NDSS
SYMPOSIUM/2025

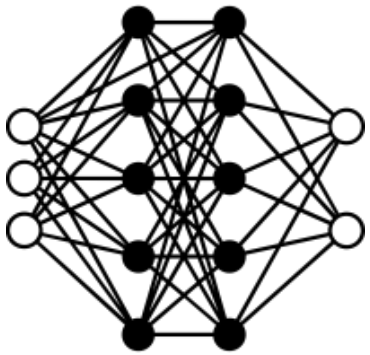
Compiled Models, Built-In Exploits: Uncovering Pervasive Bit-Flip Attack Surfaces in DNN Executables

Yanzuo Chen[†], Zhibo Liu[†], Yuanyuan Yuan[†], Sihang Hu[‡], Tianxiang Li[‡], Shuai Wang[†]

[†]The Hong Kong University of Science and Technology,
[‡]Huawei Technologies

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

This Talk, Distilled



Compile

DNN Models

DNN Executables

This Talk, Distilled

This Talk, Distilled

DNN Executables

This Talk, Distilled

DNN Executables

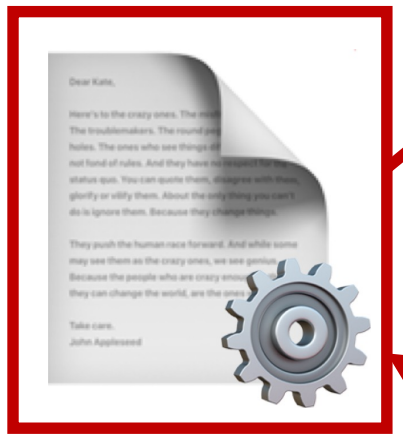
Ninja in camouflage	**95%**
Spooky ghost	4%
Professional chef	1%

This Talk, Distilled

DNN Executables

- Bit flips via hardware fault injection
  - e.g., Rowhammer

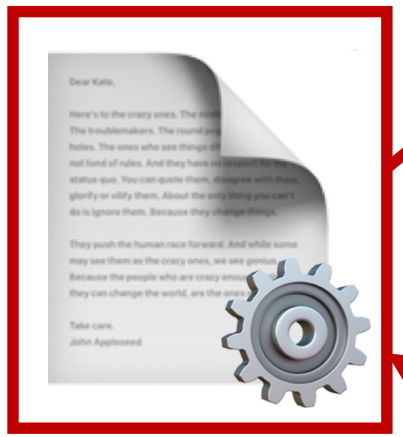
This Talk, Distilled



DNN Executables

- Bit flips via hardware fault injection
  - e.g., Rowhammer
- Pervasive in DNN executables

This Talk, Distilled



DNN Executables

- Bit flips via hardware fault injection
  - e.g., Rowhammer
- Pervasive in DNN executables
- Gray-box, restricted attacker

This Talk, Distilled

Ninja in camouflage	**95%**
Spooky ghost	4%
Professional chef	1%

DNN Executables

- Bit flips via hardware fault injection
  - e.g., Rowhammer
- Pervasive in DNN executables
- Gray-box, restricted attacker
- 70% confidence; ~1.4 flips to succeed

Preliminaries

DNN Executables

DNN Executables

- Compiled from DNN models
  - By “deep learning (DL) compilers”

DNN Executables

- Compiled from DNN models
  - By “deep learning (DL) compilers”
- Wanted for their performance
  - Optimised at the computational graph level
  - Optimised for the target hardware platform

Bit-Flip Attacks (BFAs)

- Flipping data bits in the memory (DRAM)

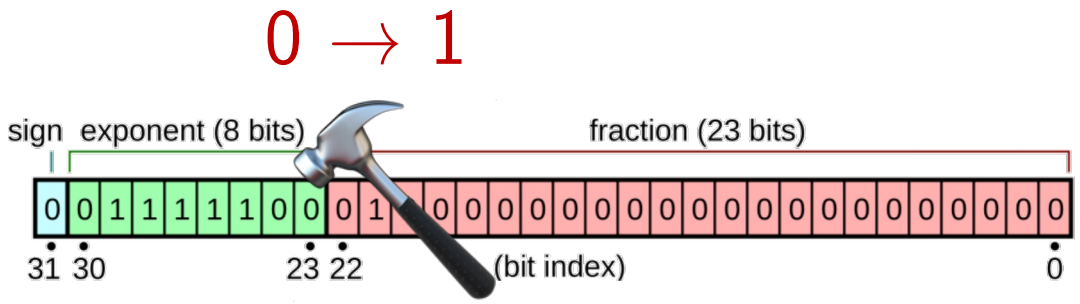
- Rowhammer: “Hardware fault injection” attack
  - Software-triggered hardware bug: Special access patterns
  - Current leakage between DRAM cells
  - DDR3✓ DDR4✓ ECC✓ DDR5✓...

$0 \rightarrow 1$
$1 \rightarrow 0$

BFAs on DNNs: Existing Work

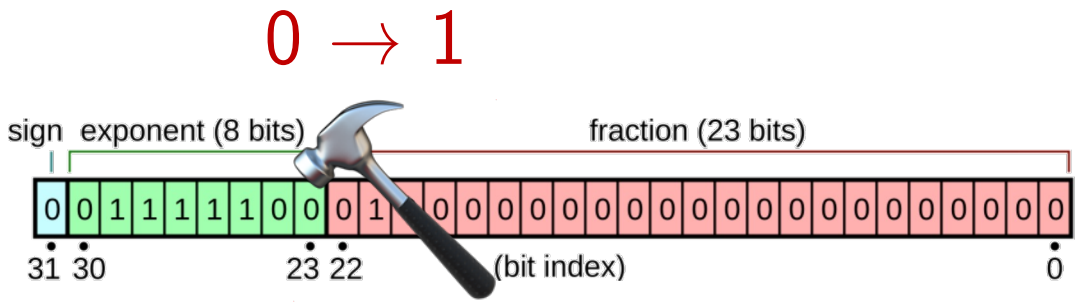
BFAs on DNNs: Existing Work

- Yes, researchers have done this
  - Flip bits in model weights
  - ⇒ A type of white-box attack



BFAs on DNNs: Existing Work

- Yes, researchers have done this
  - Flip bits in model weights
  - ⇒ A type of white-box attack



- Few flips for full-precision models
- 12~24 flips for quantised models
  - (Not anymore!)

Work	Avg. #Flips
BFA [69]	14.3
T-BFA (N-to-1) [71]	23.63
DeepHammer [88]	12.25
Ours	1.4

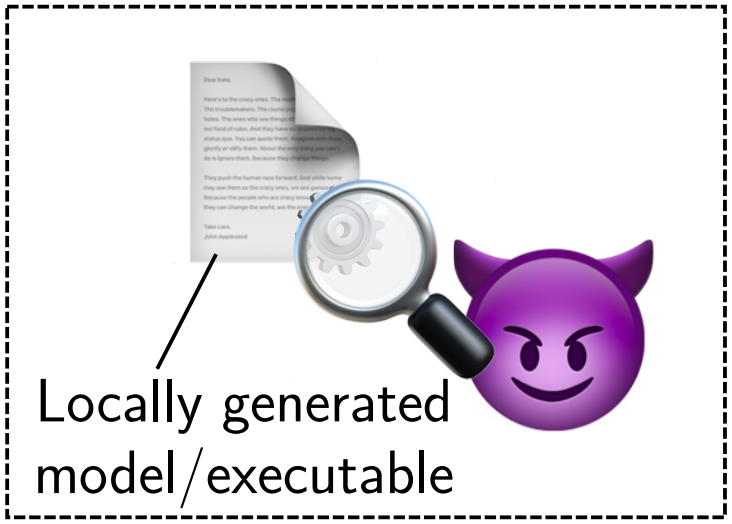
Threat Model & Motivation

Attacker Objectives

- Model intelligence depletion

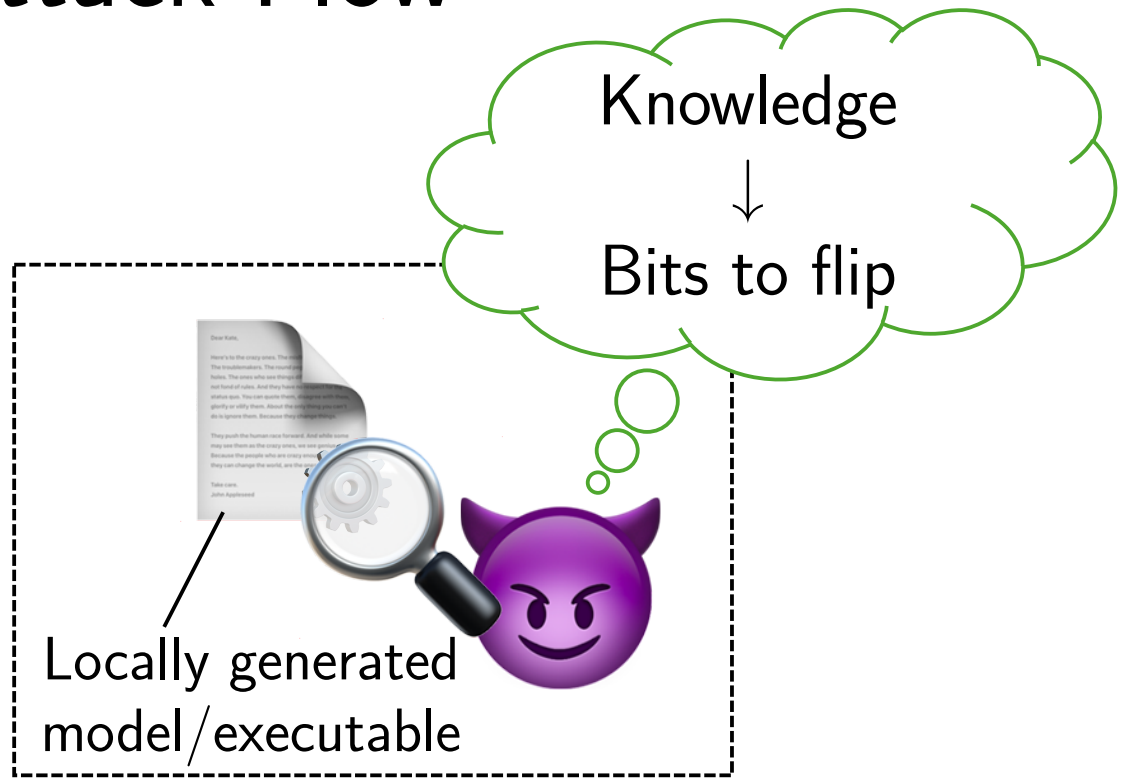
- Classification models (most frequently targeted)
  - Before attack: well-trained models, normal accuracy
  - After attack: random guessers (acc $\rightarrow 1/\#$classes)

Attack Flow

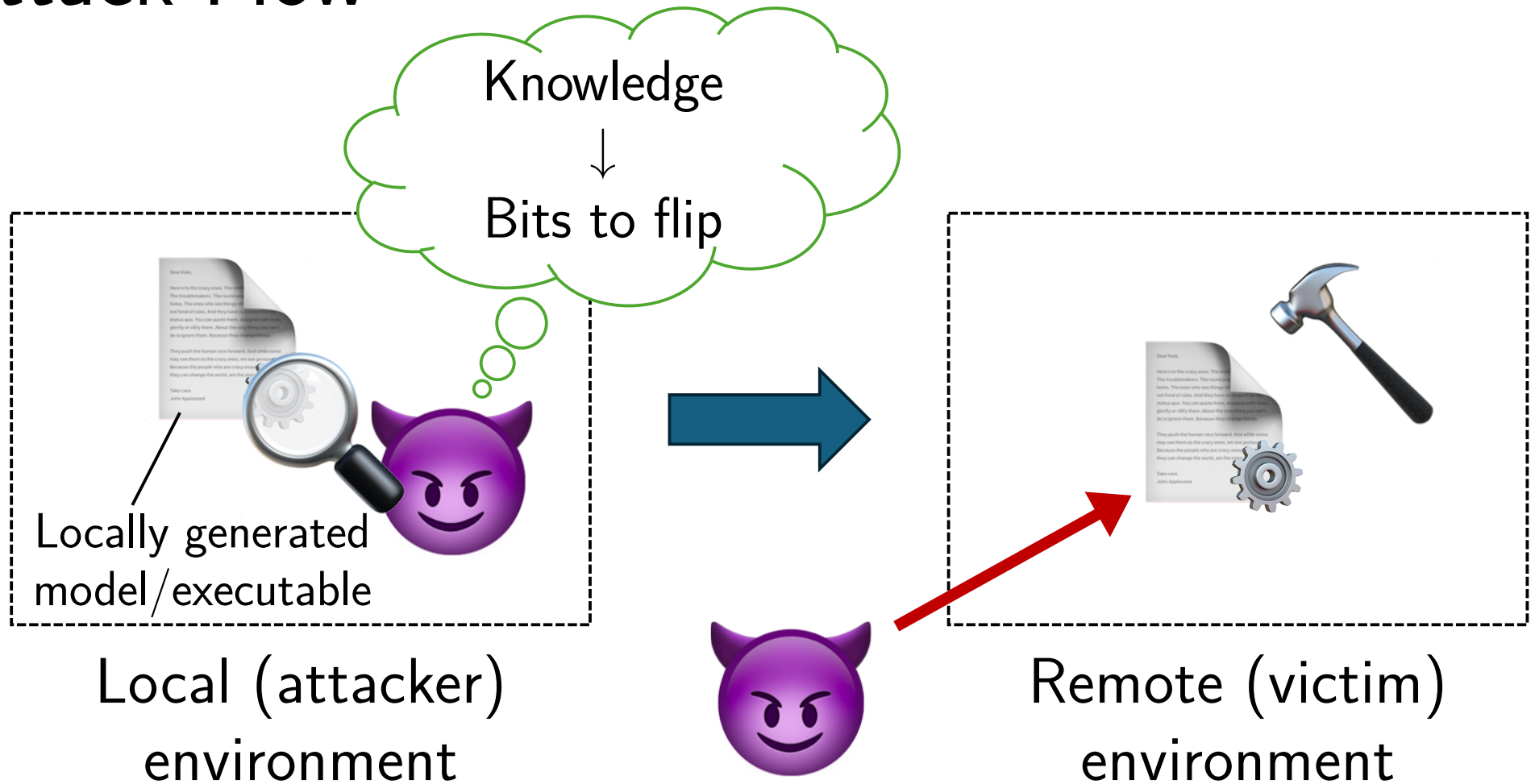


Local (attacker)
environment

Attack Flow


Knowledge
↓
Bits to flip
Locally generated
model/executable
Local (attacker)
environment

Attack Flow


Knowledge
↓
Bits to flip
Locally generated
model/executable
Local (attacker)
environment
Remote (victim)
environment

A More Restricted Attacker

A More Restricted Attacker

- Recall: White-box attackers in previous works
  - Model structures, weights ($\rightarrow$ gradients), runtime setup...
  - Problem: Weights are often confidential

A More Restricted Attacker

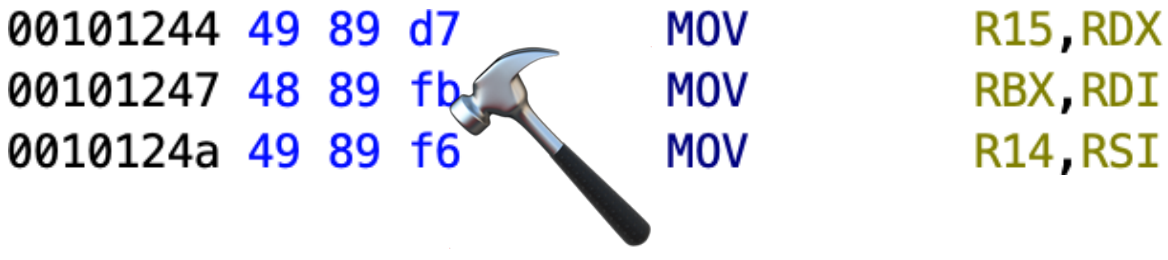
- Recall: White-box attackers in previous works
  - Model structures, weights ($\rightarrow$ gradients), runtime setup...
  - Problem: Weights are often confidential

- Our attacker: ~~Weights~~ $\Rightarrow$ No gradient-based search
  - How to identify which bits to flip?

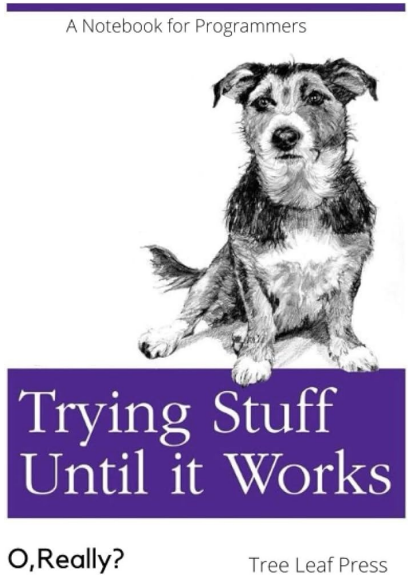
DNN “Executables”

DNN “Executables”

- DNN executables are compiled code
  - We can flip bits in the code (.text section)
  - (Which are compiled DNN operators)



The Random Baseline



Knowledge: Offset of victim's .text

↓

Bits to flip: Randomly choose in range

The Random Baseline (It was Bad)

The Random Baseline (It was Bad)

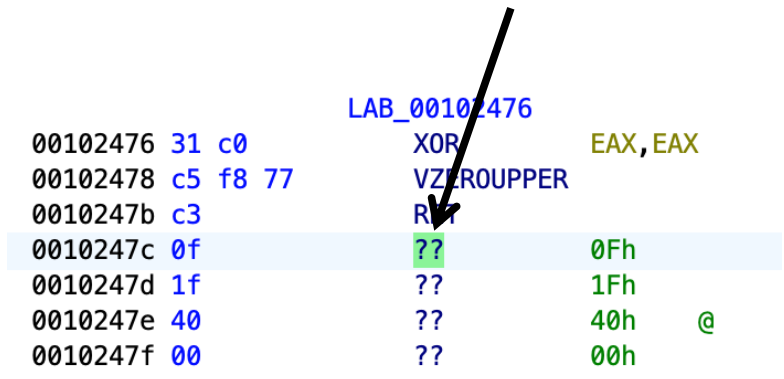
- Attack success rate (ASR): ~2%

The Random Baseline (It was Bad)

- Attack success rate (ASR): ~2%
- What happened in the remaining 98% of time?
  - Crash / no effect

```
segfault at 940c9 ip 00007f329a3df57b sp 00007f3299b54d10 error 6
segfault at 73249 ip 00007f329a3df57b sp 00007f3298b52d10 error 6
segfault at 20e09 ip 00007f329a3df57b sp 00007f329634dd10 error 6
segfault at 523c9 ip 00007f329a3df57b sp 00007f3297b50d10 error 6
segfault at ffffffffffffff89 ip 00007f329a3df57b sp 00007f32290e9b9
segfault at 7f326a8ecc40 ip 00007f329a3df56f sp 00007f322a8ecb90 er
segfault at 48000028 ip 00007f329a3df577 sp 00007f32290e9b90 error
10909] trap invalid opcode ip:7f329a3df577 sp:7f32290e9b90 error:0 i
segfault at 7f329b34fdc0 ip 00007f329a3df56f sp 00007f329734fd10 er
```

Function already returned



Scan Survey Results

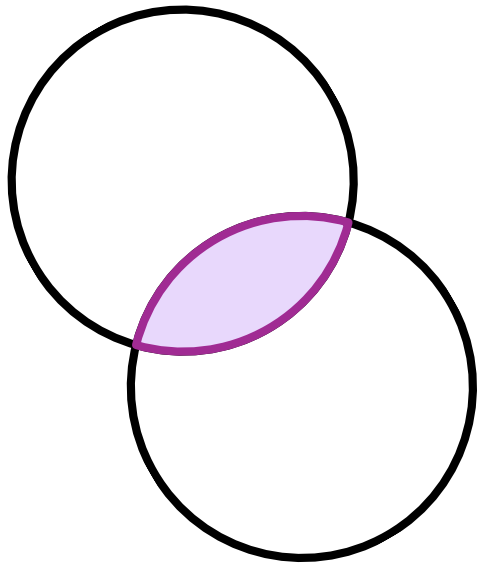
- Pervasive attack surface does exist
  - Different models / datasets / compilers
- How to identify vulnerable bits?
  - Do they change with training data? Yes
  - Do attackers have training data? No

Model	Dataset	Compiler	% Vuln.
ResNet50	CIFAR10	TVM	2.59
ResNet50	MNIST	TVM	3.14
ResNet50	Fashion	TVM	3.07
GoogLeNet	CIFAR10	TVM	2.56
GoogLeNet	MNIST	TVM	2.51
GoogLeNet	Fashion	TVM	2.59
DenseNet121	CIFAR10	TVM	2.66
DenseNet121	MNIST	TVM	2.10
DenseNet121	Fashion	TVM	2.29
ℚResNet50	CIFAR10	TVM	2.17
ℚGoogLeNet	CIFAR10	TVM	0.84
ℚDenseNet121	CIFAR10	TVM	0.52
LeNet	MNIST	TVM	2.52
DCGAN	MNIST	TVM	7.54
ResNet50	CIFAR10	Glow	2.48
ResNet50	MNIST	Glow	2.32

Do vulnerable bits overlap?

Common Vulnerable Bits

- Trying to find recurring vulnerable bits
- Same model structure, trained on two datasets
  - ~45% vulnerable bits shared

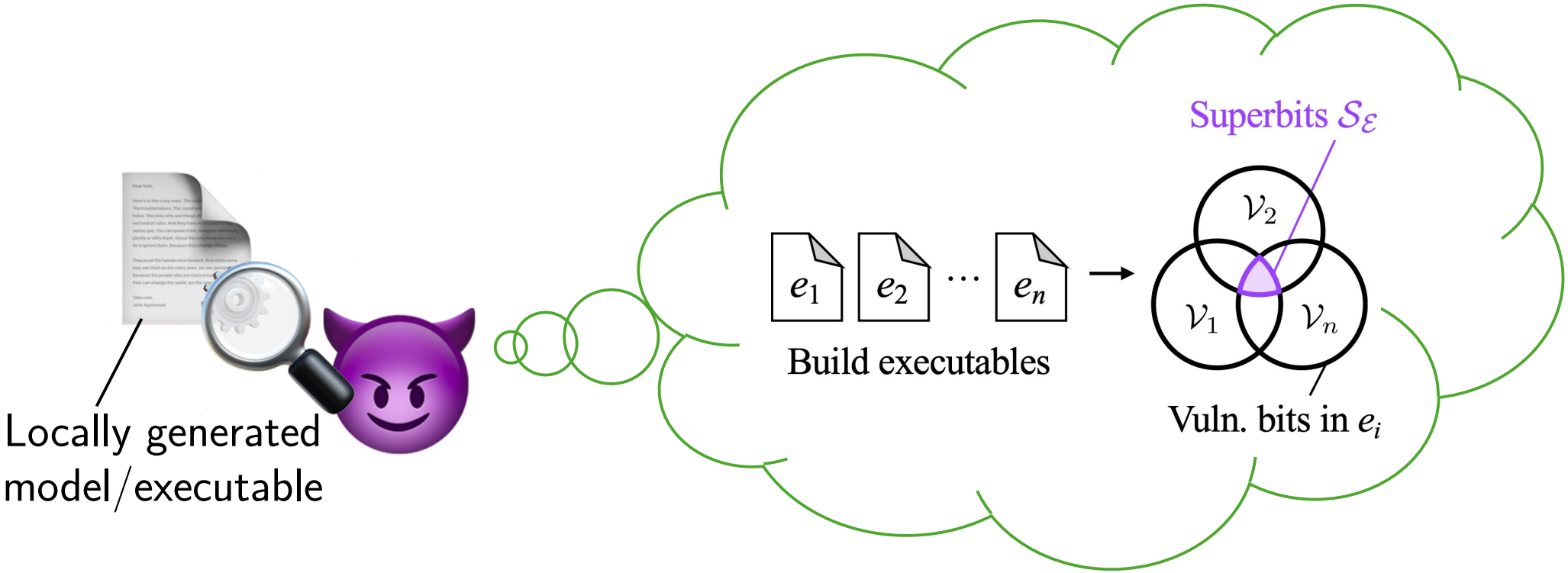


Transferable Vulnerable Bits

45% vulnerable bits transferable, despite different training sets

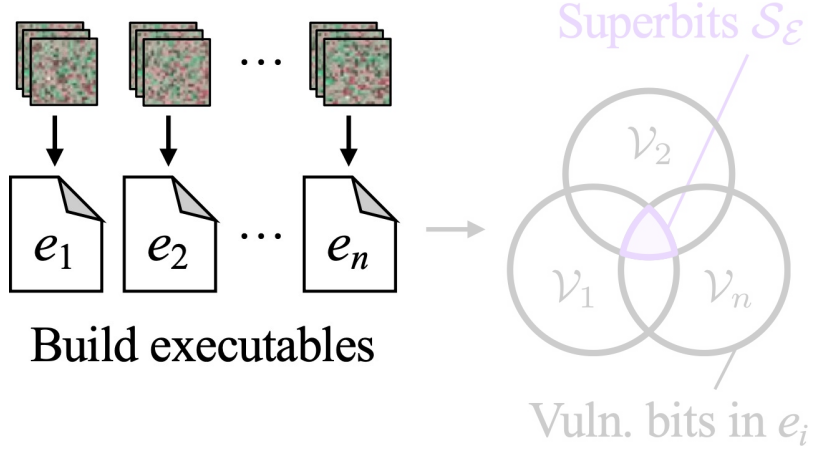
Building an Attack: In Seek of "Superbits"

- Using more local executables for profiling

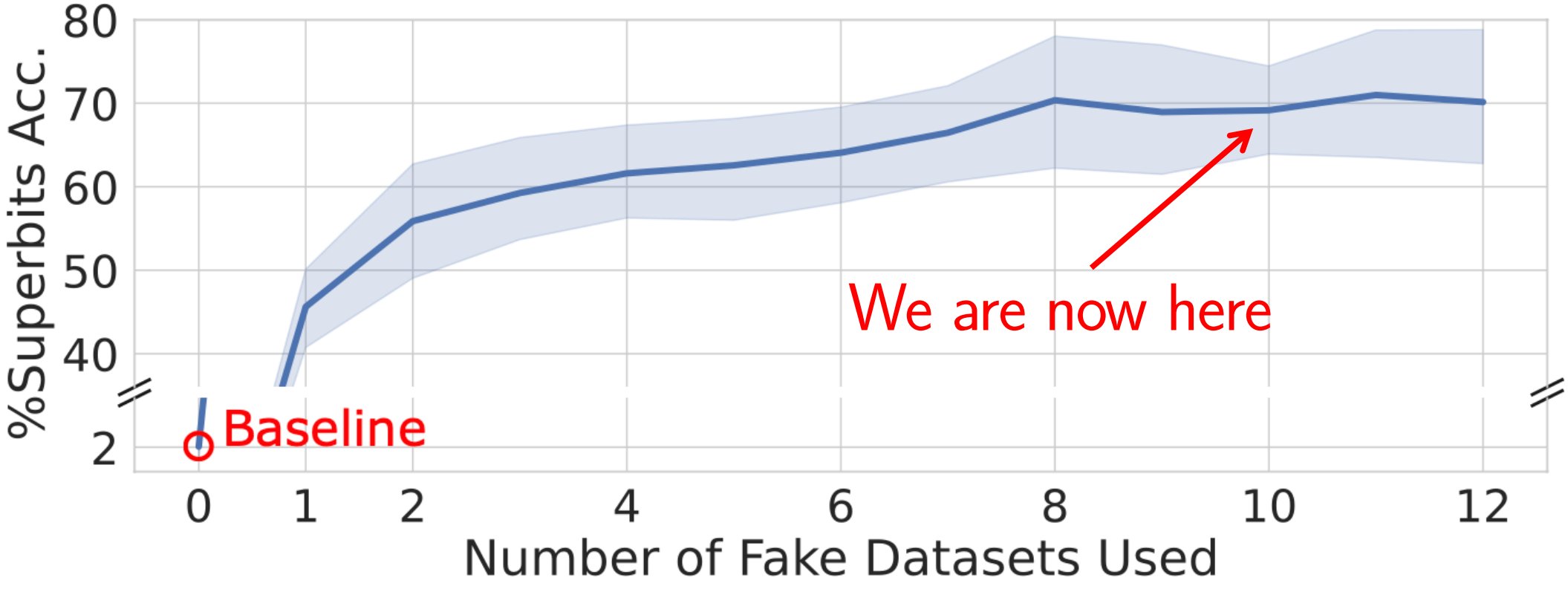


Building More Local Executables

- Train them on datasets of random noise!
  - Regulate weights
  - Unbiased choice



Attack Success Rate: 70%



Real World Examples on DDR4

Model	Dataset	#Flips	#Crashes	%Acc. Change
ResNet50	CIFAR10	1.4	0.0	87.20 → 10.00
GoogLeNet	CIFAR10	1.4	0.0	84.80 → 10.00
DenseNet121	CIFAR10	1.0	0.0	80.00 → 11.40
DenseNet121	MNIST	1.2	0.0	99.10 → 11.20
DenseNet121	Fashion	1.2	0.0	92.50 → 10.60
QResNet50	CIFAR10	1.6	0.0	86.90 → 9.60
QGoogLeNet	CIFAR10	1.4	0.0	84.60 → 11.20
QDenseNet121	CIFAR10	1.6	0.0	78.50 → 10.20
ResNet50	CIFAR10	1.4	0.0	78.80 → 10.00

Real World Examples on DDR4

Model	Dataset	#Flips	#Crashes	%Acc. Change
ResNet50	CIFAR10	1.4	0.0	87.20 → 10.00
GoogLeNet	CIFAR10	1.4	0.0	84.80 → 10.00
DenseNet121	CIFAR10	1.0	0.0	80.00 → 11.40
DenseNet121	MNIST	1.2	0.0	99.10 → 11.20
DenseNet121	Fashion	1.2	0.0	92.50 → 10.60
QResNet50	CIFAR10	1.6	0.0	86.90 → 9.60
QGoogLeNet	CIFAR10	1.4	0.0	84.60 → 11.20
QDenseNet121	CIFAR10	1.6	0.0	78.50 → 10.20
ResNet50	CIFAR10	1.4	0.0	78.80 → 10.00

Avg = ~1.4 flips

What Does That Mean Exactly?

- Compare with DeepHammer, a SOTA method
  - Same attack objectives
  - Applicable to quantised models ("harder to attack")
  - ~12 flips were needed on average vs. ~1.4
  - White-box attacker was required vs. gray-box

Work	Avg. #Flips
BFA [69]	14.3
T-BFA (N-to-1) [71]	23.63
DeepHammer [88]	12.25
Ours	1.4

Conclusion

- DNN executables have large bit-flip attack surfaces
- We achieve 70% confidence in vulnerable bits identification, ~1.4 flips to ruin model intelligence
- More security research on DNN executables please!

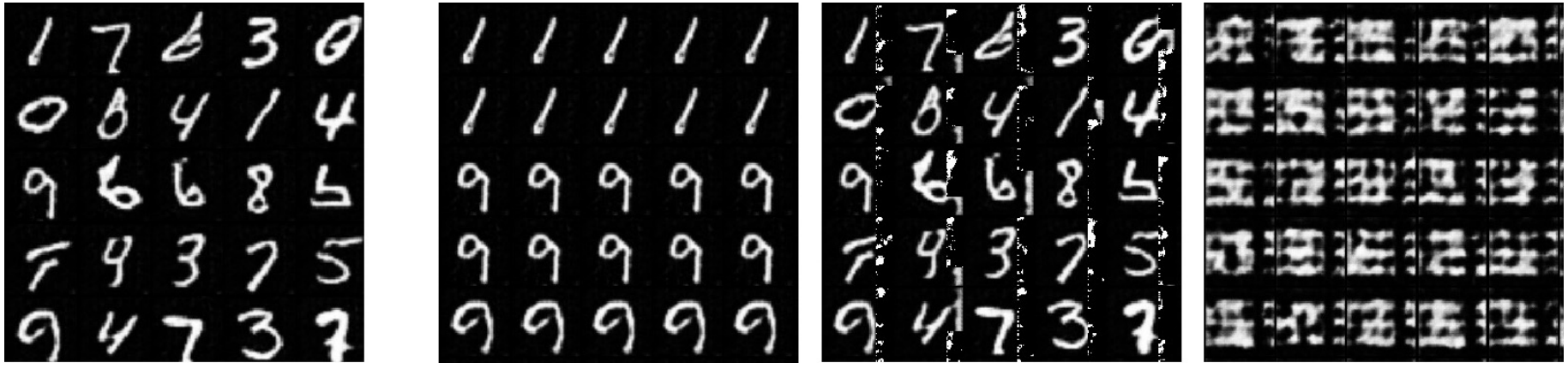
Thank You!

- PDF, source code, other materials
  - Visit yanzuo.ch/debfa

- Contact me
  - Yanzuo Chen: ychenjo@cse.ust.hk
  - Homepage: yanzuo.ch

Compiled Models, Built-In Exploits: Uncovering Pervasive Bit-Flip Attack Surfaces in DNN Executables.
By Yanzuo Chen, Zhibo Liu, Yuanyuan Yuan, Sihang Hu, Tianxiang Li, and Shuai Wang.

Different Effects on GAN



Comparison with Existing Methods

TABLE XI

A COMPARISON OF ATTACK PERFORMANCE WITH PRIOR WORKS. FOR MITIGATIONS, ○, ◐, AND ● DENOTE NO, PARTIAL, AND FULL MITIGATION, RESPECTIVELY.

Work	Attack Target	Avg. #Flips	Mitigable by				
			Q [92]	A [85]	D [13]	W [49]	N [50]
BFA [69]	Weights	14.3	◐	◐	●	●	●
T-BFA (N-to-1) [71]	Weights	23.63	◐	◐	●	●	●
DeepHammer [88]	Weights	12.25	◐	◐	●	●	●
Ours	Structure	1.4	○	○	○	○	○

Case Study Example: Control Flow Broken

Addr	Opcode bytes	x86 assembly instruction
0x70	83 F8 28	cmp eax, 28h ;; max ID
0x73	0F 4D C2	cmovge eax, edx ;; edx=28h
0x76	39 F0	cmp eax, esi ;; esi<28h
0x78	0F 8D FA 00+ 00 00	jge func_end

(a) Assembly code before BFA.

0x70	83 FC 28	cmp esp, 28h ;; true
0x73	0F 4D C2	cmovge eax, edx ;; true
0x76	39 F0	cmp eax, esi ;; true
0x78	0F 8D FA 00+ 00 00	jge func_end ;; exit

(b) Assembly code after BFA.