# CS 374: Operating Systems I

# Part IV – Rowhammer and side-channels

Sanghyun Hong

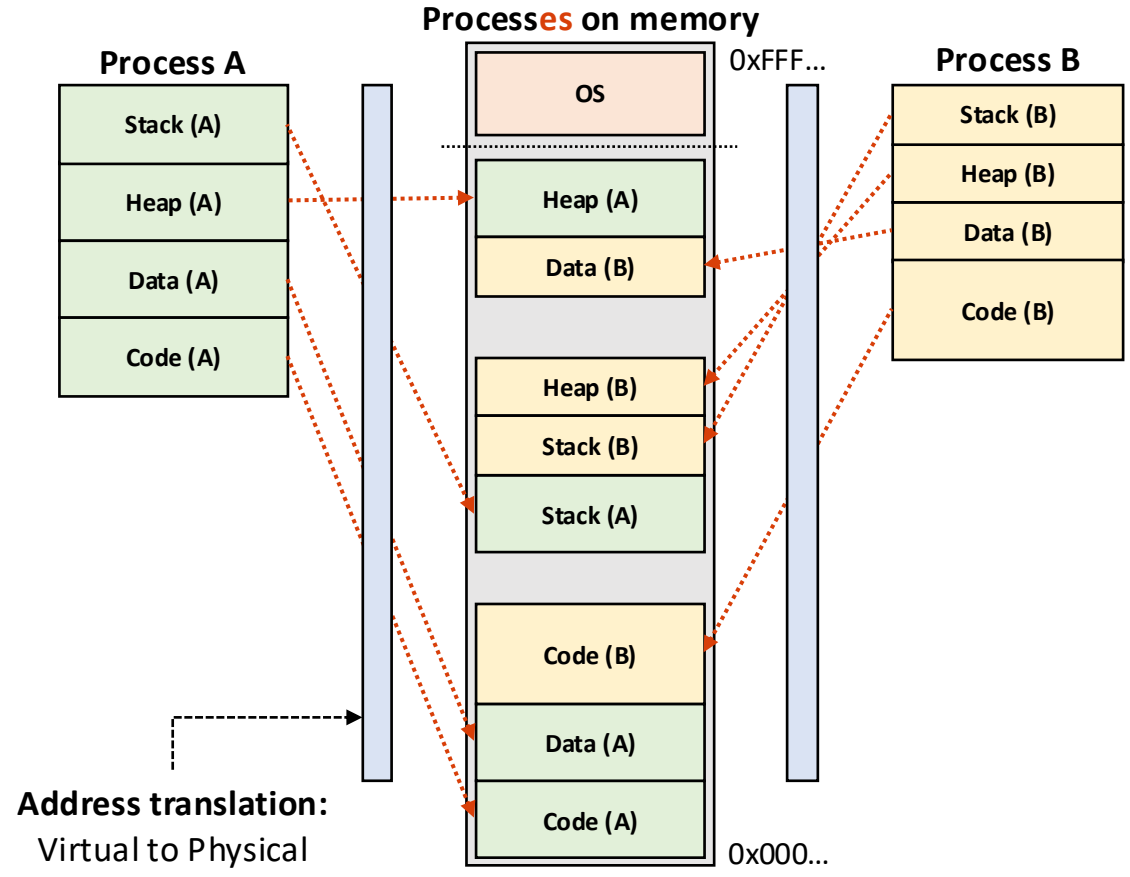
sanghyun.hong@oregonstate.edu


SAIL

Secure AI Systems Lab

# PROCESS ISOLATION

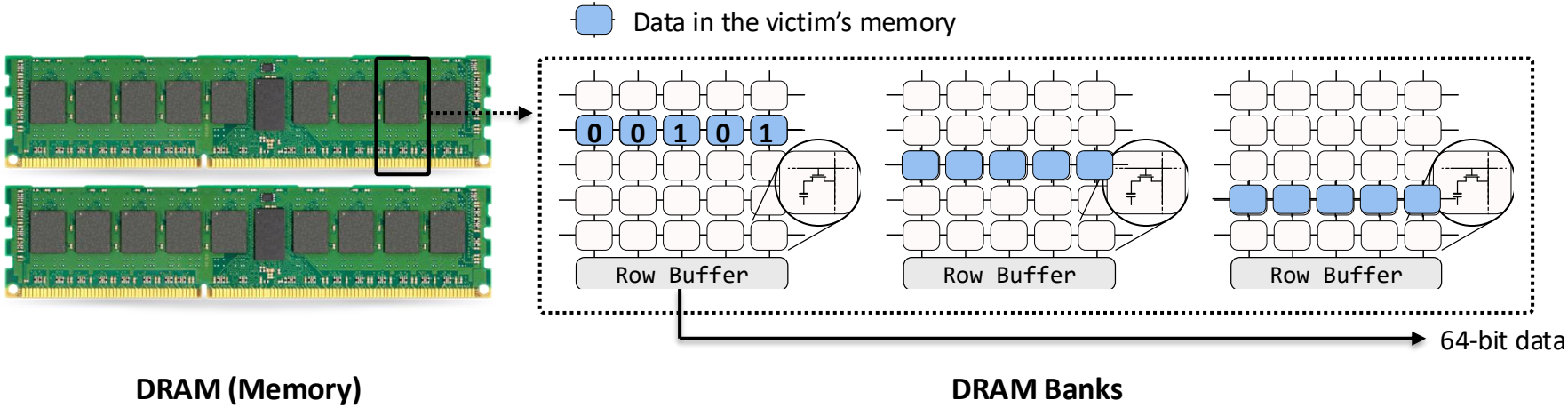
- **Security mechanism**
  - No segment is shared
  - Security reasons
    - Data breach
    - System crashes
    - Control other processes
    - …



# ROWHAMMER: SOFTWARE-INDUCED HARDWARE FAULTS

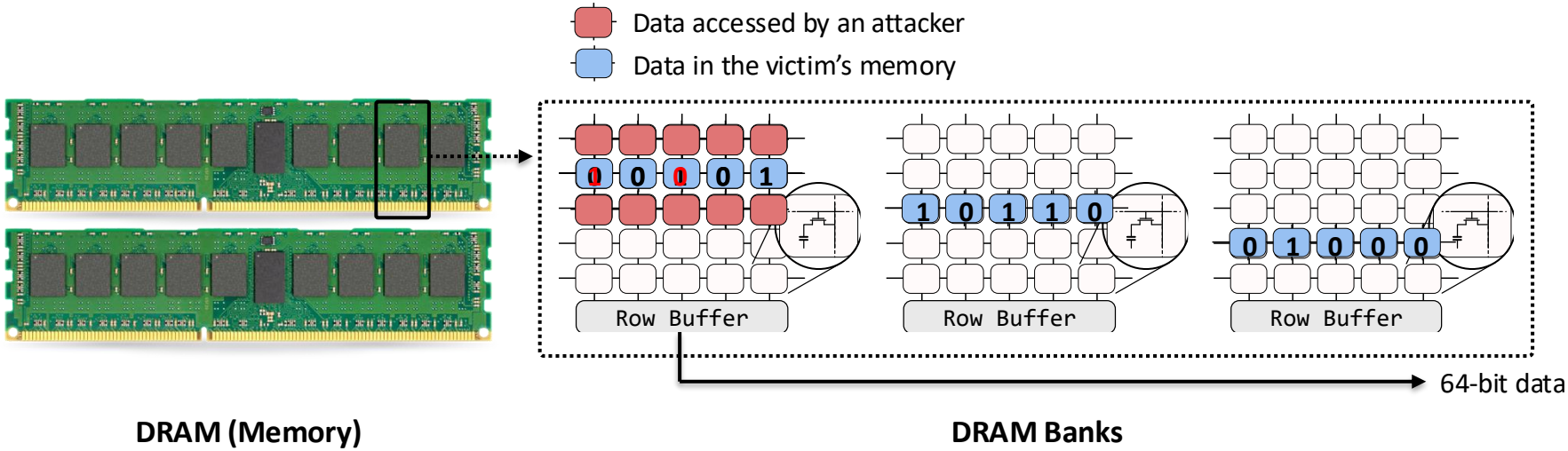
# PRACTICAL HARDWARE ATTACK – ROWHAMMER

- Rowhammer attacks
  - Single-bit corruption primitives in DRAM-level
  - Software-induced hardware fault attack



# PRACTICAL HARDWARE ATTACK – ROWHAMMER

- Rowhammer attacks
  - Single-bit corruption primitives in DRAM-level
  - Software-induced hardware fault attack
  - Cross-VM: attacker only requires a co-located VM



# ROWHAMMER X AI – UNEASY COEXISTENCE

# GRACEFUL DEGRADATION[1]

- **Techniques** that rely on the *graceful degradation*
  - **Pruning**[2] : to reduce the inference cost
  - **Quantization**[3] : to compress the network size
  - **Adding noise**[4] : to improve the robustness against adv. examples

[1]LeCun et al., *Optimal Brain Damage*, NIPs'90
[2]Li et al., *Pruning Filters for Efficient ConvNets*, ICLR'17
[3]Wang et al., *Training Deep Neural Networks with 8-bit Floating Point Numbers*, NeuralPS'18
[4]Zhou et al., *Breaking Transferability of Adversarial Samples with Randomness*, ArXiv'18

# GRACEFUL DEGRADATION[1]

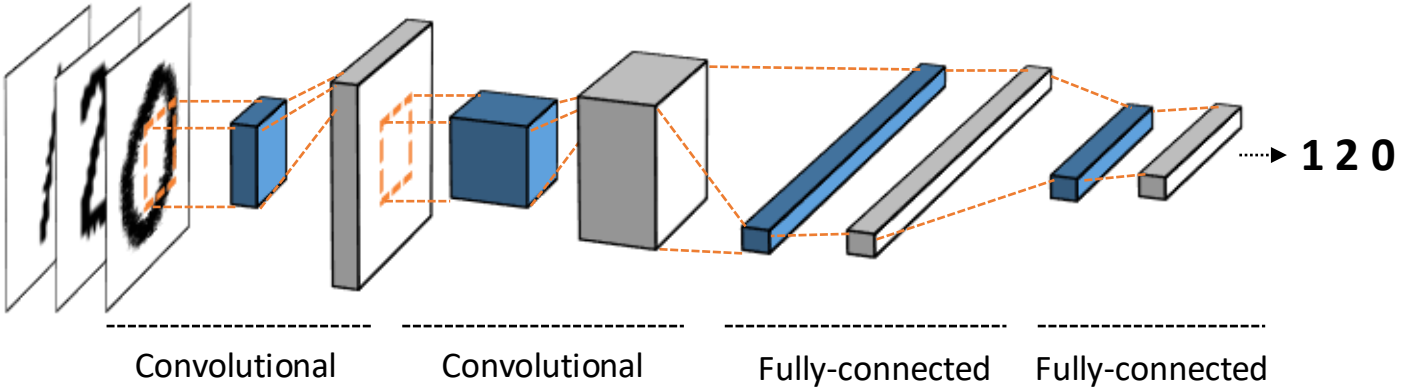
- **Techniques** that rely on the *graceful degradation*
  - **Pruning**[2] : to reduce the inference cost
  - **Quantization**[3] : to compress the network size
  - **Adding noise**[4] : to improve the robustness against adv. examples

- **Prior work** showed it is difficult to *cause the accuracy drop*
  - **Indiscriminate poisoning**[4]: blend poisons ≈ **11% drop** (avg.)
  - **Storage media errors**[5] : a lot of random bit errors ≈ 5% drop (avg.)
  - **Hardware fault attacks**[6,7] : a lot of random faults ≈ 7% drops (avg.)

[4]Steinhardt et al., *Certified Defenses for Data Poisoning Attacks*, NeuralPS'17
[5]Qin et al., *Robustness of Neural Networks against Storage Media Errors*, Arxiv'17
[6]Li et al., *Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications*, SC'17
[7]Breier et al., *DeepLaser: Practical Fault Attack on Deep Neural Networks*, Arxiv'18

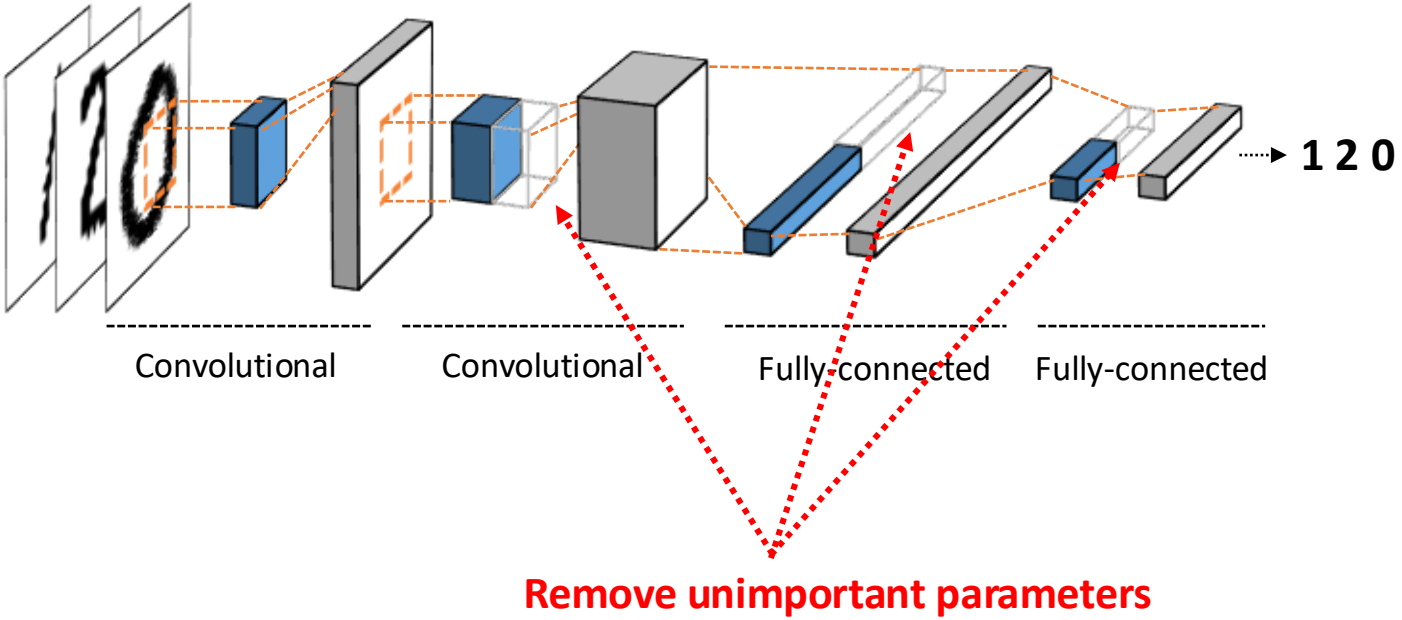
# ILLUSTRATION: HOW DNN COMPUTES

- Accuracy: 99%



# THE BEST-CASE: OPTIMAL BRAIN DAMAGE[1]

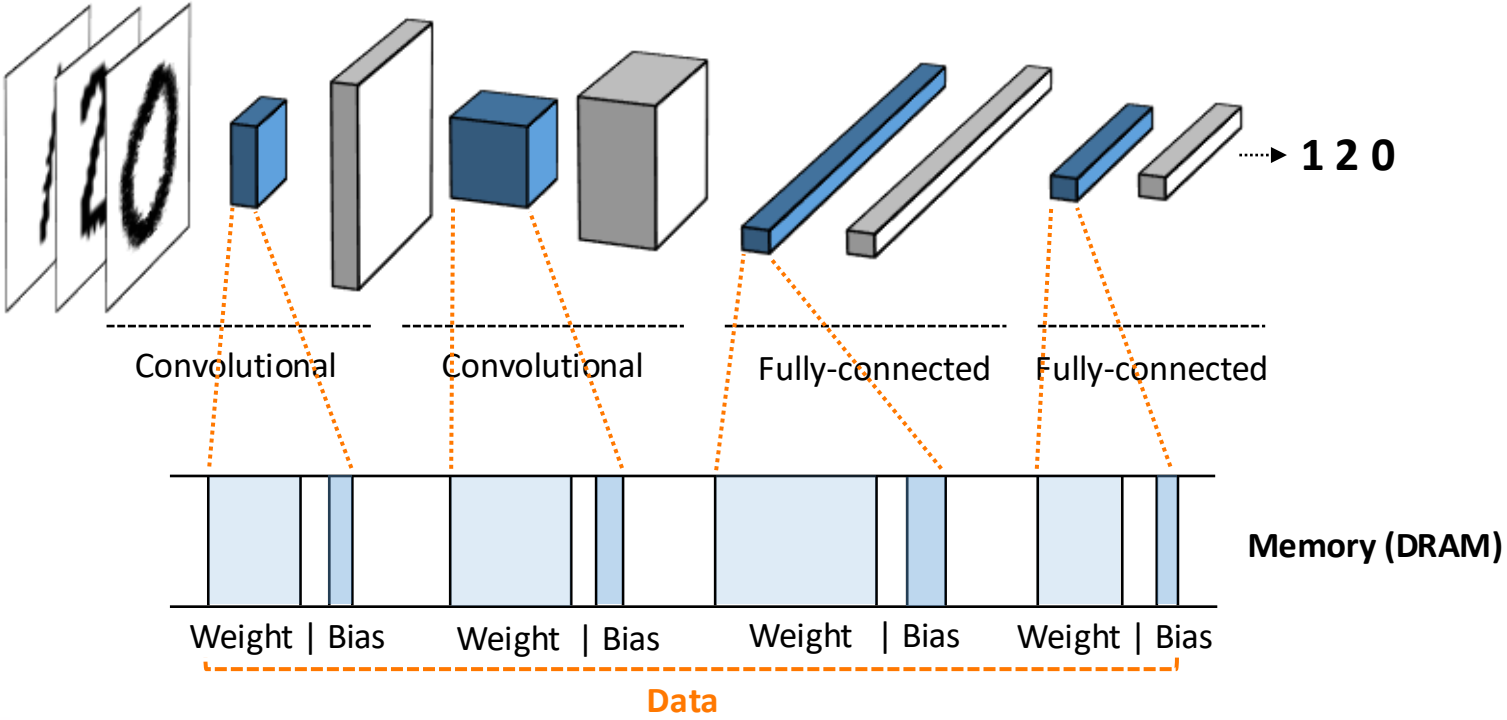
- Accuracy: **99%** (0% drop)




[1]LeCun et al., *Optimal Brain Damage*, NIPs'90

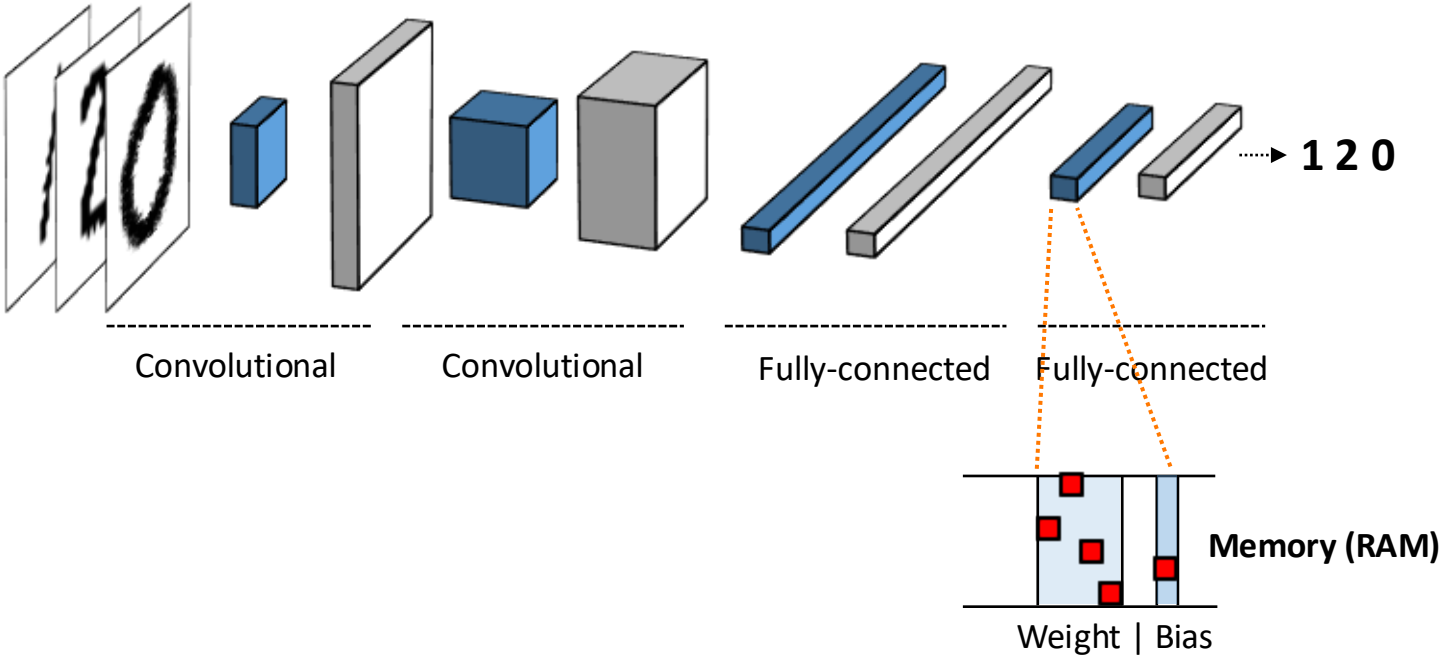
# ILLUSTRATION: DNN'S IN-MEMORY REPRESENTATION

- Accuracy: 99%



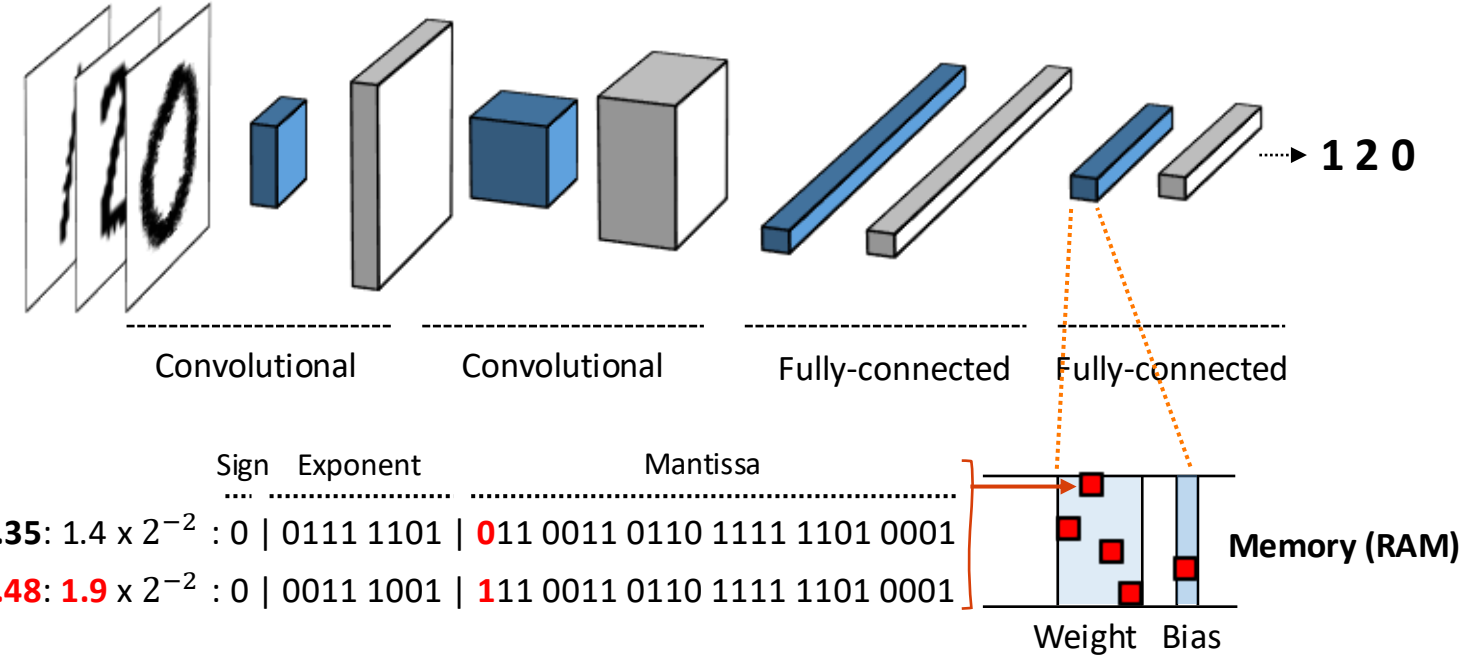
# THE AVG-CASE: BITWISE ERRORS IN DNN'S IN-MEMORY REPR.

- Accuracy: **94%** (5% drop on avg.)



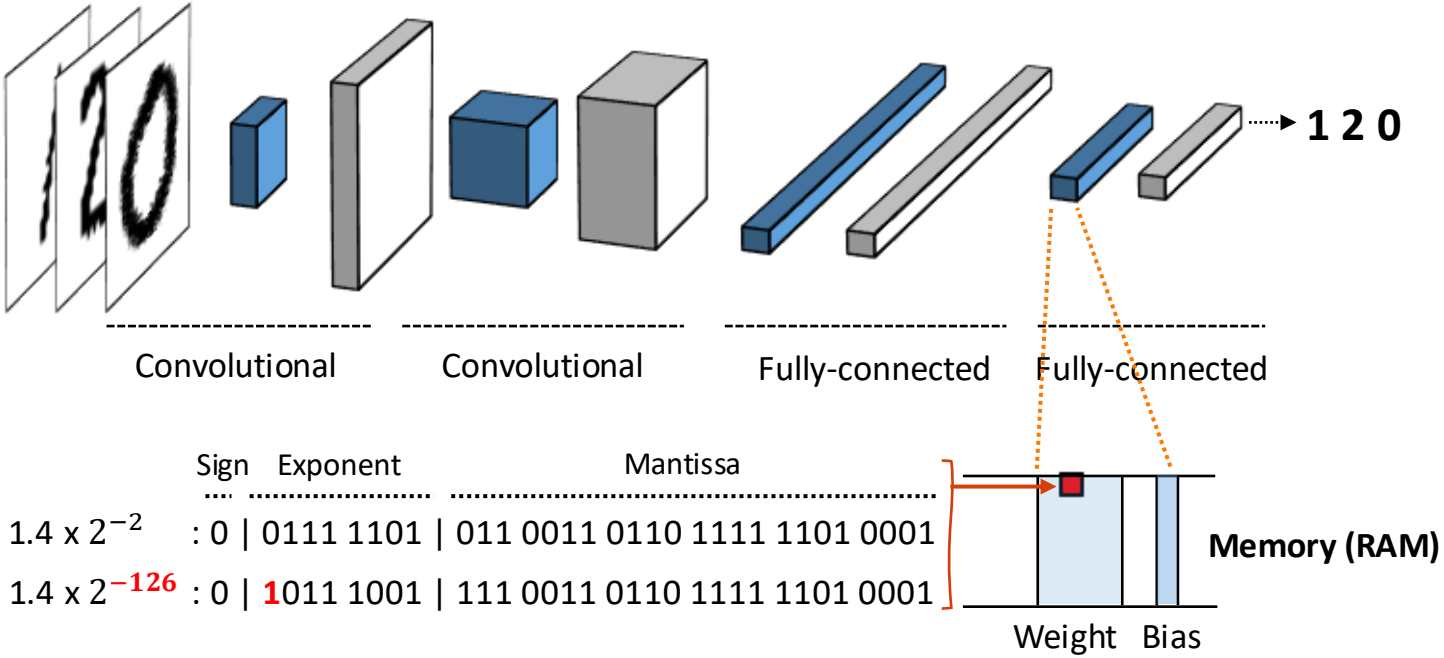
# THE AVG-CASE: BITWISE ERRORS IN DNN'S IN-MEMORY REPR.

- Accuracy: **94%** (5% drop on avg.)



# THE WORST-CASE: A SINGLE BIT-FLIP

- Accuracy: **58%** (41% drop)



# How Vulnerable Are DNNs to a Single Bit-flip?

- **Methodology**
  1) Flip each bit in all parameters of a DNN model
  2) Measure the accuracy over the test-set for each flip
  3) Mark **Achilles bits** – when the bit flips, it causes the **acc. drop > 10%**

# How Vulnerable Are DNNs to a Single Bit-flip?

- **Methodology**
  1) Flip each bit in all parameters of a DNN model
  2) Measure the accuracy over the test-set for each flip
  3) Mark **Achilles bits** – when the bit flips, it causes the **acc. drop > 10%**

- **Quantifying the vulnerability**
  1) **Max. drop** : the maximum acc. drop, observed from a model
  2) **Ratio**. : % of parameters in a model that contains at least one Achilles bit

# MNIST Models

| Network | Acc. | # Params | Acc. Drop | Ratio |
|---|---|---|---|---|
| B(ase) | 95.71 | 21,840 | | |
| B-Wide | 98.46 | 85,670 | | |
| B-PReLU | 98.13 | 21,843 | | |
| B-Dropout | 96.86 | 21,840 | | |
| B-DP-Norm | 97.97 | 21,962 | | |
| L(eNet)5 | 98.81 | 61,706 | | |
| L5-Dropout | 98.72 | 61,706 | | |
| L5-D-Norm | 99.05 | 62,598 | | |

# MNIST Models

| Network | Acc. | # Params | Acc. Drop | Ratio |
|---|---|---|---|---|
| B(ase) | 95.71 | 21,840 | 98 % | |
| B-Wide | 98.46 | 85,670 | 99 % | |
| B-PReLU | 98.13 | 21,843 | 99 % | |
| B-Dropout | 96.86 | 21,840 | 99 % | |
| B-DP-Norm | 97.97 | 21,962 | 99 % | |
| L(eNet)5 | 98.81 | 61,706 | 99 % | |
| L5-Dropout | 98.72 | 61,706 | 99 % | |
| L5-D-Norm | 99.05 | 62,598 | 98 % | |

- Max. drop $\geq$ **98%** in all models

# MNIST Models

| Network | Acc. | # Params | Acc. Drop | Ratio |
|---|---|---|---|---|
| B(ase) | 95.71 | 21,840 | **98 %** | **50%** |
| B-Wide | 98.46 | 85,670 | **99 %** | **50%** |
| B-PReLU | 98.13 | 21,843 | **99 %** | **99%** |
| B-Dropout | 96.86 | 21,840 | **99 %** | **49%** |
| B-DP-Norm | 97.97 | 21,962 | **99 %** | **51%** |
| L(eNet)5 | 98.81 | 61,706 | **99 %** | **47%** |
| L5-Dropout | 98.72 | 61,706 | **99 %** | **45%** |
| L5-D-Norm | 99.05 | 62,598 | **98 %** | **49%** |

- Max. drop $\geq$ **98%** in all models
- **> 45%** of params contain $\geq$ **1 Achilles bit** in all the DNNs

# LARGE, COMPLEX DNN MODELS

| Dataset | Network | Acc. | # Params | Acc. Drop | Ratio |
|---|---|---|---|---|---|
| CIFAR-10 | B(ase) | 83.74 | 776K | 94 % | 46.8% |
| | B-Slim | 82.19 | 197K | 93 % | 46.7% |
| | B-Dropout | 81.18 | 776K | 94 % | 40.5% |
| | B-D-Norm | 80.17 | 778K | 97 % | 45.9% |
| | AlexNet | 83.96 | 2.5M | 96 % | 47.3% |
| | VGG16 | 91.34 | 14.7M | 99 % | 46.2% |
| ImageNet | AlexNet | 79.07 | 61.1M | 100 % | 47.3% |
| | VGG16 | 90.38 | 138.4M | 99 % | 42.1% |
| | ResNet50 | 92.86 | 25.6M | 100 % | 47.8% |
| | DenseNet161 | 93.56 | 28.9M | 100 % | 49.0% |
| | InceptionV3 | 88.65 | 27.2M | 100 % | 40.8% |

- Max. drop $\geq$ **98%** in all models
- **> 45%** of params contain $\geq$ **1 Achilles bit** in all the DNNs

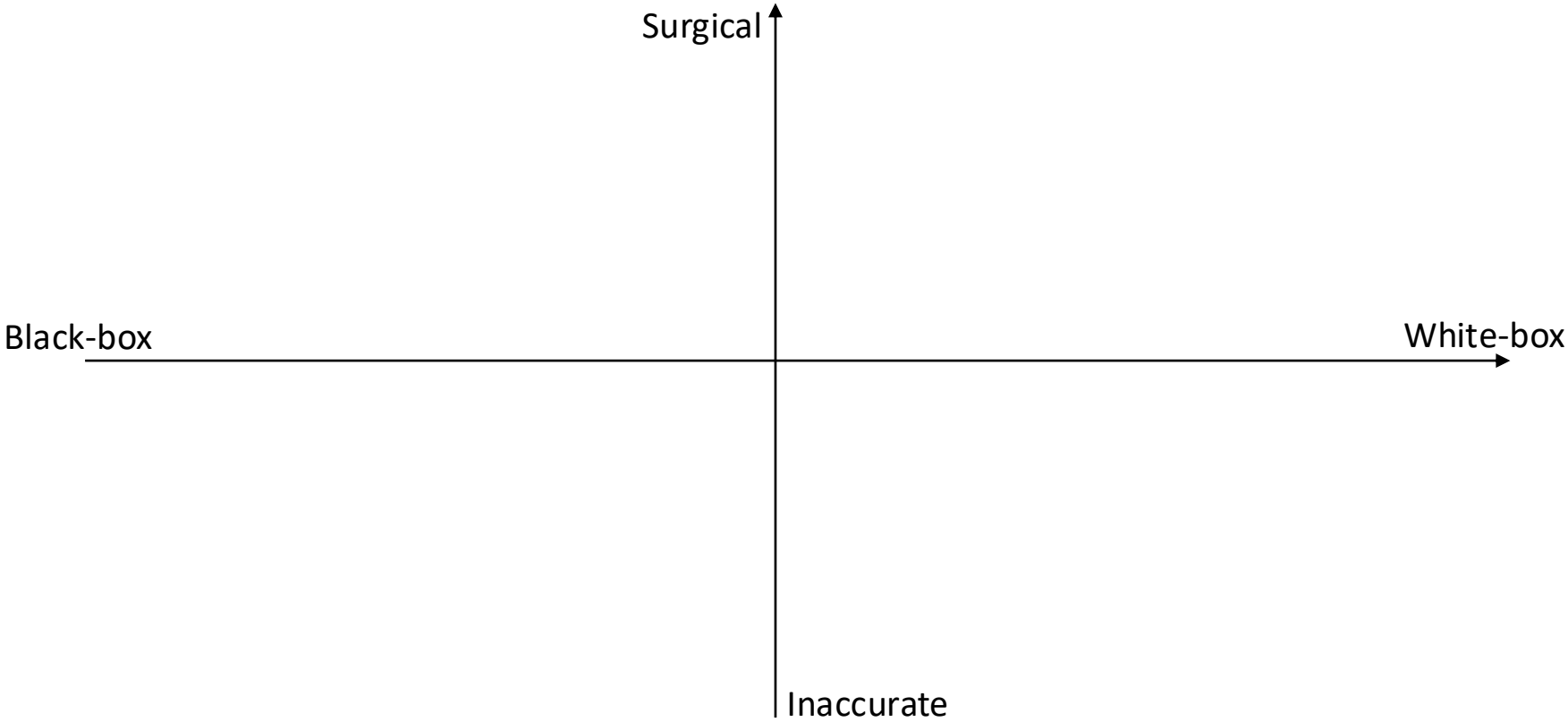
**The Vulnerability of DNNs to A Bit-flip Is Prevalent**

# THREAT MODEL – BIT-FLIP ATTACKER

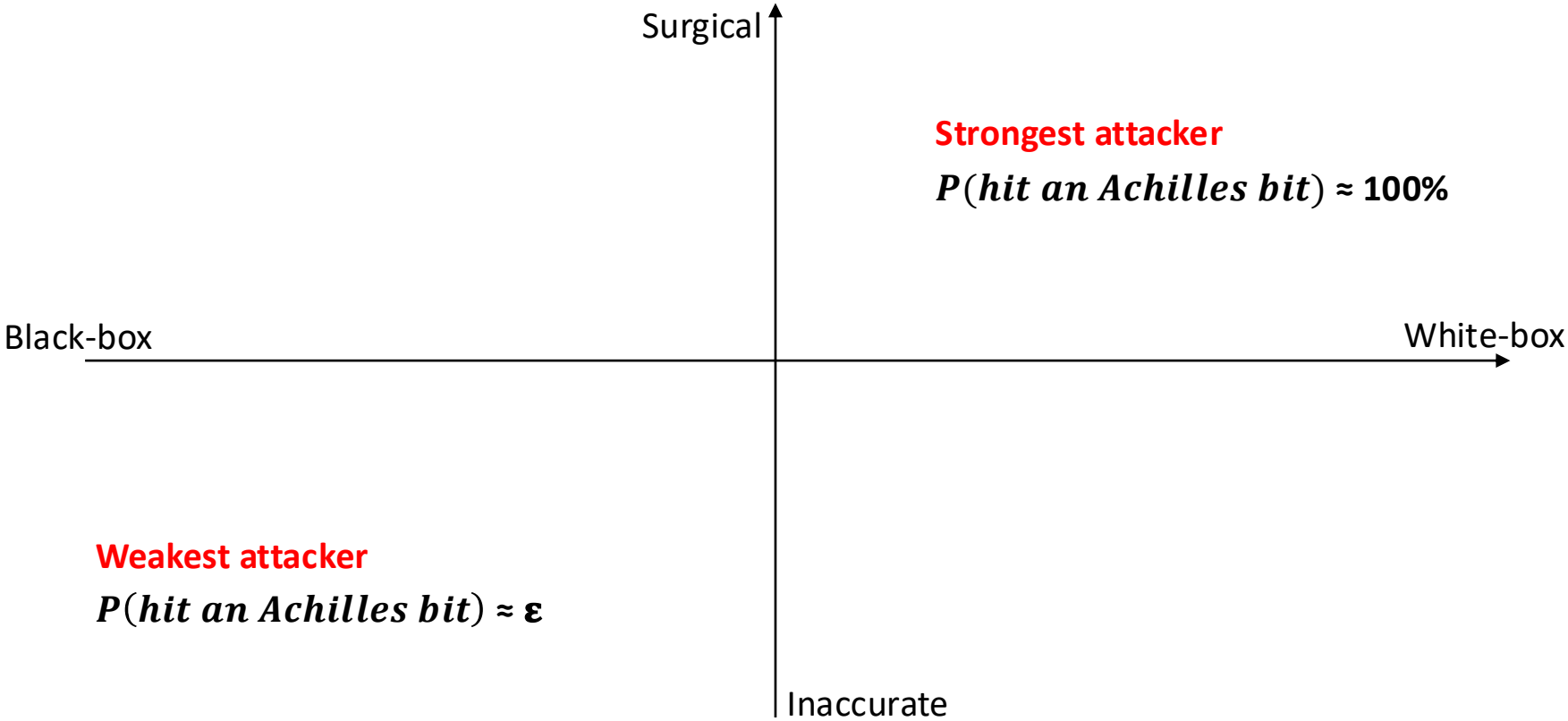
- Capability
  - **Surgical** : can control the location of a bit-flip in memory
  - **Inaccurate**: cannot control the bit-flip location

- Knowledge
  - **White-box**: knows which parameters are vulnerable
  - **Black-box** : has no knowledge of a victim model

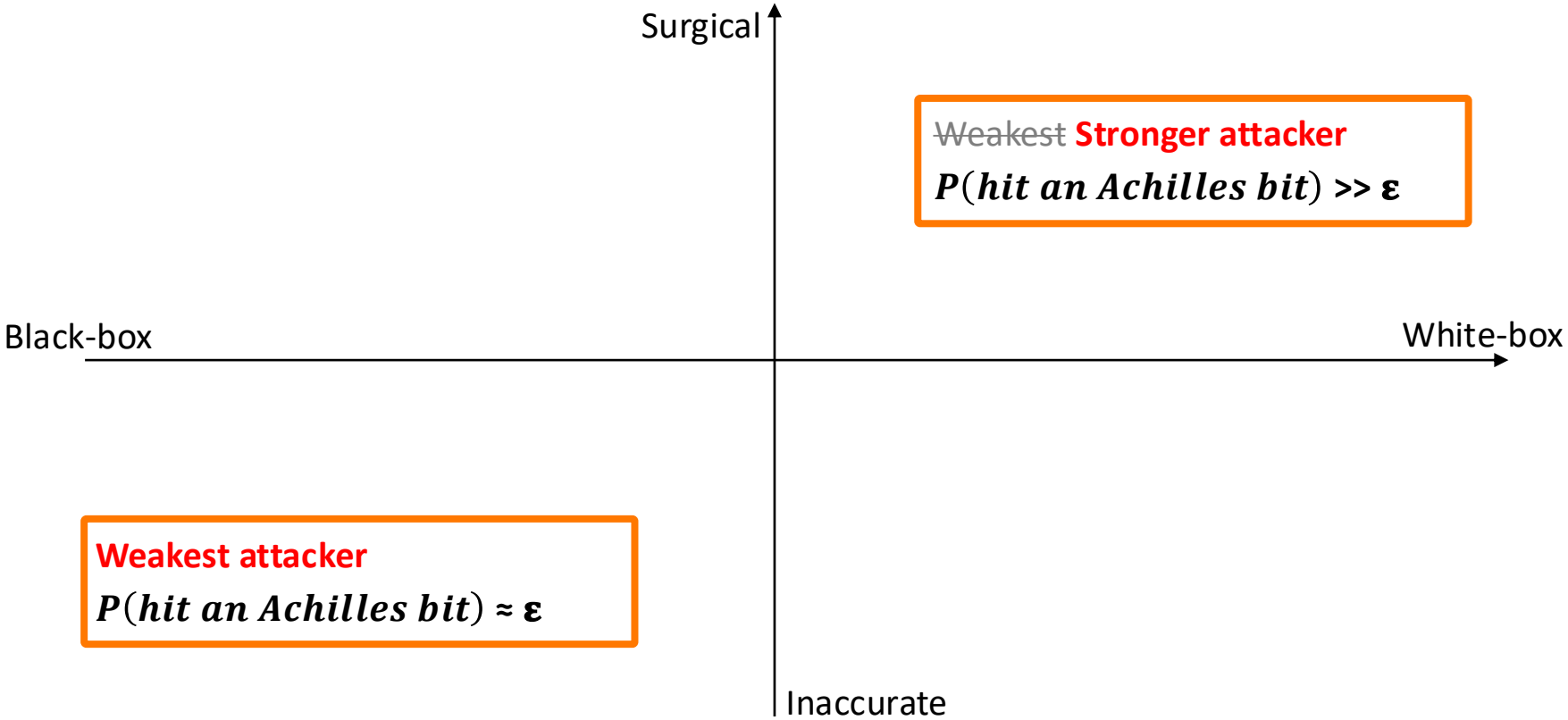
# THREAT MODEL – SINGLE-BIT ADVERSARY



# THREAT MODEL – SINGLE-BIT ADVERSARY



# Threat Model – If the Adversary Can Flip Multiple Bits?



# EVALUATION

- **MLaaS scenario**
  - **Victim** : runs an off-the-shelf model (VGG16) in a VM
  - **Attacker** : runs Rowhammer attacks against the victim's VM

- **Rowhammer** (Hammertime[1] DB)
  - Explore Rowhammer attacks systematically on **12 different DRAM chips**
  - Experiments:
    - ***300** experiments*: ***25** runs × each of 12 DRAM chips*
    - ***7500** bit-flips* : ***300** cumulative bit-flips × **300** experiments*

[1]Tartar et al., *Defeating Software Mitigations against Rowhammer: A Surgical Precision Hammer*, RAID'18

# EVALUATION

- **Results**
  - The weakest attacker can inflict **severe damage** to the victim system
    - On average, **62%** of the attacks cause the **acc. drop > 10%**
    - The time it takes to cause the acc. drop is **< few minutes**
  - Our attack is **inconspicuous**
    - Only 6 program crashes (**0.08%**) were observed over 7500 bit-flip attempts

# TAKEAWAYS

- DNNs are ***not*** resilient to worst-case param. perturbations
  - All DNNs have a bit whose flip causes the accuracy drop up to 100%
  - 40-50% of all parameters in a model are vulnerable

- **The vulnerability of DNNs to fault attacks is under-studied**
  - One can inflict the vulnerability with ***weaker attacks***, *e.g.*, blind Rowhammer
  - The attacker can launch this attack in a practical setting, *e.g.*, in the cloud

- **We need solutions from both systems and ML**
  - **Systems:** defenses that prevent flipping a specific-bit are not sufficient
  - **ML:** future work is required to build DNNs robust against new attacks

# FLUSH+RELOAD: CACHE-BASED TIMING SIDE-CHANNEL

# PRELIMINARIES ON MEMORY ARCHITECTURE

- The X86 cache
  - Memory architecture

# PRELIMINARIES ON MEMORY ARCHITECTURE

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access

# PRELIMINARIES ON CACHE

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses

# PRELIMINARIES ON CACHE

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses
    - x86 system:
      - Divides memory into blocks (= lines in cache)
      - Stores lines recently accessed by a program

# PRELIMINARIES ON CACHE – SHARED BETWEEN CORES

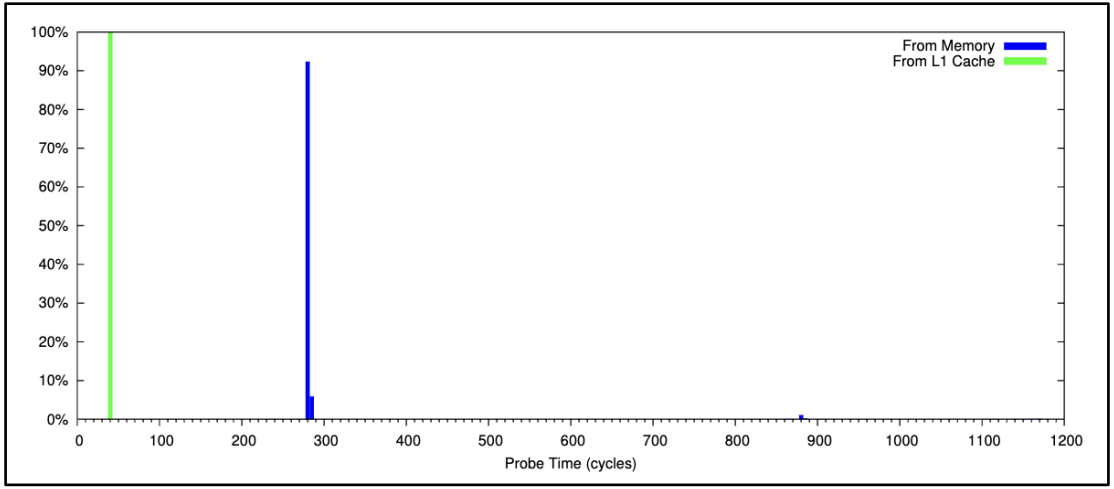
- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses
    - x86 system:
      - Divides memory into blocks (= lines in cache)
      - Stores lines recently accessed by a program
  - The last-layer-cache (LLC: L3) is shared across multiple cores
    - Improve the system performance
    - Think of a shared library in memory used by multiple programs

# PRELIMINARIES ON CACHE – CONSISTENCY

- The X86 cache
  - Memory and cache are often in inconsistent states
  - In case of this cache conflict, system flushes the cache line
    - clflush: one can flush the cache line
    - Think about: what happens if one flushes a cache line intentionally

# FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Exploit the cache flush to leak information on the victim's memory access
  - Assumptions:
    - The victim and the attacker access the shared code (e.g., shared libraries)
    - The victim's process and the attacker's process can be on the same or in different cores
    - The attacker can flush the cache line intentionally

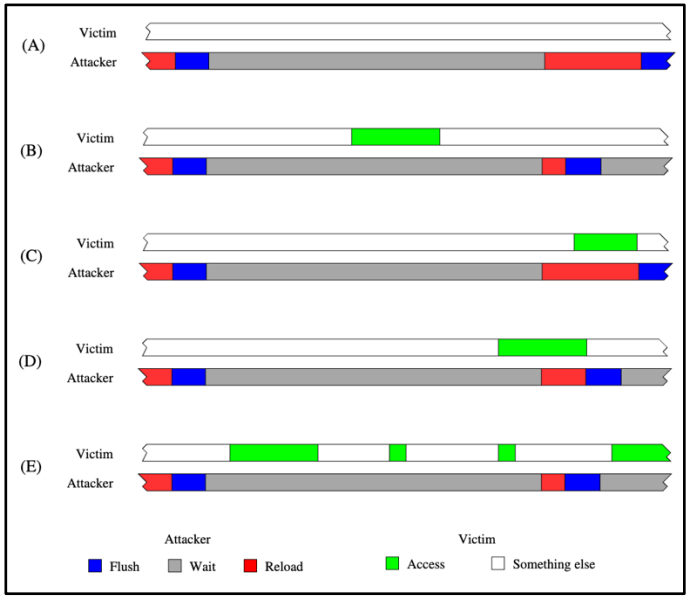


# FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Exploit the cache flush to leak information on the victim's memory access
  - Assumptions:
    - The victim and the attacker access the shared code (e.g., shared libraries)
    - The victim's process and the attacker's process can be on the same or in different cores
    - The attacker can flush the cache line intentionally
  - Flush+Reload procedure
    - Step 1: The attacker first flush the cache line (or lines)
    - Step 2: They will wait for a few cycles (e.g., 2000 CPU cycles)
    - Step 3: They will access the same cache line(s) again
    - Step 4: Measure the time it takes to load the data
      - Slow access: the data has not been accessed by the victim
      - Fast access : the data is accessed by the victim
    - Repeat Step 1-4 forever

# FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Flush+Reload procedure
    - Step 1: The attacker first flush the cache line (or lines)
    - Step 2: They will wait for a few cycles (e.g., 2000 CPU cycles)
    - Step 3: They will access the same cache line(s) again
    - Step 4: Measure the time it takes to load the data
      - Slow access: the data has not been accessed
      - Fast access : the data is accessed by the victim
    - Repeat Step 1-4 forever



# FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation … why?

# FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation
  - Public key: e N
  - Private key: d (that satisfies ed = 1)
  - To ciphertext: $C = M^e \bmod N$
  - To plaintext: $C^d \bmod N$
    - $(M^e)^d \bmod N$
    - $M^{ed} \bmod N$
    - M mod N (N is a really large prime, so mostly it's N)

# FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation
  - Public key: e N
  - Private key: d (that satisfies ed = 1)
  - To ciphertext: $C = M^e \bmod N$
  - To plaintext: $C^d \bmod N$
    - $(M^e)^d \bmod N$
    - $M^{ed} \bmod N$
    - M mod N (N is a really large prime, so mostly it's N)
  - Implementation of the modular exponentiation
    - Square and multiply algorithm (see the right)
    - In here e is equal to d above
    - For clear bits: square – reduce
    - For set bits : square – reduce – multiply – reduce

**Square and multiply algorithm**

```
x ←1
for i ←|e|-1 downto 0 do
    x ←x^2 mod n
    if (e_i =1) then
        x = xb mod n
    endif
done
return x
```

# FLUSH+RELOAD DEMONSTRATION – GNUPG

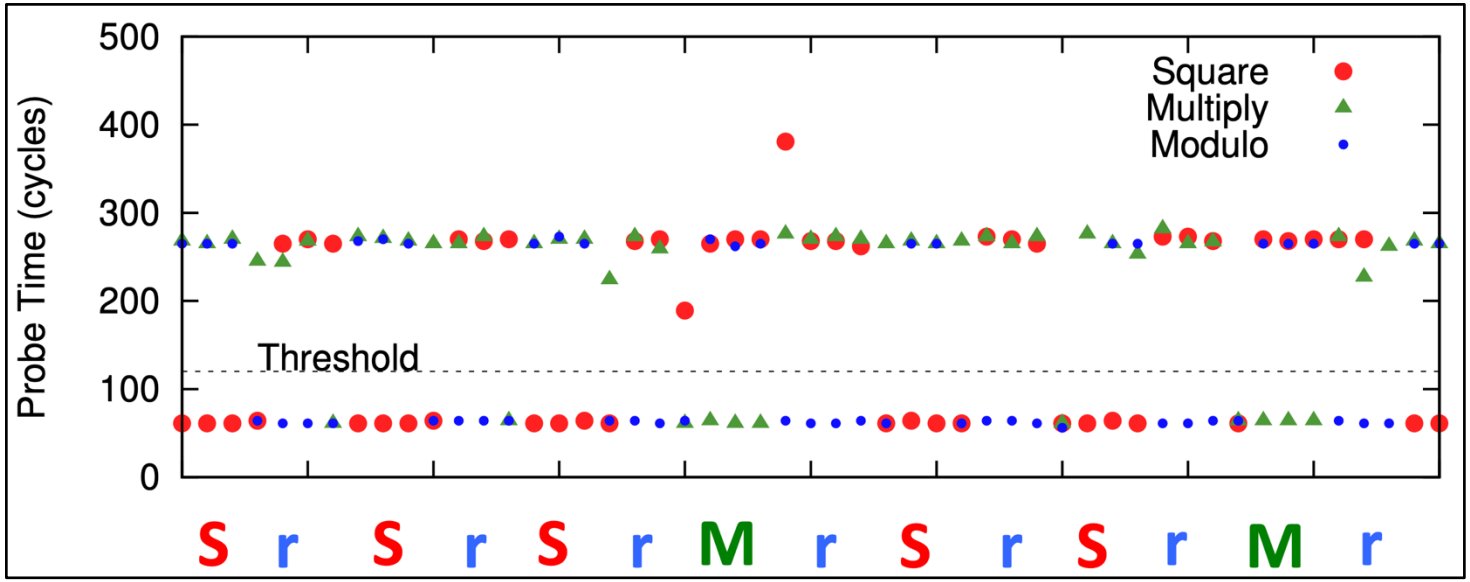
- GnuPG (GPG) operation … why?

# FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - GPG uses the RSA algorithm
    - Encryption and digital signatures
    - 0 bit: square – reduce
    - 1 bit: square – reduce – multiply – reduce

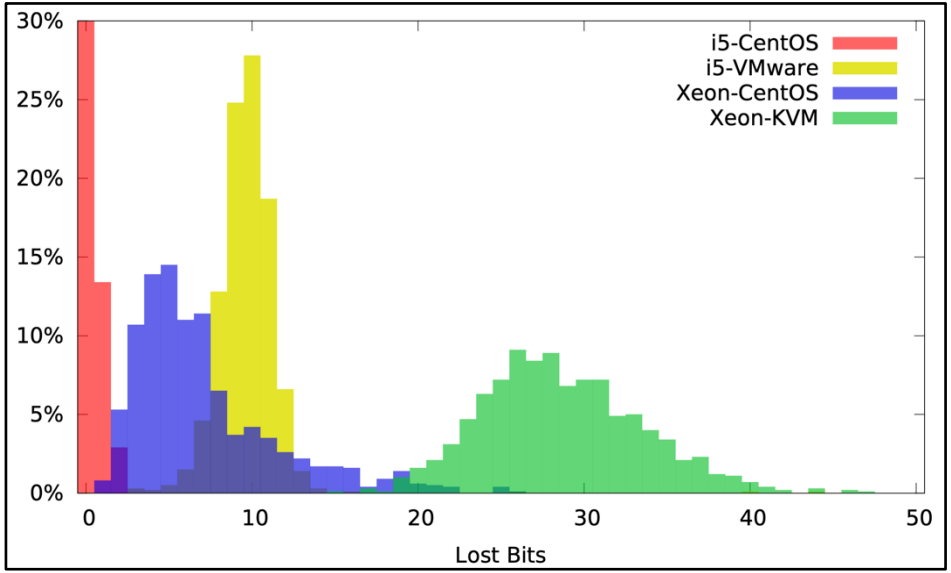
# FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - Run Flush+Reload
    - Extract the sequence of operations of the modular exponentiation
    - Each Flush+Reload attempt is 2048 cycle – reconnaissance



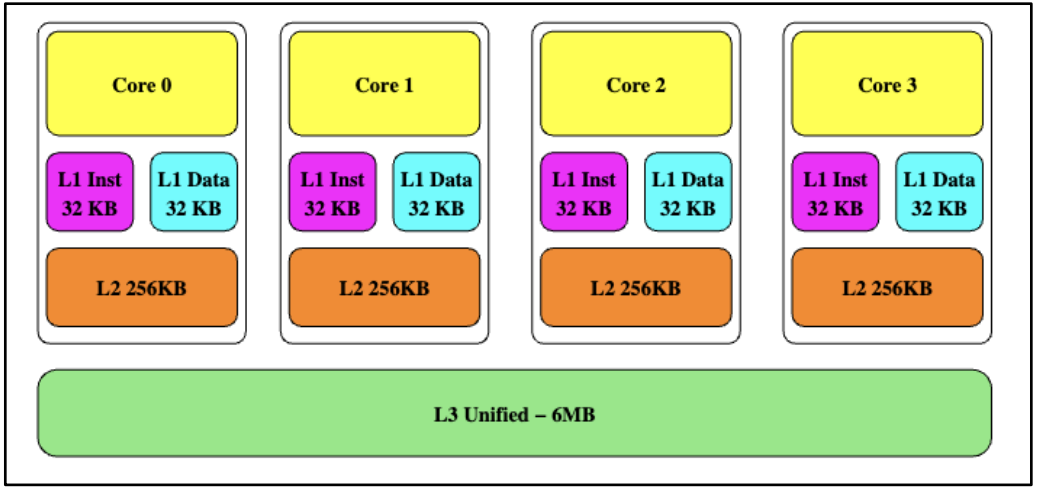
# FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - Run Flush+Reload
    - Extract the sequence of operations of the modular exponentiation
    - Each Flush+Reload attempt is 2048 cycle – reconnaissance
    - The attack can have false negatives



# FLUSH+RELOAD IMPLICATIONS

- Cross-VM attacks
  - The attacker and the victim VMs are on the same host, but on different cores
  - The attacker VM can spy the behaviors of the victim
    - If the attacker knows what software libraries used by the victim
    - It's easier to do that for commodity software



# FLUSH+RELOAD X AI – UNEASY COEXISTENCE

# UNIQUE DNN ARCHITECTURES REPRESENT ATTRACTIVE TARGETS

## Image classification (ImageNet) leaderboard

| Model | Detail | Input size | Top-1 Acc | Top-5 Acc | Param(M) | Mult-Adds | FLOPS(G) |
|---|---|---|---|---|---|---|---|
| EfficientNet-B7 | (2.0, 3.1, 600, 0.5) | 600x600 | 84.4 | 97.1 | 66 | | 37000 |
| GPipe-AmoebaNet-B | (N=6, F=512) | 480x480 | 84.3 | 97 | 557 | | |
| EfficientNet-B6 | (1.8, 2.6, 528, 0.5) | 528x528 | 84 | 96.9 | 43 | | 19000 |
| AmoebaNet-A | (N=6, F=448) | 331x331 | 83.9 | 96.6 | 469 | 104B | |
| EfficientNet-B5 | (1.6, 2.2, 456, 0.4) | 456x456 | 83.3 | 96.7 | 30 | | 9900 |
| AmoebaNet-B | (N=6, F=228) | 331x331 | 83.1 | 96.3 | 155.3 | 41.1B | |
| PNASNet-5_Large_331 | (N=4, F=216) | 331x331 | 82.9 | 96.2 | 86.1 | 25.0B | 25.169 |
| Oct-ResNet-152+SE | α=0.125, test:331 | 224x224 | 82.9 | 96.3 | 66.8 | | 22.2 |
| AmoebaNet-B | (N=6, F=190) | 331x331 | 82.8 | 96.1 | 86.7 | 23.1B | |
| SENet-154 | | 320x320 | 82.7 | 96.2 | 145.8 | 42.3B | |
| NASNet-A_Large_331 | (N=6, F=168) | 331x331 | 82.7 | 96.2 | 88.9 | 23.8B | 24.021 |
| EfficientNet-B4 | (1.4, 1.8, 380, 0.4) | 380x380 | 82.6 | 96.3 | 19 | | 4200 |
| AmoebaNet-B | (N=6, F=190) | 331x331 | 82.3 | 96.1 | 84 | 22.3B | |
| Oct-ResNet-152 | α=0.125, test:331 | 224x224 | 82.3 | 96 | 60.2 | | 22.2 |
| SKNet-101 | | 320x320 | 81.6 | | 48.9 | | 8.46 |
| SENet101 | | 320x320 | 81.6 | | 48.9 | | 8.46 |
| RandWire-WS | C=154 | 320x320 | 81.6 | 95.6 | 61.5 | | 16 |
| Oct-ResNet-152+SE | α=0.125, test:224 | 224x224 | 81.6 | 95.7 | 66.8 | | 10.9 |
| Dual-Path-Net-131 | | 320x320 | 81.5 | 95.8 | 79.5 | 32.0B | |
| Oct-ResNet-152 | α=0.125, test:224 | 224x224 | 81.4 | 95.4 | 60.2 | | 10.9 |
| PolyNet | | 331x331 | 81.3 | 95.8 | 92 | 34.7B | 34.768 |
| SENet-154 | | 224x224 | 81.16 | 95.35 | 115.088 | | 20.742 |
| DPN-98 | | 320x320 | 81.1 | | 61.6 | | 11.7 |
| EfficientNet-B3 | (1.2, 1.4, 300, 0.3) | 300x300 | 81.1 | 95.5 | 12 | | 1800 |
| ResNeXt-101 | (64x4d) | 320x320 | 80.9 | 95.6 | 83.6 | 31.5B | |
| ResNeXt101+BAM | | 320x320 | 80.85 | | 44.6 | | 8.05 |
| PyramidNet-200 | α=450 | 320x320 | 80.8 | 95.3 | 116.4 | | |
| DPN-92 | | 320x320 | 80.7 | | 37.7 | | 6.5 |
| SKNet-50 | | 320x320 | 80.68 | | 27.5 | | 4.47 |

**Top-10 Models (acc.)**

Use unique DNN architectures found by NAS
(≈ **thousands of GPU hours** for searching)

[1]ImageNet classification leaderboard: https://kobiso.github.io/Computer-Vision-Leaderboard/imagenet.html
[2]Yang et al., *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, NeurIPs'19

# UNIQUE DNN ARCHITECTURES REPRESENT ATTRACTIVE TARGETS

Image classification (ImageNet) leaderboard

| Model | Detail | Input size | Top-1 Acc | Top-5 Acc | Param(M) | Mult-A |
|---|---|---|---|---|---|---|
| EfficientNet-B7 | (2.0, 3.1, 600, 0.5) | 600x600 | 84.4 | 97.1 | 66 | |
| GPipe-AmoebaNet-B | (N=6, F=512) | 480x480 | 84.3 | 97 | 557 | |
| EfficientNet-B6 | (1.8, 2.6, 528, 0.5) | 528x528 | 84 | 96.9 | 43 | |
| AmoebaNet-A | (N=6, F=448) | 331x331 | 83.9 | 96.6 | 469 | 104B |
| EfficientNet-B5 | (1.6, 2.2, 456, 0.4) | 456x456 | 83.3 | 96.7 | 30 | |
| AmoebaNet-B | (N=6, F=228) | 331x331 | 83.1 | 96.3 | 155.3 | 41.1B |
| PNASNet-5_Large_331 | (N=4, F=216) | 331x331 | 82.9 | 96.2 | 86.1 | 25.0B |
| Oct-ResNet-152+SE | α=0.125, test:331 | 224x224 | 82.9 | 96.3 | 66.8 | |
| [illegible] | [illegible] | [illegible] | 82.7 | 96.2 | 145.8 | 42.3B |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 88.9 | 23.8B |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 19 | |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 42.3B |
| SKNet-101 | | 320x320 | 81.8 | | 48.9 | |
| SENet101 | | 320x320 | 81.6 | | 48.9 | |
| RandWire-WS | C=154 | 320x320 | 81.6 | 95.6 | 61.5 | |
| Oct-ResNet-152+SE | α=0.125, test:224 | 224x224 | 81.6 | 95.7 | 66.8 | |
| Dual-Path-Net-131 | | 320x320 | 81.5 | 95.8 | 79.5 | 32.0B |
| Oct-ResNet-152 | α=0.125, test:224 | 224x224 | 81.4 | 95.4 | 60.2 | |
| PolyNet | | 331x331 | 81.3 | 95.8 | 92 | 34.7B |
| SENet-154 | | 224x224 | 81.16 | 95.35 | 115.088 | |
| DPN-98 | | 320x320 | 81.1 | | 61.6 | |
| EfficientNet-B3 | (1.2, 1.4, 300, 0.3) | 300x300 | 81.1 | 95.5 | 12 | |
| ResNeXt-101 | (64x4d) | 320x320 | 80.9 | 95.6 | 83.6 | 31.5B |
| ResNeXt101+BAM | | 320x320 | 80.85 | | 44.6 | |
| PyramidNet-200 | α=450 | 320x320 | 80.8 | 95.3 | 116.4 | |
| DPN-92 | | 320x320 | 80.7 | | 37.7 | |
| SKNet-50 | | 320x320 | 80.68 | | 27.5 | |

**Top-10 Models (score)**

**5-10 ML experts** require to design those unique DNN architectures for language tasks

**Language task (GLUE) leaderboard**

GLUE SuperGLUE Paper </> Code Tasks Leaderboard

| Rank | Name | Model | URL | Score | CoLA | SST-2 | MRPC |
|---|---|---|---|---|---|---|---|
| 1 | ERNIE Team - Baidu | ERNIE | | 90.9 | 74.4 | 97.8 | 93.9/91.8 |
| 2 | DeBERTa Team - Microsoft | DeBERTa / TuringNLRv4 | | 90.8 | 71.5 | 97.5 | 94.0/92.0 |
| 3 | HFL iFLYTEK | MacALBERT + DKM | | 90.7 | 74.8 | 97.0 | 94.5/92.6 |
| 4 | Alibaba DAMO NLP | StructBERT + TAPT | | 90.6 | 75.3 | 97.3 | 93.9/91.9 |
| 5 | PING-AN Omni-Sinitic | ALBERT + DAAF + NAS | | 90.6 | 73.5 | 97.2 | 94.0/92.0 |
| 6 | T5 Team - Google | T5 | | 90.3 | 71.6 | 97.5 | 92.8/90.4 |
| 7 | Microsoft D365 AI & MSR AI & GATECH | MT-DNN-SMART | | 89.9 | 69.5 | 97.5 | 93.7/91.6 |
| 8 | Huawei Noah's Ark Lab | NEZHA-Large | | 89.8 | 71.7 | 97.3 | 93.3/91.0 |
| 9 | Zihang Dai | Funnel-Transformer (Ensemble B10-10-10H1024) | | 89.7 | 70.5 | 97.5 | 93.4/91.2 |
| 10 | ELECTRA Team | ELECTRA-Large + Standard Tricks | | 89.4 | 71.7 | 97.1 | 93.1/90.7 |

[1]ImageNet classification leaderboard: https://kobiso.github.io/Computer-Vision-Leaderboard/imagenet.html
[2]Yang et al., *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, NeurIPs'19

# UNIQUE DNN ARCHITECTURES REPRESENT ATTRACTIVE TARGETS

**Cloud Services Available for Users to Construct Unique DNN Architectures**

Google's AutoML

Amazon SageMaker

Microsoft Azure ML

# Unique DNN Architectures Represent Attractive Targets

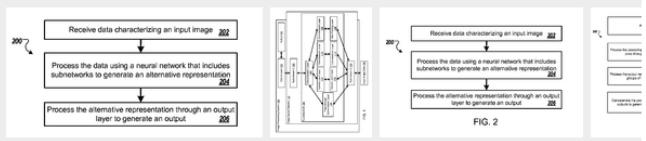
## Processing images using deep neural networks

US20160063359A1

### Abstract

Methods, systems, and apparatus, including computer programs encoded on computer storage media, for image processing using deep neural networks. One of the methods includes receiving data characterizing an input image; processing the data characterizing the input image using a deep neural network to generate an alternative representation of the input image, wherein the deep neural network comprises a plurality of subnetworks, wherein the subnetworks are arranged in a sequence from lowest to highest, and wherein processing the data characterizing the input image using the deep neural network comprises processing the data through each of the subnetworks in the sequence; and processing the alternative representation of the input image through an output layer to generate an output from the input image.

### Images (4)




### Classifications

G06K9/66 Methods or arrangements for recognition using electronic means using simultaneous comparisons or correlations of the image signals with a plurality of references, e.g. resistor matrix references adjustable by an adaptive method, e.g. learning

View

## Data pipeline and deep learning system for autonomous driving

US20190391587A1
United States

Download PDF | Find Prior Art | Similar

**Inventor:** Timofey Uvarov, Brijesh Tripathi, Evgene FAINSTAIN
**Current Assignee :** Tesla Inc

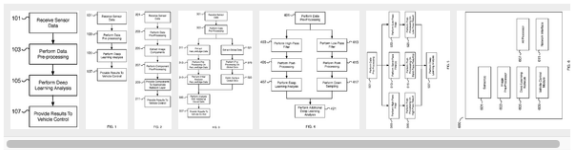
**Worldwide applications**
2018 · US 2019 · WO KR

**Application US16/013,817 events**
2018-06-20 • Application filed by Tesla Inc
2018-06-20 • Priority to US16/013,817
2019-01-14 • Assigned to TESLA, INC.
2019-12-26 • Publication of US20190391587A1
Status • Pending

### Abstract

An image captured using a sensor on a vehicle is received and decomposed into a plurality of component images. Each component image of the plurality of component images is provided as a different input to a different layer of a plurality of layers of an artificial neural network to determine a result. The result of the artificial neural network is used to at least in part autonomously operate the vehicle.

### Images (7)



### Classifications

G05D1/0221 Control of position or course in two dimensions specially adapted to land vehicles with means for defining a desired trajectory involving a learning process

View 12 more classifications

Show all events

External links: USPTO, USPTO Assignment, Espacenet, Global Dossier, Discuss

**They Become Intellectual Property or Trade Secret**

# STEALING THEM GIVES THE ATTACKER AN EDGE FOR FREE

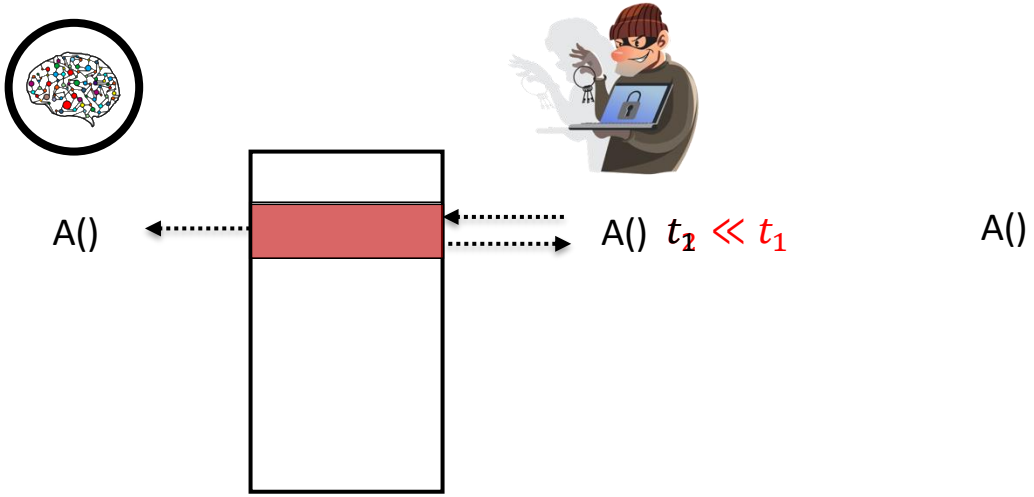
- The attacker can:
  1) Train a fully-functional model that has the **SoTA accuracy**
  2) Train a model on **a different dataset**[1] and **achieve high accuracy**
  3) Exploit the vulnerabilities[2] **specific to the victim's architecture**

[1]So et al., *Evolved Transformer,* ICML19
[2]Xiao et al., *Seeing Is Not Believing: Camouflage Attacks on Image Scaling Algorithm,* USENIX'19

# Can an Attacker Steal Unique Architectures in Practice?

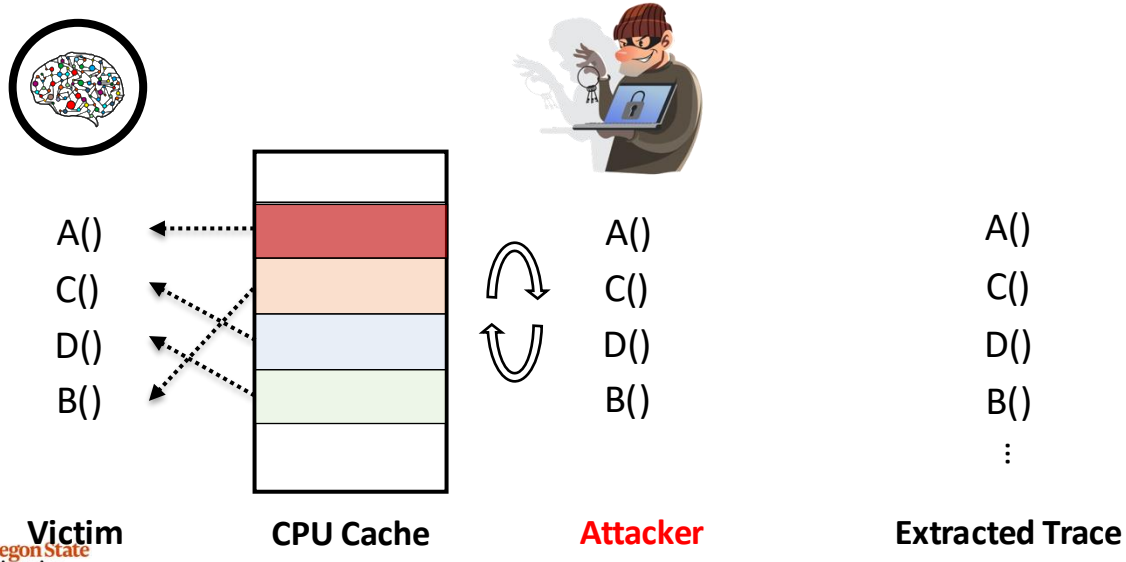
# PRACTICAL WEAPON – FLUSH+RELOAD

- Flush+Reload: a cache side-channel attack
  - Software-induced hardware attack
  - Retrieves the residual information in the cache

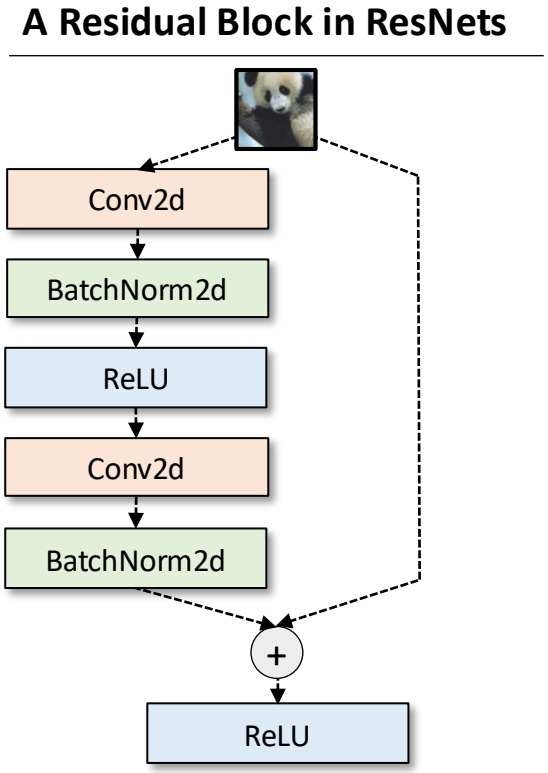


# PRACTICAL WEAPON – FLUSH+RELOAD

- Flush+Reload: a cache side-channel attack
  - Software-induced hardware attack
  - Retrieves the residual information in the cache
  - Cross-VM: attacker only requires a co-located VM

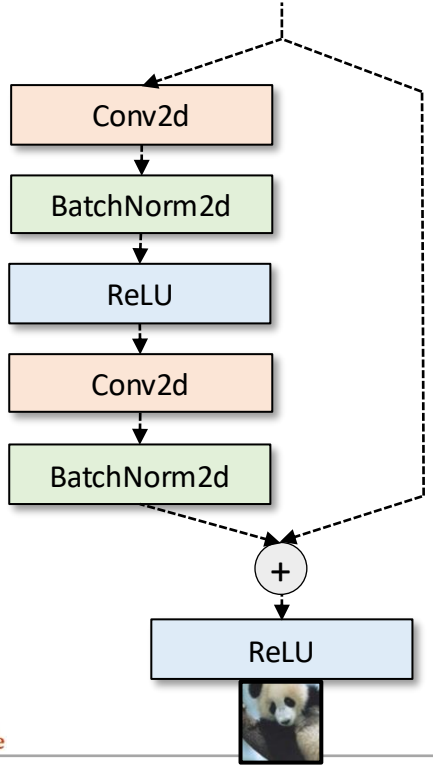


# A Computational Trace of A DNN Observed via Flush+Reload



# A Computational Trace of A DNN Observed via Flush+Reload

## A Residual Block in ResNets

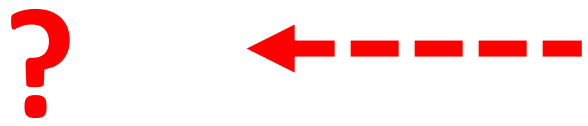



## Flush+Reload Trace

| | |
|---|---|
| $T_1$ | Conv2d |
| $T_2$ | GEMM (conv) |
| $T_3$ | GEMM (oncopy) |
| $T_4$ | GEMM (itcopy) |
| ... | |
| $T_4$ | BatchNorm2d |
| $T_5$ | ReLU |
| $T_6$ | Conv2d |
| $T_7$ | GEMM (conv) |
| $T_8$ | GEMM (oncopy) |
| $T_9$ | GEMM (itcopy) |
| ... | |
| $T_{10}$ | BatchNorm2d |
| $T_{11}$ | add |
| $T_{12}$ | ReLU |

# Our Attack: Reconstruction of DNN Architectures from the Trace

**A Residual Block in ResNets**




**Flush+Reload Trace**

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
...
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
...
$T_{10}$ BatchNorm2d
$T_{11}$ add
$T_{12}$ ReLU

# Our Attack: Reconstruction of DNN Architectures from the Trace

**A Residual Block in ResNets**

① **A computational graph**
② **Layer configurations (kernel size…)**

**Flush+Reload Trace**

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
…
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
…
$T_{10}$ BatchNorm2d
$T_{11}$ add
$T_{12}$ ReLU

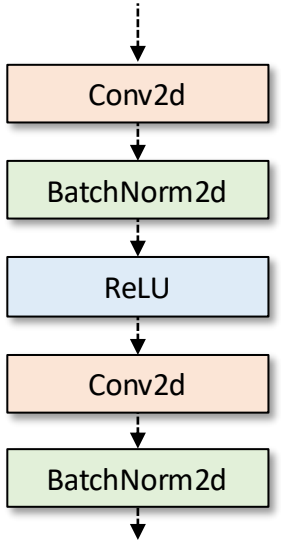
**Prior work**[1]

Assumes the attacker already knew the victim's architecture family, *e.g.,* ResNets

[1]**Hong et al.** *Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks* [Pre-print'18]

# STEP 1: A DNN COMPUTES LAYERS SEQUENTIALLY

## A Residual Block for ResNets

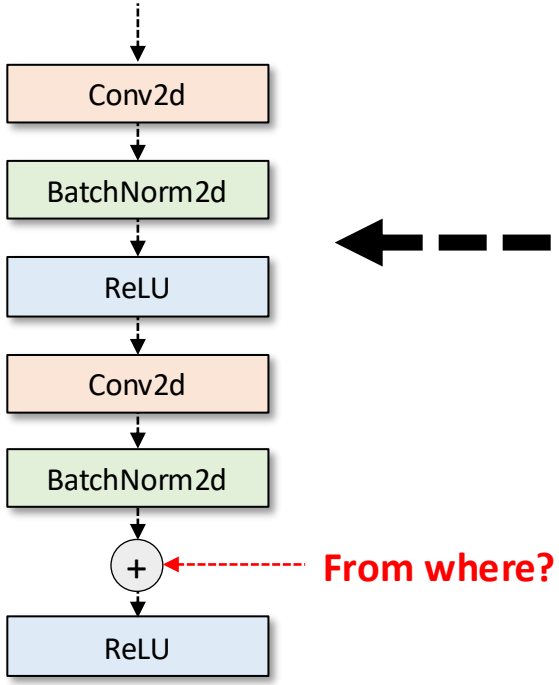



## Flush+Reload Trace

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
Conv2d
...
$T_4$ **BatchNorm2d**
$T_5$ **ReLU**
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
Conv2d
...
$T_{10}$ **BatchNorm2d**
$T_{11}$ **add**
$T_{12}$ **ReLU**

# STEP 1: A DNN COMPUTES LAYERS SEQUENTIALLY

## A Residual Block for ResNets

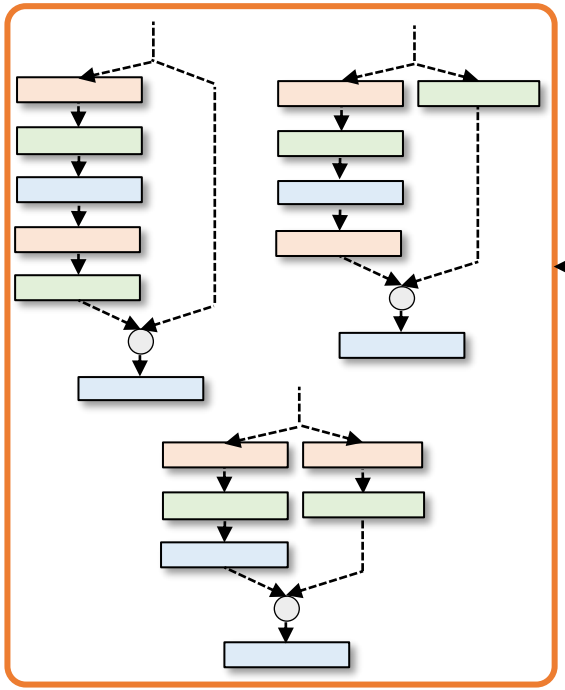



## Flush+Reload Trace

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
Conv2d
...
$T_4$ **BatchNorm2d**
$T_5$ **ReLU**
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
Conv2d
...
$T_{10}$ **BatchNorm2d**
$T_{11}$ **add**
$T_{12}$ **ReLU**

# STEP 2: GENERATE ALL CANDIDATE ARCHITECTURES

**Candidate Architectures**

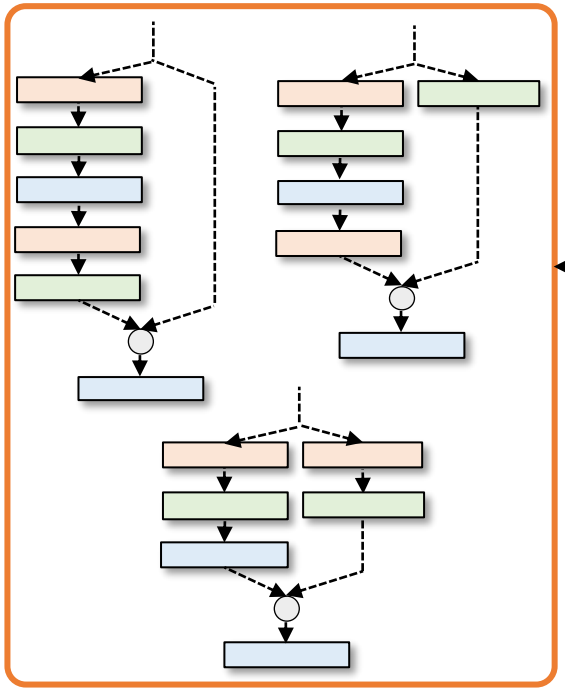


**Flush+Reload Trace**

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
...
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
...
$T_{10}$ BatchNorm2d
$T_{11}$ **add**
$T_{12}$ ReLU

# STEP 3: ELIMINATE INCOMPATIBLE CANDIDATES

**Candidate Architectures**

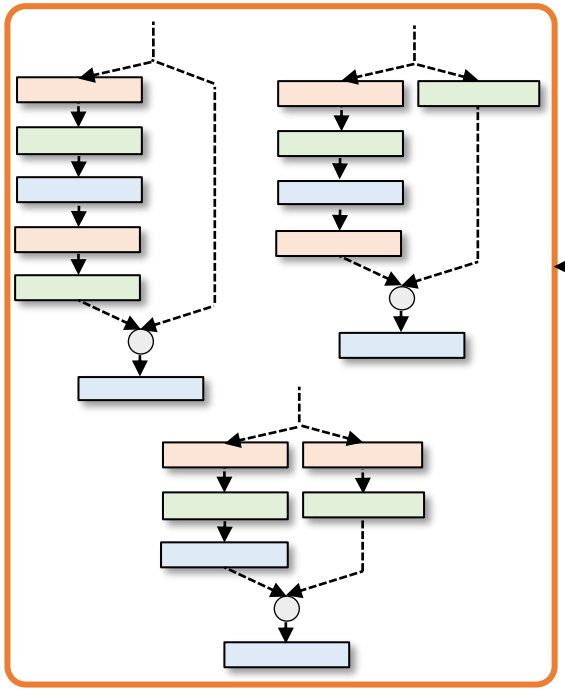


**Flush+Reload Trace**

$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
...
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
...
$T_{10}$ BatchNorm2d
$T_{11}$ add
$T_{12}$ ReLU

# Step 3-1: The Computation Time ∝ The Size of The Layer

**Candidate Architectures**



**Flush+Reload Trace**

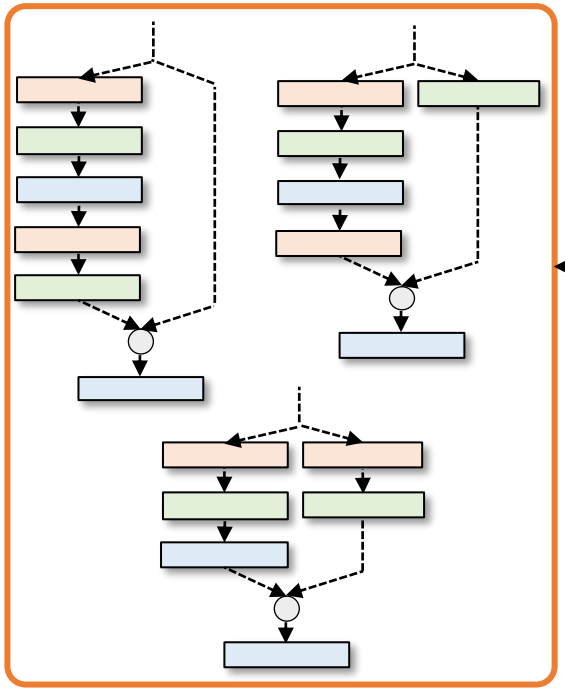
$T_1$ Conv2d
$T_2$ GEMM (conv)
$T_3$ GEMM (oncopy)
$T_4$ GEMM (itcopy)
...
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ GEMM (conv)
$T_8$ GEMM (oncopy)
$T_9$ GEMM (itcopy)
...
$T_{10}$ BatchNorm2d
$T_{11}$ add
$T_{12}$ ReLU

$T_4 - T_1$

$T_5 - T_4$

# STEP 3-2: COMBINE ADDITIONAL INFORMATION

**Candidate Architectures**

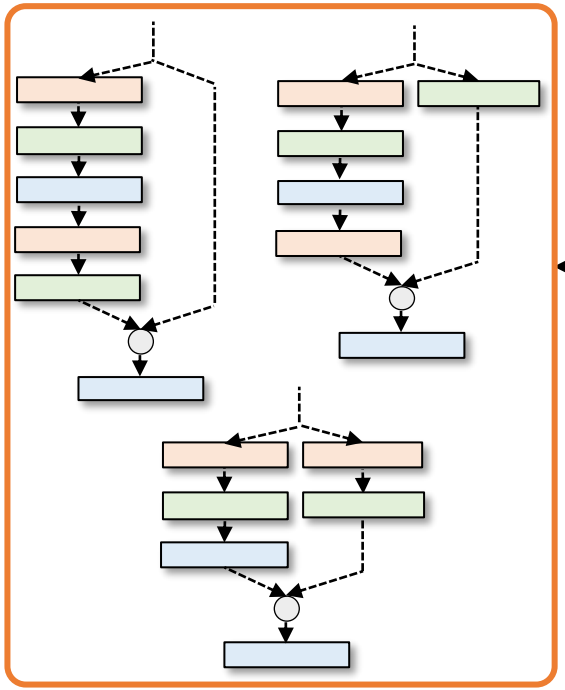


**Flush+Reload Trace**

$T_1$ Conv2d
$T_2$ **GEMM (conv)**
$T_3$ **GEMM (oncopy)**
$T_4$ **GEMM (itcopy)**
...
$T_4$ BatchNorm2d
$T_5$ ReLU
$T_6$ Conv2d
$T_7$ **GEMM (conv)**
$T_8$ **GEMM (oncopy)**
$T_9$ **GEMM (itcopy)**
...
$T_{10}$ BatchNorm2d
$T_{11}$ add
$T_{12}$ ReLU

# STEP 3: ELIMINATE INCOMPATIBLE CANDIDATES

**Candidate Architectures**

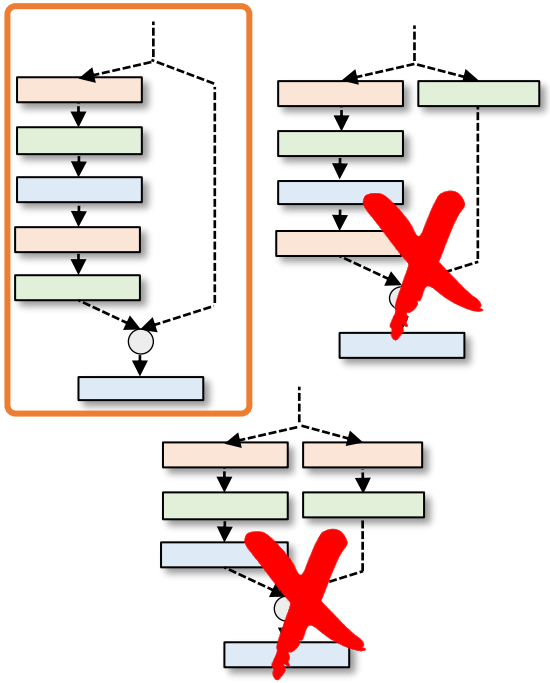


**Flush+Reload Trace**

| | |
|---|---|
| [1] Conv2d, | $i_1, o_1, k_1, s_1, p_1, \ldots$ |
| [2] BatchNorm2d, | $i_2, o_2$ |
| [3] ReLU, | $i_3, o_3$ |
| [4] Conv2d, | $i_4, o_4, k_4, s_4, p_4, \ldots$ |
| [5] BatchNorm2d, | $i_5, o_5$ |
| [6] add, | $i_6, o_6$ |
| [7] ReLU, | $i_7, o_7$ |

# STEP 4: ELIMINATE INCOMPATIBLE CANDIDATES

## Candidate Architectures



## Flush+Reload Trace

[1] Conv2d, $t_1, i_1, o_1, k_1, s_1, p_1, \ldots$
[2] BatchNorm2d, $t_2, i_2, o_2$
[3] ReLU, $t_3, i_3, o_3$
[4] Conv2d, $t_4, i_4, o_4, k_4, s_4, p_4, \ldots$
[5] BatchNorm2d, $t_5, i_5, o_5$
[6] add, $t_6, i_6, o_6$
[7] ReLU, $t_7,$

# EVALUATION

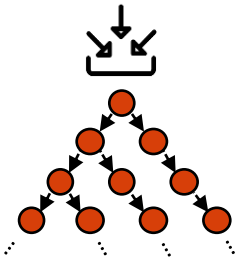
- **Setup**
  - **MLaaS scenario:**
    - A VM runs a unique DNN architecture
    - Our attacker monitors the VM via Flush+Reload
  - **Unique DNNs:**
    - **MalConv:** a novel data pre-processing arch.
    - **ProxylessNAS-CPU:** a novel DNN arch.

# EVALUATION

- **Setup**
  - **MLaaS scenario:**
    - A VM runs a unique DNN architecture
    - Our attacker monitors the VM via Flush+Reload
  - **Unique DNNs:**
    - **MalConv:** a novel data pre-processing arch.
    - **ProxylessNAS-CPU:** a novel DNN arch.
- **Results**
  - Reconstruct both the architectures *correctly*
  - Reconstruction is cheaper than construction:
    - **MalConv:** *a few minutes*
    - **ProxylessNAS-CPU:** *12 CPU hours* < 40k GPU hours for running NAS

# TAKEAWAY: DNNS ARE *MORE* VULNERABLE TO RECONSTRUCTION ATTACKS

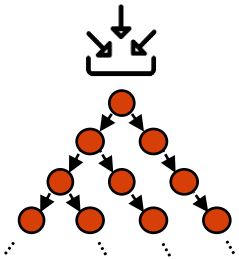
**Traditional software** (*e.g.*, MS Word)



- Not open-source software
  - Difficult to monitor computations
- Rarely repeat the same computations
  - Most inputs are computed differently
  - $\mathrm{N}(\mathrm{inputs}) \gg 1$ (for reconstructions)

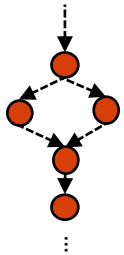
# TAKEAWAY: DNNS ARE *MORE* VULNERABLE TO RECONSTRUCTION ATTACKS

**Traditional software** (*e.g.*, MS Word)



- Not open-source software
  - Difficult to monitor computations
- Rarely repeat the same computations
  - Most inputs are computed differently
  - $\mathrm{N}(\mathrm{inputs}) \gg 1$ (for reconstructions)

**DNNs**



- *Open-source* software
  - Easy to know its internals
- Use the same computations
  - For all the inputs
  - $\mathrm{N}(\mathrm{inputs}) = 1$ (for reconstructions)

# Thank You!

Sanghyun Hong

https://secure-ai.systems/courses/Sec-Grad/current


SAIL
Secure AI Systems Lab