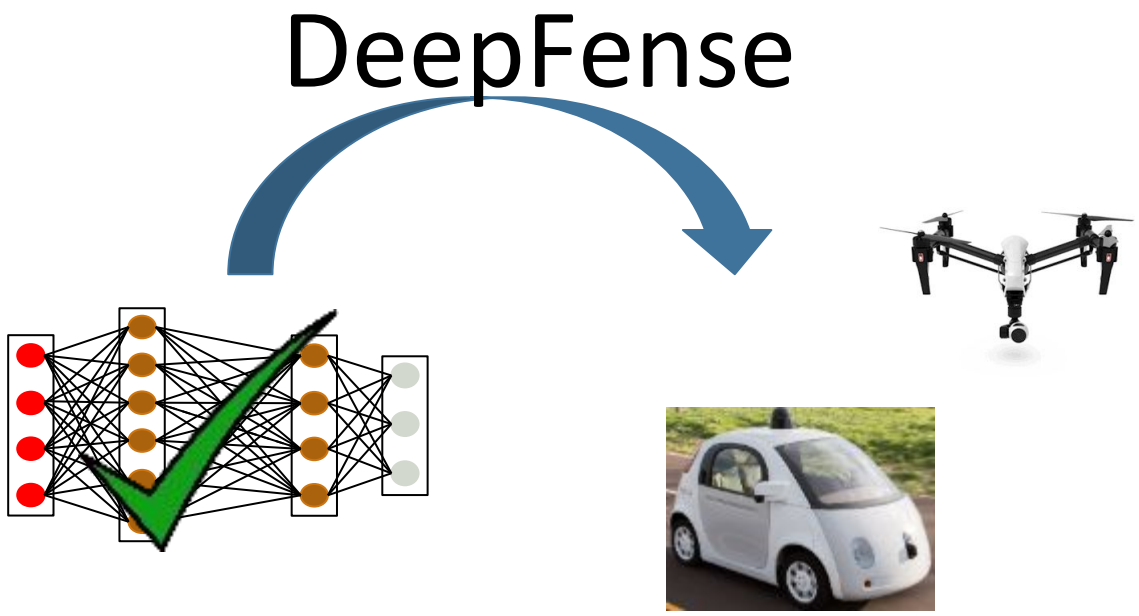
We provide automated APIs for training input & latent defender modules

- Our API takes the maximum number of defender modules as a constraint along with the victim model and training data to generate the corresponding defenders

Each trained defender is then mapped to a hardware accelerator for efficient execution of defender modules and minimize the corresponding run-time overhead

[1] B. Rouhani, M. Javaheripi, M. Samragh. T. Javidi, F. Koushanfar "DeepFence: Characterizing and Defending Adversarial Samples." ICCAD'18

Practical design experiences



Attack scenarios

We evaluate the performance of the proposed countermeasure against the following three attack scenarios:

I. **Fast Gradient Sign**[1]

- Minimizes the L_∞ of the perturbation
- One-shot update rule: $X \leftarrow X + \mathcal{E} \times sign(\nabla_X f)$

II. **JSMA**[2]

- Minimizes the L_2 norm of the perturbation and the number of altered pixels
- Iterative update over selected pixels

III. **Deepfool**[3]

- Minimizes the L_2 norm of the perturbation
- $X \leftarrow X + \mathcal{E} \times P_{opt}$
- P_{opt} is the optimal perturbation in each iteration defined in [3]

IV. **Carlini&WagnerL2**[4]

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples"
[2] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings"
[3] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks"
[4] N. Carlini, D. Wagner, "Towards evaluating the robustness of neural networks"

Black-box attacks

Area Under Curve (AUC) score of MRR methodology against different attack scenarios for MNIST, CIFAR10, and ImageNet benchmark

In this experiment, the attacker knows everything about the DL model but is not aware of the defense mechanism

	MNIST	CIFAR10	ImageNet
FGS	0.996	0.911	0.881
JSMA	0.995	0.966	-
Deepfool	0.996	0.960	0.908
CarliniL2	0.989	0.929	0.907
BIM	0.994	0.907	0.820

Adaptive white-box attack

Our MRR methodology is significantly more robust against prior-art works in face of adaptive white-box attacks

In this experiment, we have considered Carlini and Wagner adaptive attack assuming that the attacker knows everything about the DL model and defense mechanism

	MRR Methodology (White-box Attack)												Prior-Art Defenses (Gray-box Attack)		
Security Parameter	SP=1%						SP=5%						Magnet	Efficient Defenses	APE-GAN
Number of Defenders	N=0	N=1	N=2	N=4	N=8	N=16	N=0	N=1	N=2	N=4	N=8	N=16	N=16	-	-
Defense Success	-	43%	53%	64%	65%	66%	-	46%	63%	69%	81%	84%	1%	0%	0%
Normalized Distortion (L_2)	1.00	1.04	1.11	1.12	1.31	1.38	1.00	1.09	1.28	1.28	1.63	1.57	1.37	1.30	1.06
FP Rate	-	2.9%	4.4%	6.1%	7.8%	8.4%	-	6.9%	11.2%	16.2%	21.9%	27.6%	-	-	-

[1] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017

[2] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Ef- ficient defenses against adversarial attacks. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. ACM, 2017.

[3] Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. Ape-gan: Adversarial perturbation elimination with gan. ICLR, 2017

[4] Nicholas Carlini and David Wagner. “Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples.” arXiv preprint arXiv:1711.08478, 2017.

DeepSigns and DeepMarks

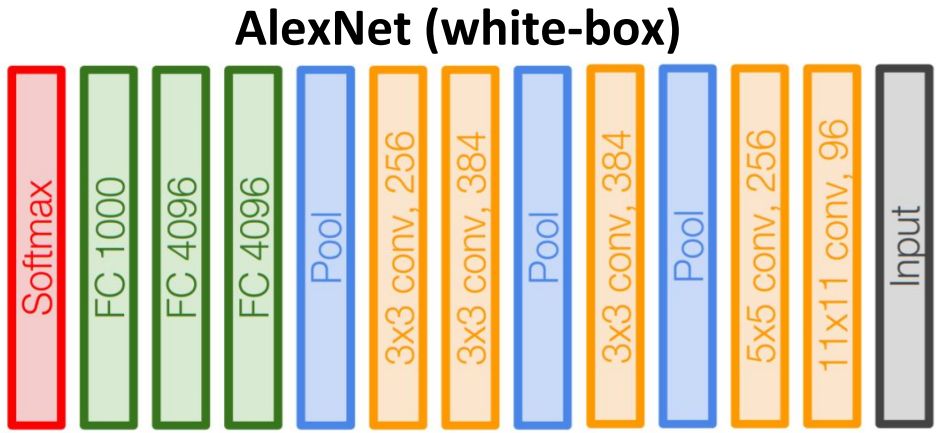
The First Deep Learning IP Protection for both black-box and white-box settings + acceleration and automation for embedded applications

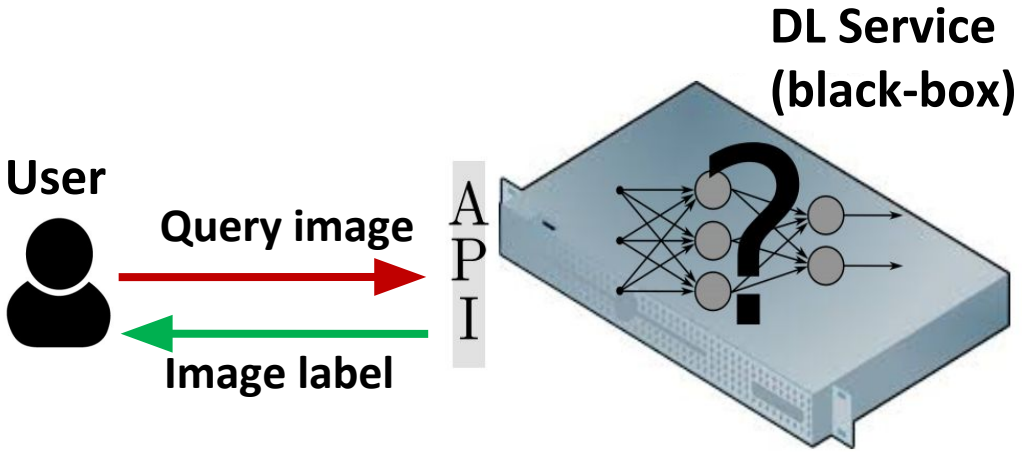
Motivation for ML IP protection

- Training a high-performance Deep Neural Network (DNN) is ***expensive*** since the process requires:
  - Massive amount of proprietary training data
  - Significant computational resources
- Pre-trained DNN is considered as the Intellectual Property (IP) of the model builder and needs to be protected
- Concern: how to prove the ownership of a DNN after it is deployed?

Challenges for watermarking DL

- Various application scenarios:
  - White-box: DNN is shared with the public and model internal details are accessible
  - Black-box: DNN is deployed in a remote service and only the output is accessible
- State-of-the-art solutions:
  - Weights watermarking [1]: only applicable in the white-box setting
  - Zero-bit watermarking [2,3]: embed a zero-bit watermark (WM) in black-box





[1] Y. Uchida, et al. 'Embedding watermarks into deep neural networks', ICMR 2017
[2] E. L. Merrer et al. 'Adversarial frontier stitching for remote neural network watermarking' arXiv preprint 2017
[3] Y Adi, et al. 'Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring', USENIX 2018

DeepSigns' Contribution

- Suggesting the ***first end-to-end*** watermarking framework for systematic IP protection in ***both white-box and black-box*** setting
- Yielding ***high detection rate*** and ***low false alarm rate*** while preserving the prediction ***accuracy***
- ***Robust*** against a variety of model modification attacks and watermark overwriting attacks
- Devising an Application Programming Interface ***(API) to automate*** the adoption of DeepSigns to various DL models, including convolutional, residual, and fully-connected networks.

DeepSigns methodology

- DeepSigns **white-box watermarking**
  - **Embedding**: Train/Fine-tune the target DNN with an activation regularized loss ($\mathcal{L}_{WM}$) in addition to the conventional cross-entropy loss ($\mathcal{L}_0$):

$$\mathcal{L} = \mathcal{L}_0 + \gamma\, \mathcal{L}_{WM}(\mathcal{T}(activations),\ b)$$

    - $\mathcal{T}$: transformation function defined by the model owner
    - b: identity information specified by the owner (a binary string)
    - γ: watermark regularization strength
  - **Extraction**: Threshold the transformation of the activations obtained from the queried DNN:

$$b' = Threshold(\mathcal{T}(activations'),\ \ 0.5)$$

    - Ideally, $b' = b$ (bit error rate is 0) when $\mathcal{L}_{WM}$ is sufficiently minimized during training

DeepSigns methodology (Cont'd)

- DeepSigns **black-box watermarking**
  - **Embedding**: Train/Fine-tine the target DNN with a set of watermarking key image, key label pairs (X^{key}, Y^{key}).
    - The WM key set $\left(X^{key}, Y^{key}\right)$ is designed such that the original model has low accuracy on the key set
    - The mixture of the training data and $\left(X^{key}, Y^{key}\right)$ is used in DNN training to preserve the prediction accuracy while embedding the WM
  - **Detection**: Model owner queries the remote DL service with X^{key} and compares the responses with Y^{key}. The WM is decided to be present if the accuracy on the key set is larger than the pre-defined threshold

Automation of DeepSigns

- DeepSigns provides ***wrapper*** that can be readily integrated with popular DL frameworks, including TensorFlow, PyTorch, Theano

```
## DeepSigns Wrapper Utilization in white-box setting
from DeepSigns import subsample_training_data
from DeepSigns import WM_activity_regularizer
from DeepSigns import get_activations
from DeepSigns import extract_WM_from_activations
from DeepSigns import compute_BER
from utils import create_marked_model

## create WM trigger keys
X^key = subsample_training_data(T, {X^train , Y^train })
## instantiate customized WM activity regularizer
WM_reg = WM_activity_regularizer(λ1, λ2, b, A)
model = create_marked_model(WM_reg , model topology)
## embed WM by standard training of the marked model
model.fit(X^train , Y^train)

## extract WM from the activation maps and compute BER
μ_l^{s*M} = get_activations(model, X^key, l, T)
b̃ = extract_WM_from_activations(μ_l^{s*M}, A)
BER = compute_BER(b̃, b)
```

```
## DeepSigns Wrapper Utilization in black-box setting
from DeepSigns import key_generation
from DeepSigns import count_response_mismatch
from DeepSigns import compute_mismatch_threshold
from utils import create_model

## generate WM key pairs
model = create_model(model topology)
model.load_weights('baseline_weights')
(X^key, Y^key) = key_generation(model, K, X^train)
X^retrain = np.vstack(X^key, X^train)
Y^retrain = np.vstack(Y^key, Y^train)
## embed WM by finetuning the model with the WM key
model.fit(X^retrain, Y^retrain)

## query model with key set to detect WM
Y^pred = model.predict(X^key)
m = count_response_mismatch(Y^pred , Y^key)
θ = compute_mismatch_threshold(p, K, C)
WM_detected = 1 if m < θ else 0
```

DeepSigns performance

- DeepSigns Performance:
  - **Functionality preserving**: The watermarked model achieves the same level of accuracy compared to the baseline model

Dataset	Baseline Accuracy	Accuracy of Marked Model		DL Model Type	DL Model Architecture
MNIST	98.54%	K = 20	N = 4	MLP	784-512FC-512FC-10FC
		98.59%	98.13%		
CIFAR10	78.47%	K = 20	N = 4	CNN	3*32*32-32C3(1)-32C3(1)-MP2(1) -64C3(1)-64C3(1)-MP2(1)-512FC-10FC
		81.46%	80.70%		
CIFAR10	91.42%	K = 20	N = 128	WideResNet	Please refer to [32].
		91.48%	92.02%		
ImageNet	74.72%	K = 20	–	ResNet50	Please refer to [4].
		74.21%	–		

DeepSigns performance (cont'd)

- DeepSigns Performance:
  - **Robustness against pruning attack**: Tolerate up to 90%, 99%, and 99.5% parameter pruning on MNIST, CIFAR-10, and ImageNet dataset, respectively
  - **Robustness against fine-tuning**: The embedded WM can be detected after the marked model is fine-tuned
  - **Robustness against overwriting**: The original WM remains detectable after the attacker embeds a new WM using the same approach
  - **Capacity**: Allows up to 64 bits and 256 bits WM embedding on MNIST and CIFAR10 dataset
  - **Security**: DeepSigns watermarking method leaves no tangible footprint in the model, thus the attacker cannot detect the presence of the WM

Motivation for DL fingerprinting

- Digital watermarking technique cannot distinguish different users who are using the same IP provided by the model owner
- If IP infringement is discovered, how to determine which user has misused the IP? – Fingerprinting!

- Digital Fingerprinting of DNNs: make each distributed DNN unique and distinguishable

Challenges for fingerprinting

- There are no prior works on digital fingerprinting of DNNs
- Existing DNN watermarking frameworks only consider the singer-user scenario and provide ownership proof for IP protection
- How to provide a robust, collusion-secure solution that supports both IP protection and Digital Right Management (DRM) for DNNs in a multi-user setting?

DeepMarks' contribution

- Proposing the ***first end-to-end fingerprinting*** methodology for systematic IP protection and DRM in the DL domain
- Enabling ***unique identification*** of users
- ***Robust*** against a variety of model transformation attacks and fingerprint collusion attack
- Devising an ***(API) to automate*** the adoption of DeepMarks fingerprinting technique to various DNN architectures

DeepMarks methodology

- DeepMarks includes two modules for fingerprinting a DNN:
  - **Fingerprint construction**: The model owner generates an anti-collusion codebook (ACC, [4]) $\mathbf{C}$ and an orthogonal basis matrix $\boldsymbol{U}$ to construct FPs for all users:

$$\boldsymbol{f_j} = \sum_{i=1}^{v} \boldsymbol{b_{ij}}\ \boldsymbol{u_j},\ \textbf{for j=1,..., n, and } \boldsymbol{b_{ij}} \textbf{ = 2 } \boldsymbol{c_{ij}} \textbf{ - 1.}$$

  - **Fingerprint embedding**: The owner trains/ fine-tunes the target DNN with the following regularized loss:

$$\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{L}_0} + \gamma\, MSE\left(\boldsymbol{f_j} - \boldsymbol{Aw}\right)$$

    - $\boldsymbol{f_j}$: fingerprint for j^{th} user
    - A: secret projection matrix specified by the owner
    - w: weights selected by the owner
    - γ: fingerprint regularization strength

[4] Y. Yu et al, "Group-oriented anti-collusion fingerprint based on BIBD code', EBISS 2010

DeepMarks methodology

- DeepMarks supports the following two usages:
  - **User identification**: The model owner extracts the code-vector ($\boldsymbol{c_j}'$) from the weights of the queried DNN by computing:

$$\boldsymbol{f_j' = Aw'}$$

$$\boldsymbol{b_j' = f_j' * U, \quad c_j' = \frac{b_j' + 1}{2}}$$

The recovered code-vector $\boldsymbol{c_j}'$ is compared with the codebook **C** to uniquely identify the user.

  - **Collusion detection**: The model owner extracts the code-vector from the colluded DNN and uses the property of anti-collusion code to uniquely identify the colluders

$$\boldsymbol{colluders = ACC\_identify(C,\ c_{colluded})}$$

DeepMarks automation

- DeepMarks provides ***wrapper*** that is compatible with existing DL frameworks (e.g. TensorFlow, PyTorch, Theano)
- The wrapper supports two utilizations:
  - User identification
  - Collusion detection

```
## DeepMarks Wrapper Utilization
from DeepMarks import generate_BIBD_ACC_codebook
from DeepMarks import FP_weight_regularizer
from DeepMarks import extract_FP_from_weights
from DeepMarks import identify_user
from DeepMarks import detect_colluders
from utils import create_marked_model

## model owner generates codebook for all users
C = generate_BIBD_ACC_codebook(v, n, k)
for i in range(n):
      ## instantiate customized FP weight regularizer for i^th user
      FP_reg = FP_weight_regularizer(λ, c_i, A)
      model = create_marked_model(FP_reg , model topology)
      ## embed FP by standard training with FP regularizer
      model.fit(X^train , Y^train)

## extract FP from the weights and identify unique users
c̃ = extract_FP_from_weights(w̃, A)
ĩ = identify_user(c̃, C)
## detect participants of collusion attack from the colluded weights
i_colluded = detect_colluders(w_colluded, C)
```

DeepMarks performance evaluation

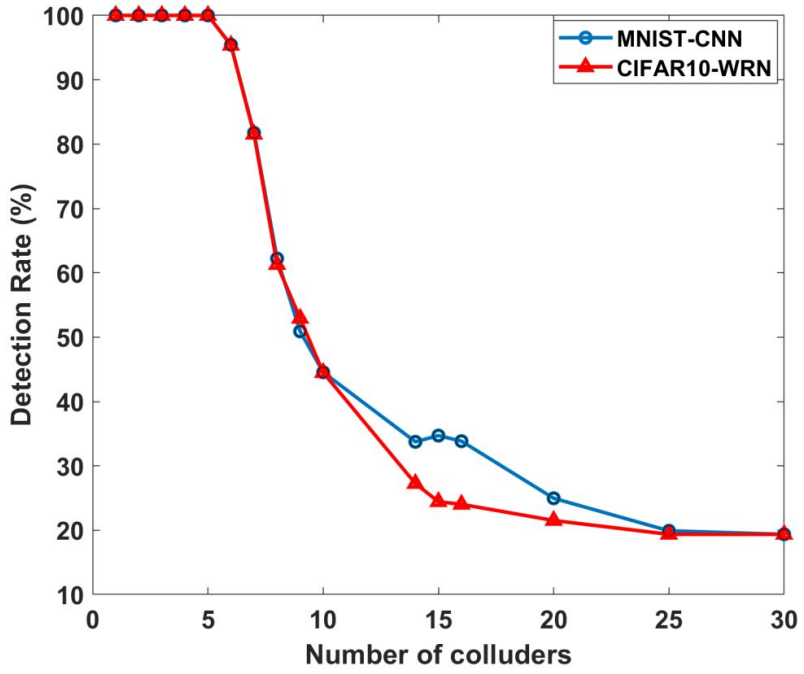
- DeepMarks Performance:
  - **Functionality preserving**: The fingerprinted model achieves a comparable accuracy as baseline model

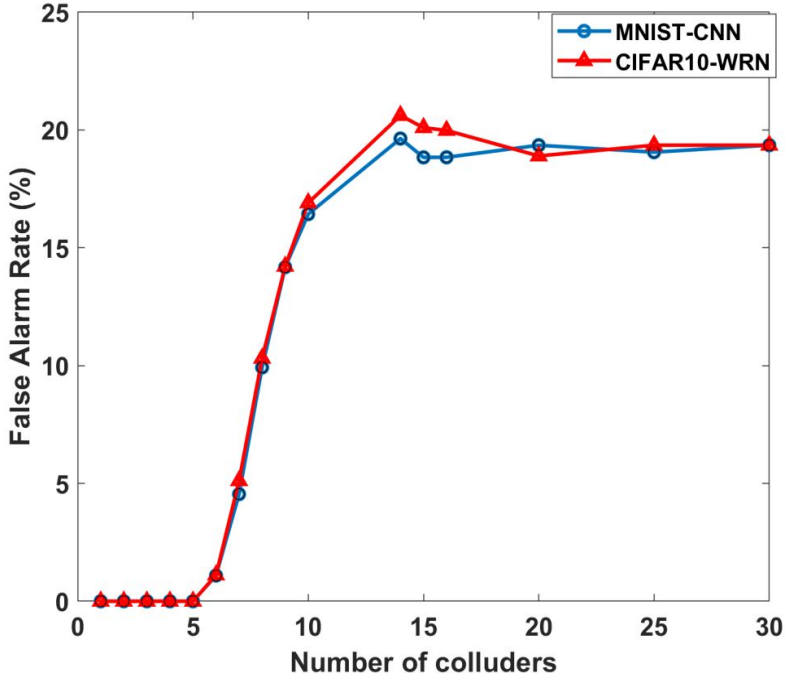
Benchmark	MNIST-CNN			CIFAR10-WRN		
Setting	Baseline	Fine-tune without fingerprint	Fine-tune with fingerprint	Baseline	Fine-tune without fingerprint	Fine-tune with fingerprint
Test Accuracy (%)	99.52	99.66	99.72	91.85	91.99	92.03

  - **Robust against parameter pruning**: Tolerate up to 95% and 99% parameter pruning on MNIST and CIFAR10 benchmarks
  - **Robust against fine-tuning:** The embedded fingerprint can be extracted after the model is fine-tuned

DeepMarks performance (cont'd)

- DeepMarks Performance:
  - **Collusion resilience**: The maximal number of detectable colluders (which is 5 as shown below) is consistent with the theoretical value given by ACC





Machine Learning on Encrypted Data

Cryptographically secure and on embedded devices...

Cryptographic tools

- Garbled Circuits (GC): a generic Secure Function Evaluation (SFE) protocol that enables two parties to evaluate an arbitrary function on the private data in constant number of interactions.
- Goldreich-Micali-Wigderson (GMW): an SFE protocol that requires one round of interaction between two parties per layer of AND gates. Requires lower data transfer compared to GC.
- Secret Sharing (SS): a method to distribute a share among several untrusted parties, e.g., additive secret sharing and Shamir's secret sharing.
- Homomorphic Encryption (HE): a cryptographic encryption scheme that allows computation on encrypted form of data.

Private training frameworks

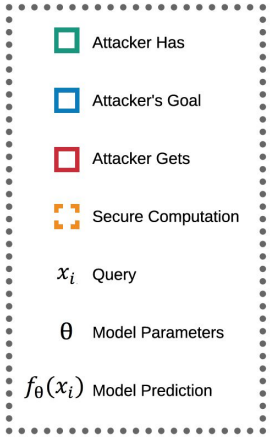
- Shokri and Shmatikov[1]: a method for collaborative deep learning that provides differential privacy. Users download the model, update the model using their own training data and upload it to the cloud. To provide privacy, users update DL model only for a subset of parameters and add specific noise to the updates. Broken by Hitaj et al.[2]
- Google[3]: proposed a secure aggregation of high-dimensional vectors held by different users. The method is based on Shamir's secret sharing and is robust against users dropping in the middle of the protocol.
- SecureML[4]: a system for privacy-preserving machine learning in general, and neural networks in particular. The system is based on HE, GC, and SS protocols. Data owners secret share their data with two non-colluding servers which privately train the neural network.

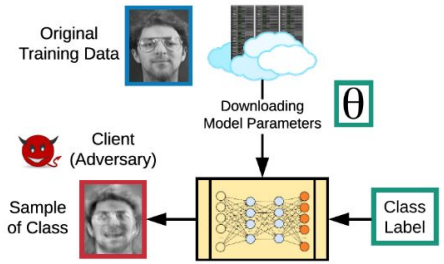
[1] Shokri, Reza, and Vitaly Shmatikov. "Privacy-preserving deep learning." In CCS, 2015.
[2] Hitaj, Briland, Giuseppe Ateniese, and Fernando Perez-Cruz. "Deep models under the GAN: information leakage from collaborative deep learning." In *CCS, 2017.*
[3] Bonawitz, Keith, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical secure aggregation for privacy-preserving machine learning." In *CCS*, 2017.
[4] Mohassel, Payman, and Yupeng Zhang. "SecureML: A system for scalable privacy-preserving machine learning." In *S&P*, 2017.

Private inference frameworks

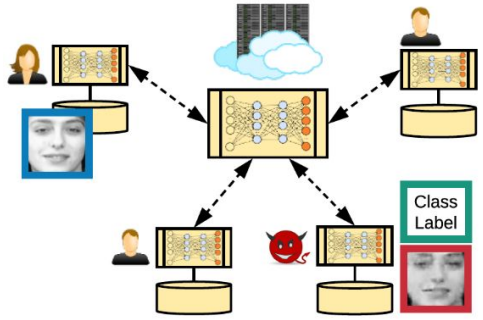
Framework	Prediction Timing (s)			Communication (MB)			Accuracy (%)	Deep Neural Network Architecture
	Offline	Online	Total	Offline	Online	Total		
CryptoNets	-	297.5	297.5	-	372.2	372.2	98.95	Conv - Sqr Act - MeanP - Sqr Act - FC
SecureML	4.7	0.18	4.88	-	-	-	93.1	FC - Sqr Act - FC - Sqr Act - FC
MiniONN (w/ Sqr Act.)	0.90	0.14	1.04	3.8	12	15.8	97.6	
MiniONN (w/ Relu & Pooling)	3.58	5.74	9.32	20.9	636.6	657.5	99	Conv - ReLu - MaxP - ReLu - MaxP - FC - ReLu - FC
DeepSecure	-	9.67	9.67	-	791	791	99	Conv - ReLu - FC - ReLu - FC
DeepSecure (+ Preprocessing)	-	1.08	1.08	-	88.2	88.2	99	
Chameleon	1.34	1.36	2.70	7.8	5.1	12.9	99	

Attacks on Neural Networks

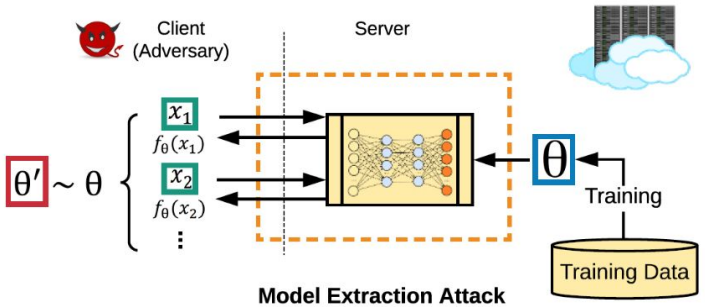




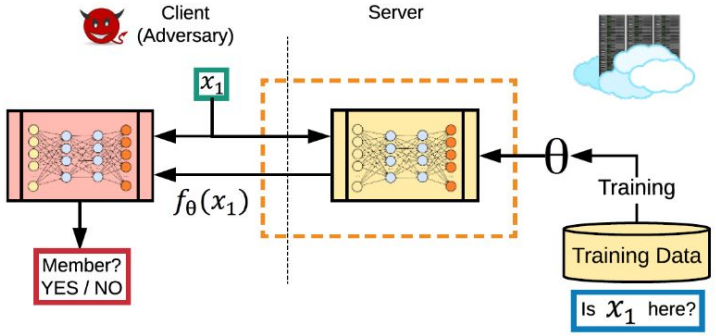
Model Inversion Attack



GAN-based Attack



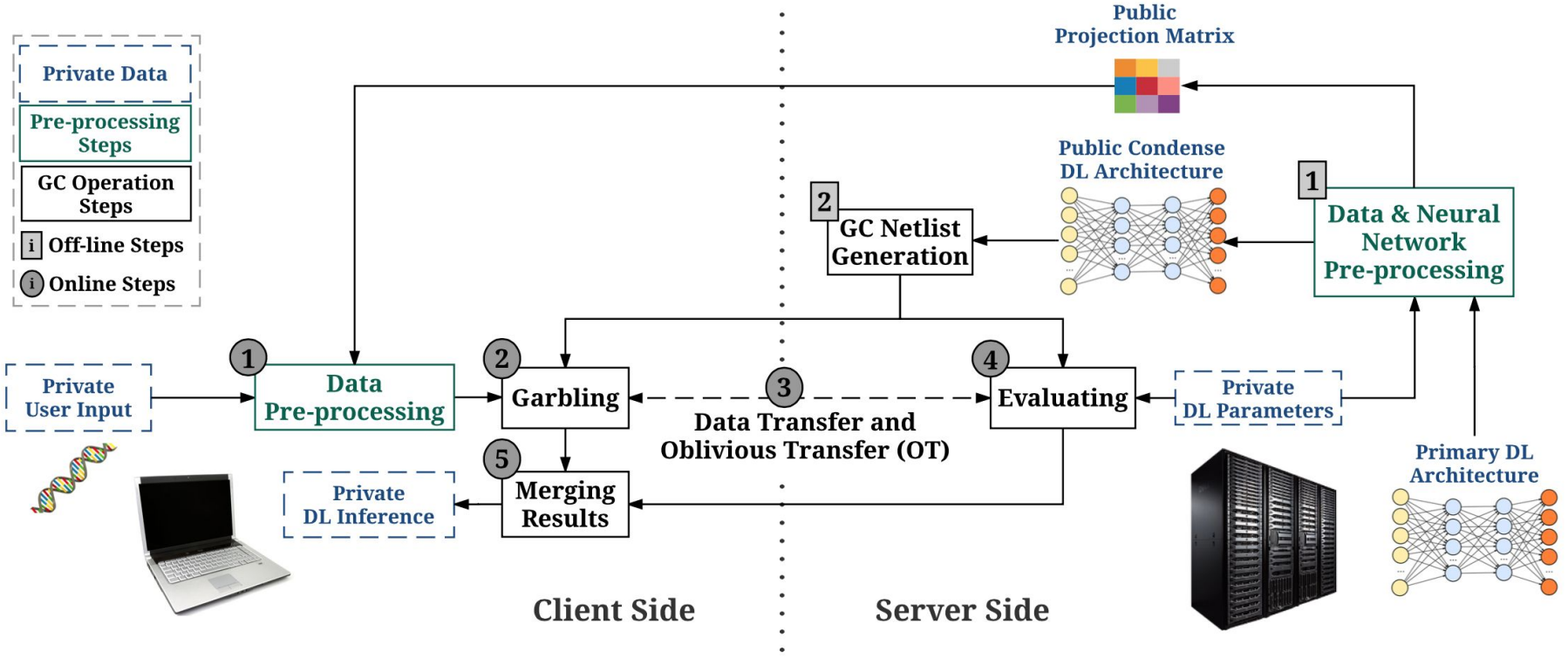
Model Extraction Attack



Membership Inference Attack

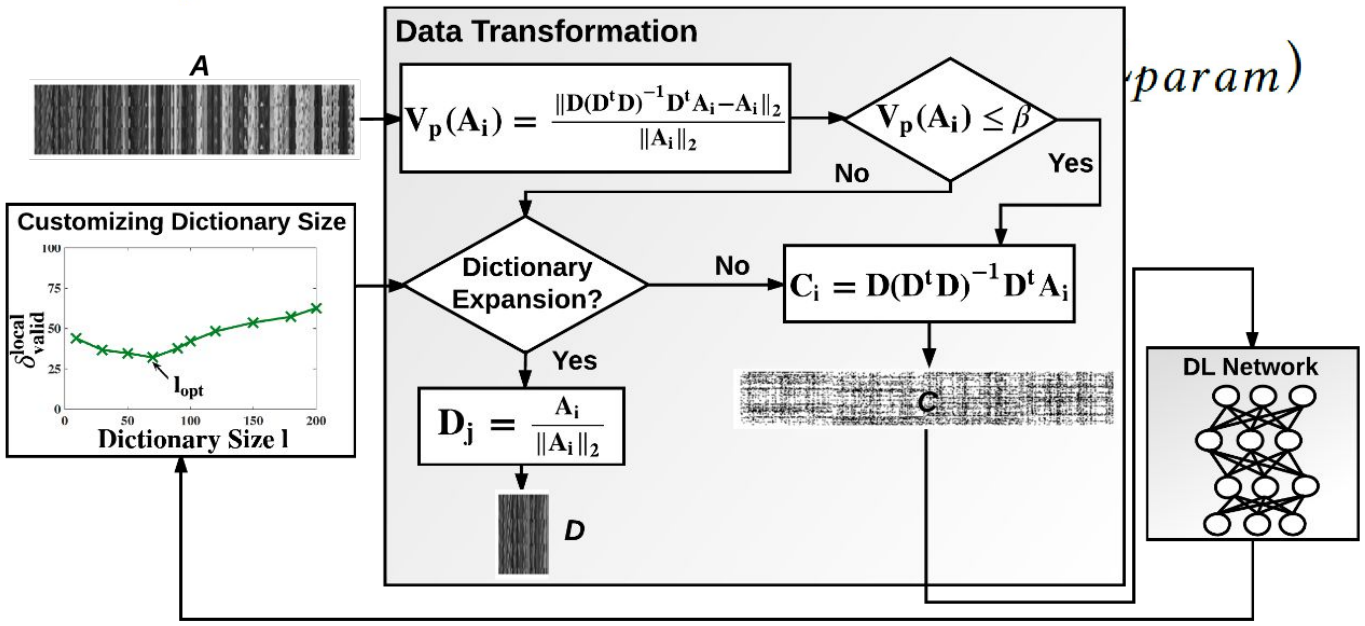
Fredrikson et al. "Model inversion attacks that exploit confidence information and basic countermeasures.“ In CCS’15.
Tramèr et al. "Stealing Machine Learning Models via Prediction APIs." In USENIX Security’16.
Hitaj et al. "Deep models under the GAN: information leakage from collaborative deep learning." In CCS’17.
Shokri et al. "Membership inference attacks against machine learning models." In S&P’17.

DeepSecure

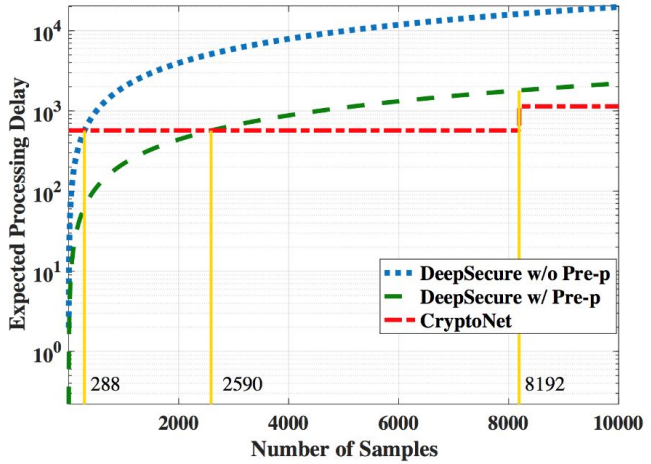

Private Data
Pre-processing Steps
GC Operation Steps
i Off-line Steps
i Online Steps
Public Projection Matrix
Public Condense DL Architecture
1
Data & Neural Network Pre-processing
2
GC Netlist Generation
Private User Input
1
Data Pre-processing
2
Garbling
3
Data Transfer and Oblivious Transfer (OT)
4
Evaluating
Private DL Parameters
5
Merging Results
Private DL Inference
Primary DL Architecture
Client Side
Server Side

DeepSecure preprocessing

$$\underset{D,\ \tilde{DL}_{param}}{Minimize}\ (T_{comm})\ s.t.,\ ||A - DC||_F \leq \epsilon||A||_F$$

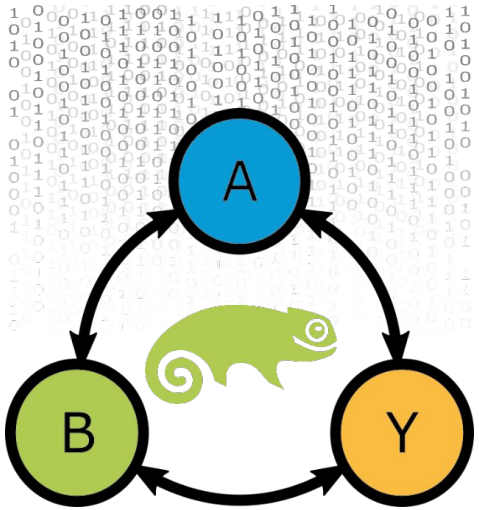


DeepSecure performance



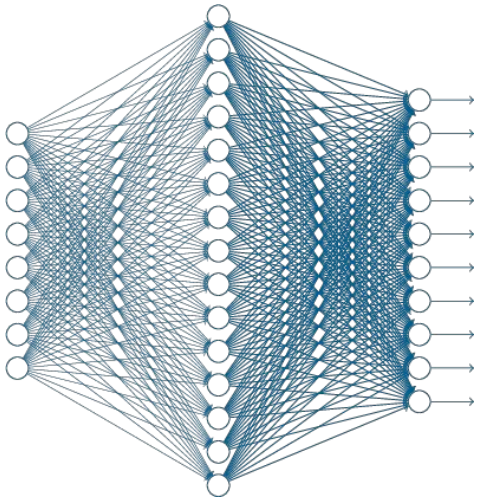
Framework	Comm.	Comp. (s)	Execution (s)	Improvement
DeepSecure without pre-processing	791MB	1.98	9.67	58.96 ×
DeepSecure with pre-processing	88.2MB	0.22	1.08	527.88×
CryptoNets	74KB	570.11	570.11	-

Chameleon



$$\begin{pmatrix} -1.02 \\ 8.43 \\ 9.24 \\ -4.38 \\ -9.65 \\ 3.55 \\ 7.58 \\ 4.51 \end{pmatrix} \times \begin{pmatrix} 0.38 \\ 2.32 \\ -5.97 \\ 5.35 \\ 2.33 \\ -6.87 \\ 5.84 \\ 2.47 \end{pmatrix}$$

$$= -1.02 \times 0.38 + \cdots + 4.51 \times 2.47$$



STP-aided Mixed-Protocol Framework for SFE

>300x less communication for pre-computation

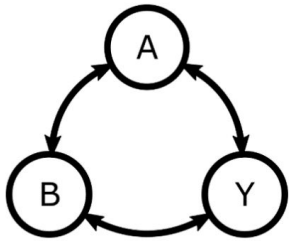
Optimized VDP protocol on Signed Fixed-Point Numbers (SFN)

VDP pre-computation at communication cost of **2 multiplications**

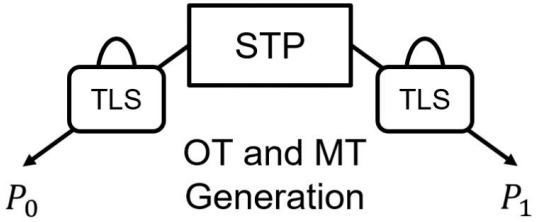
Proof-of-Concept Implementation, Evaluation on CNNs (+ SVMs)

>100x over Microsoft CryptoNets, **> 4x** over MiniONN [LJLA17]

Chameleon protocol



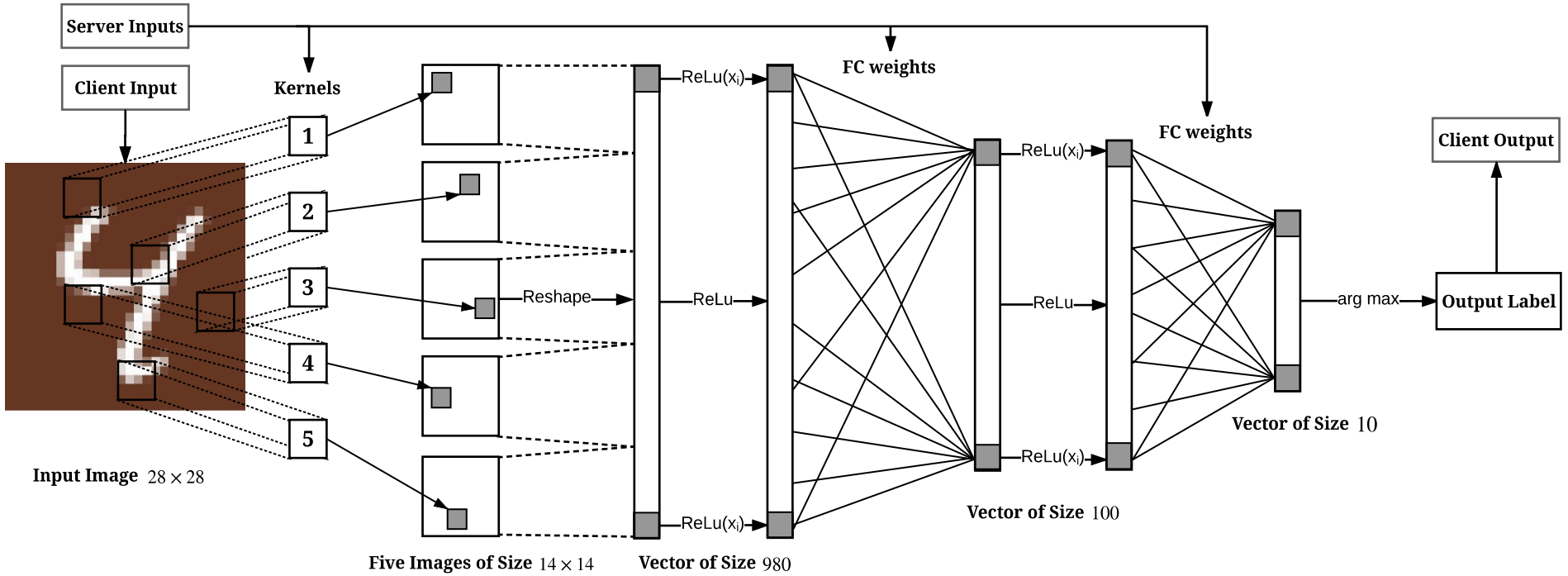
Pre-computation Phase



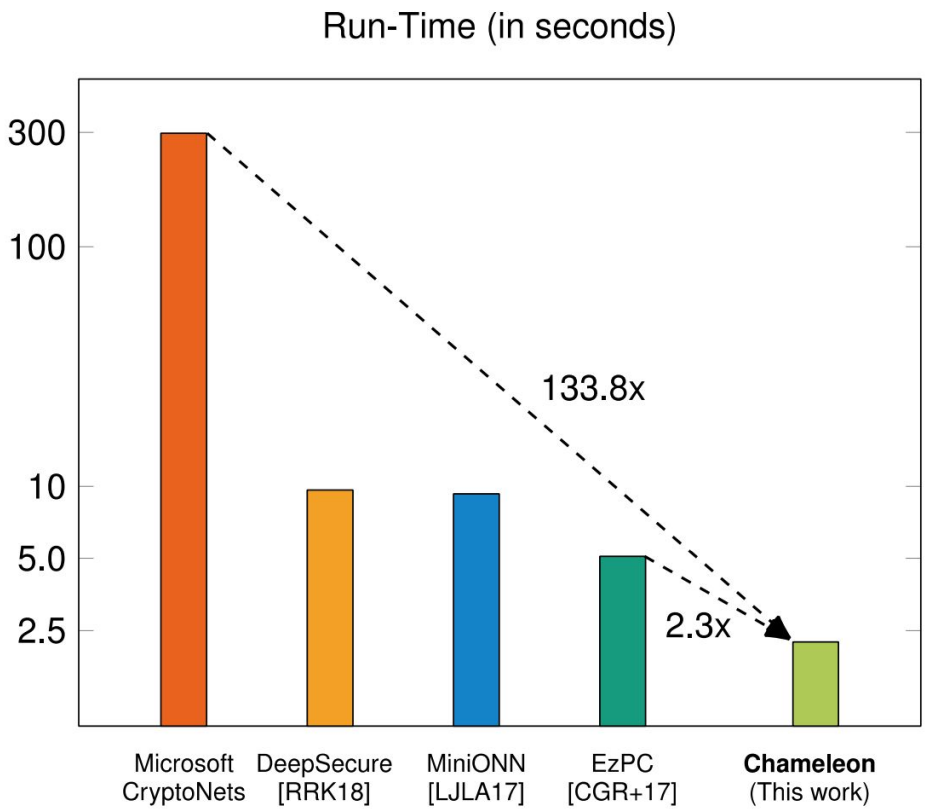
Online Phase

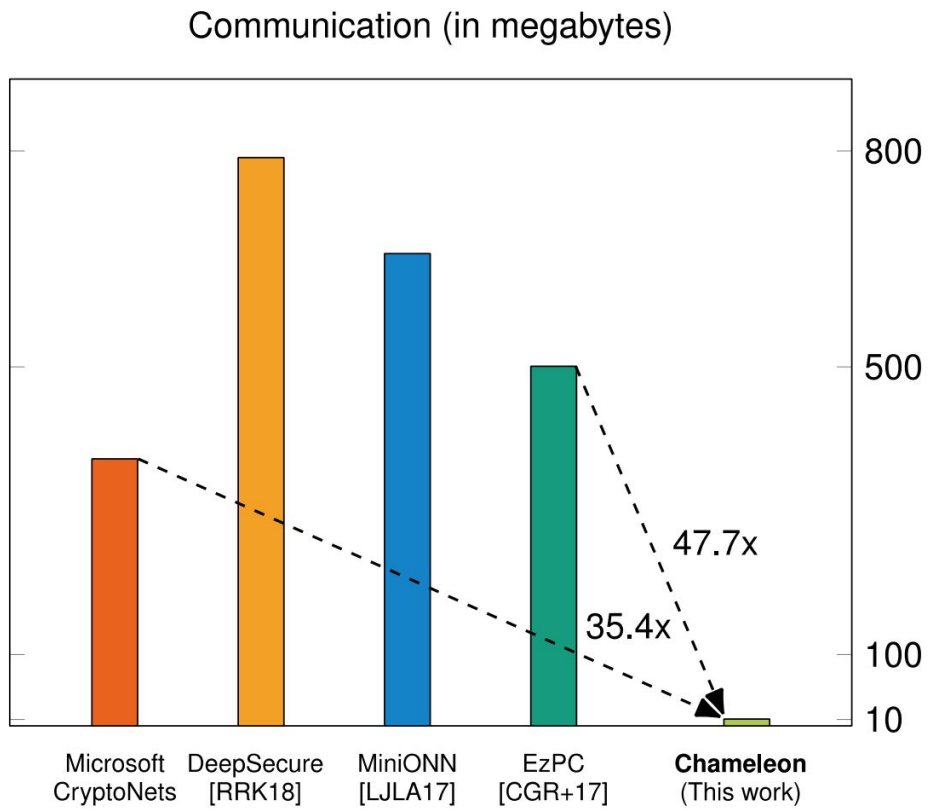
arm
TRUSTZONE

Convolutional Neural Networks (CNNs)



Chameleon performance





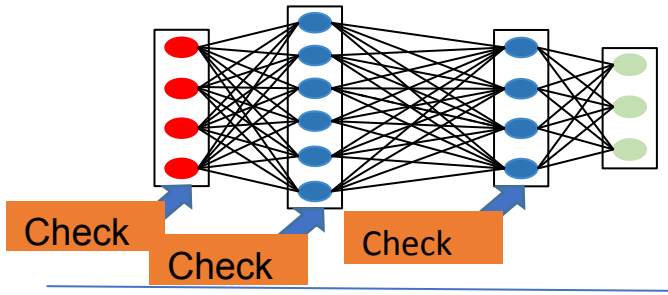
Summary and outlook

- Automation revolution and ML
- ML is increasingly applied on embedded devices
- Several risks associated, e.g.,
  - Adversarial ML
  - IP theft
  - Privacy concerns due to edge learning and sharing and cloud
- Some recent MICS solutions
  - DeepFence, DeepMarks, DeepSigns, and DeepSecure+
- Several standing challenges and opportunities remain...

Safe embedded ML technologies in UCSD/MICS

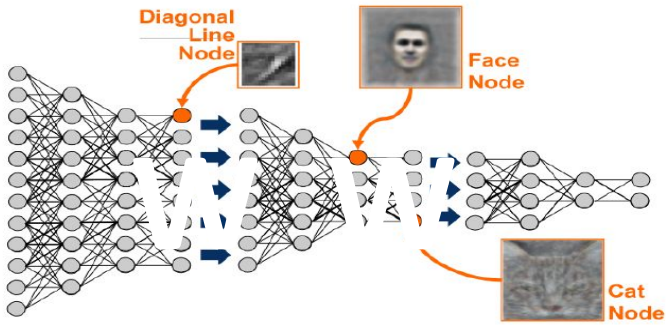
DeepFence

The first comprehensive defense Against adversarial DL on ES



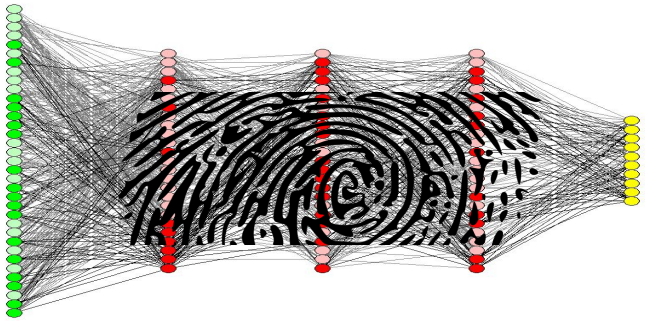
DeepMarks

The first unremovable DL watermarks



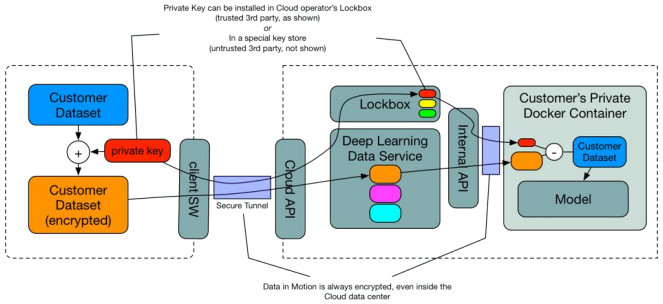
DeepSigns

The first unremovable DL fingerprints



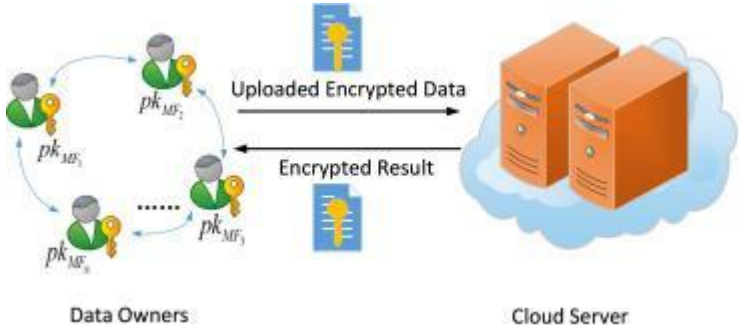
DeepIPTrust

The first hybrid trusted platform & DL for IP protection



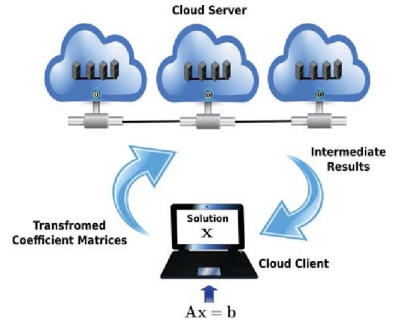
DeepSecure & Chameleon

The most efficient DL on encrypted data



Secure Federated ML

Efficient secure distribued&federated ML



Thank you!

Farinaz Koushanfar

farinaz@ucsd.edu