black hat EUROPE 2024

DECEMBER 11-12, 2024

BRIEFINGS

The Devil is in the (Micro-) Architectures: Uncovering New Side-Channel and Bit-Flip Attack Surfaces in DNN Executables

Speakers:

Yanzuo Chen
PhD at HKUST

Zhibo Liu
Postdoc at HKUST

Contributors:

Yuanyuan Yuan
Postdoc at ETH

Shuai Wang
Associate Professor
at HKUST

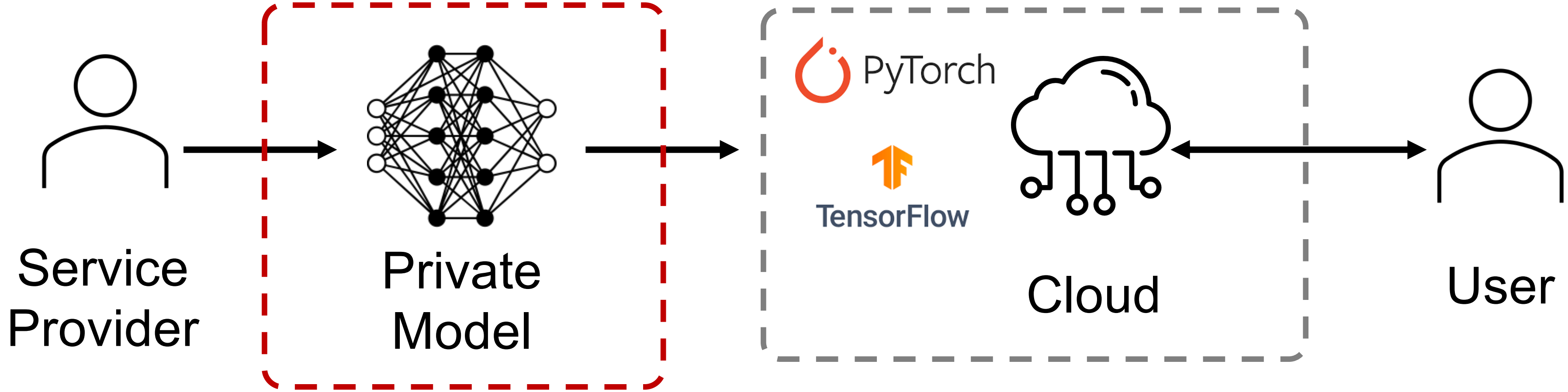
Tianxiang Li   Sihang Hu   Zhihui Lin
Security Researchers at CSI AI Red Team

The Age of AI

- Machine Learning as a Service (MLaaS)

MLaaS

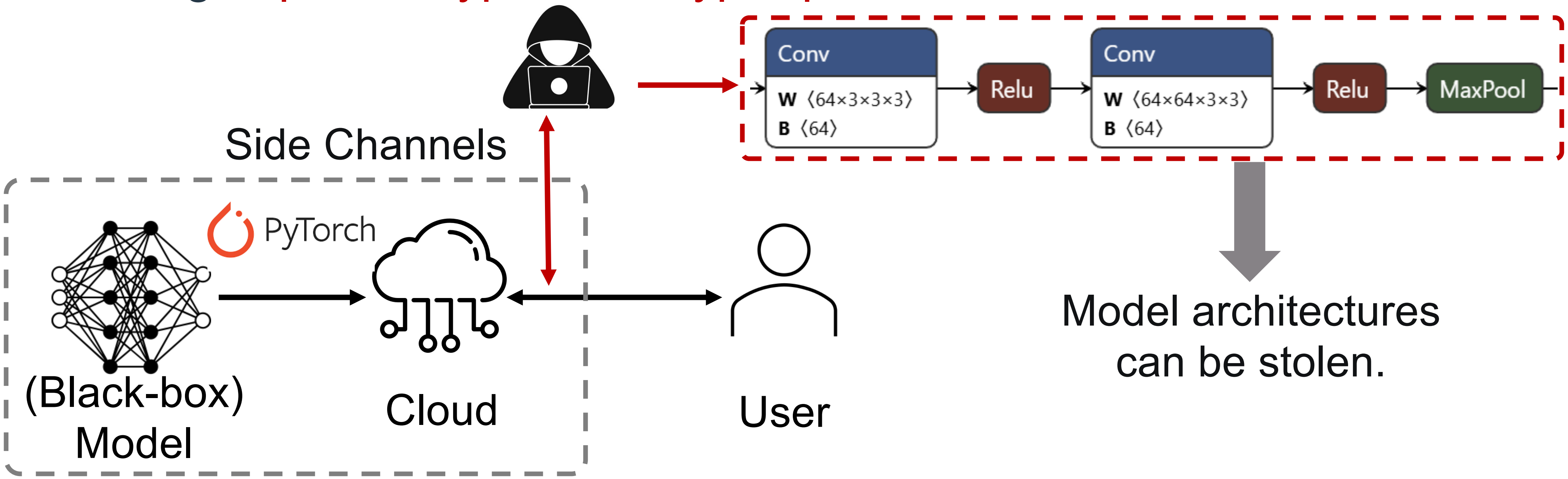
- Run ML models in could



Valuable Property
e.g., design, parameters …

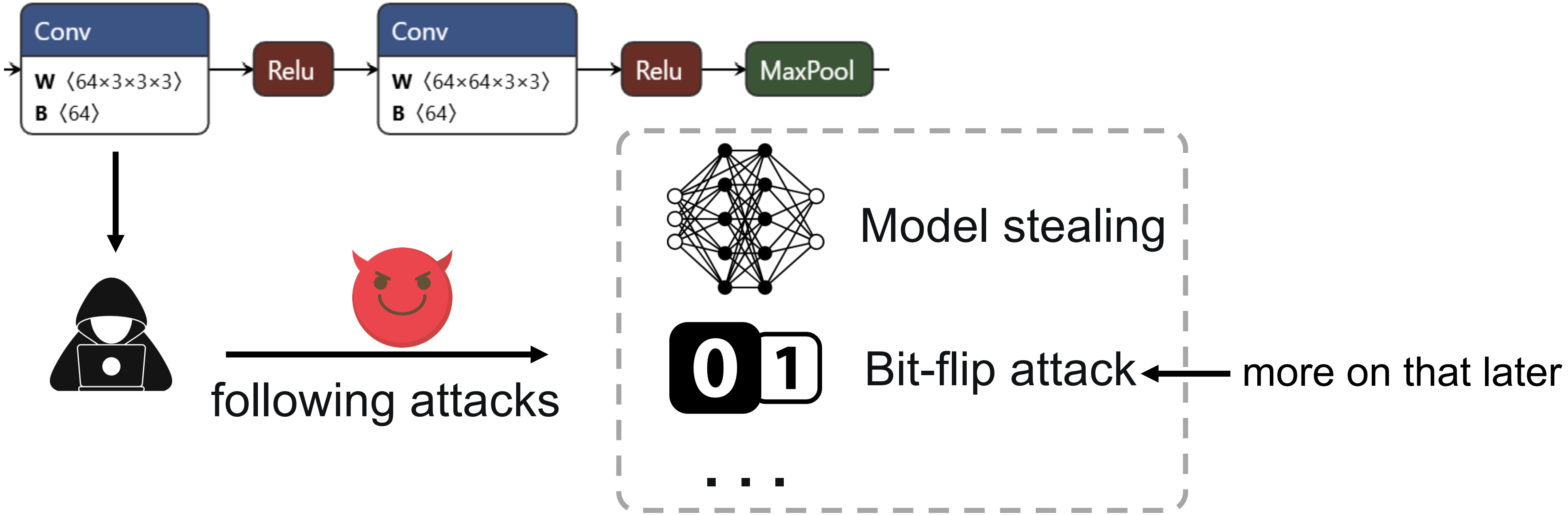
Attacks Arising

- Attacking objectives: **model architectures**
  - e.g., operator types and hyper-parameters



Attacks Arising

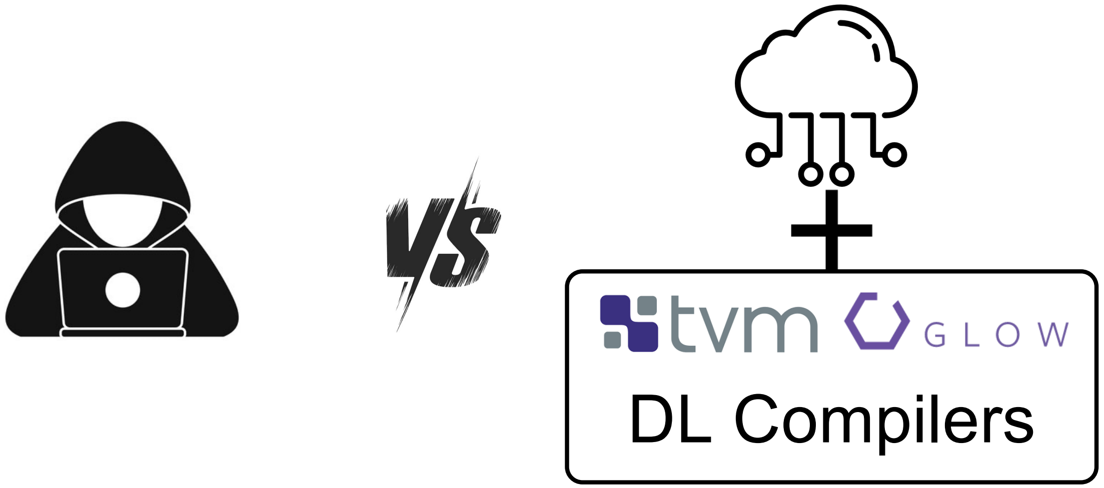
- Model architectures can enable various gray-box attacks
  - e.g., model stealing and bit-flip attack



Meanwhile

- Cloud service providers (e.g., Meta, AWS, and Google) are employing DNN compilation in resource-sharing environments for cost and profit reasons

Are DNN executables vulnerable to side-channel attacks?

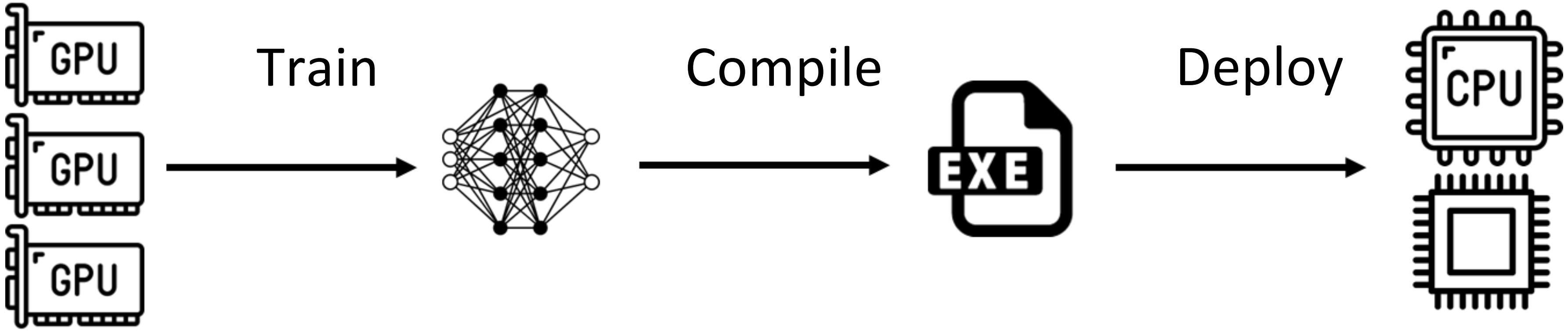


Outline

- Background
  - Deep Learning (DL) Compilation
  - DNN Executable
- How to Steal Model Architectures
  - Cache Side-Channel
- Making Models Do Bad Stuff
  - Bit-Flip Attack

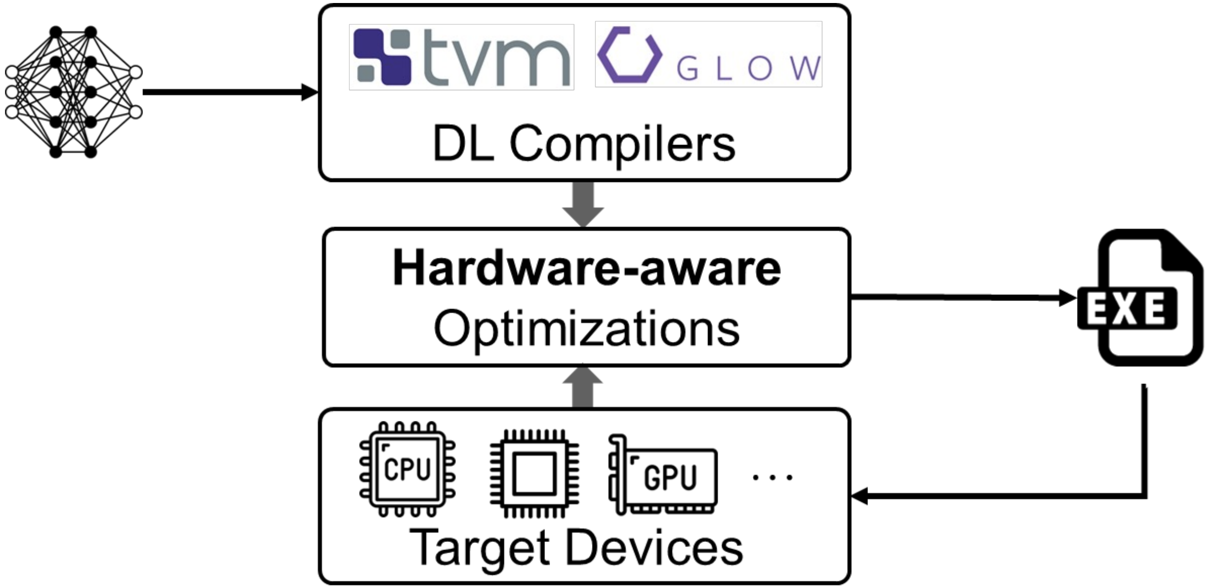
DNN Executable

- GPUs are expensive
  - Running DNNs on cost-efficient devices is popular
- DL compilation techniques are proposed to speed up DNN inference



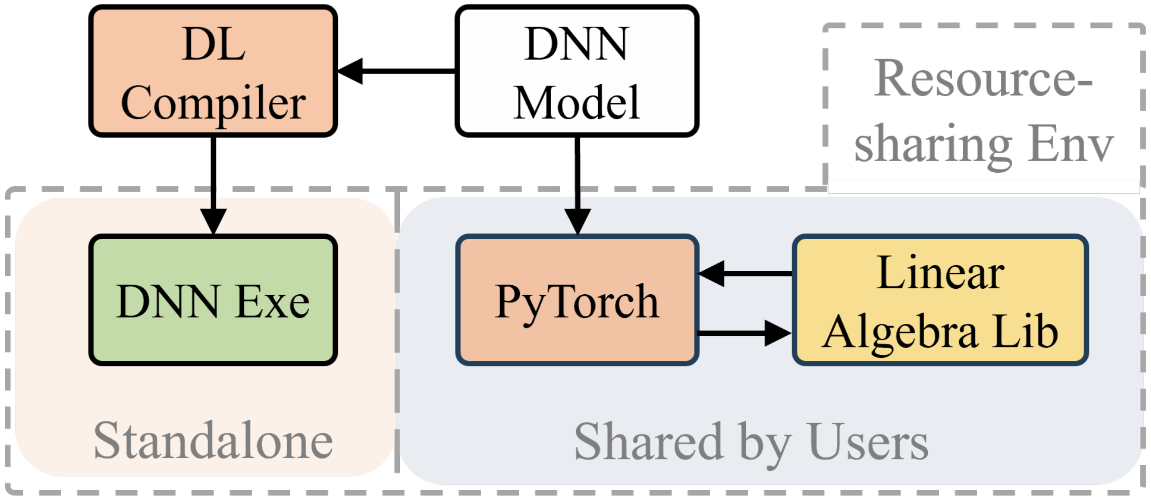
DL Compiler

- Automatically optimize the DNN and generate efficient binary code
- Unlock the full performance potential of various hardware



DNN Executable

- What are the differences compared with DL frameworks (e.g., PyTorch)
  - Each operator is optimized explicitly
  - Standalone
  - No libs during execution



Side-Channel Attacks

- Side-channel attacks on DNNs are emerging

Physical Access		Remote Access	
Electromagnetic	[Sec’19] [ASPLOS’20]	Rowhammer	[SP’22]
Bus Snooping	[Sec’21] [ASPLOS’23]	Power	[SP’24]
Power	[HOST’20]	Cache	[Sec’20]
	…		…

More discussion: yanzuo.ch/bh24

[CCS’24] DeepCache: Revisiting Cache Side-Channel Attacks in Deep Neural Networks Executables

Side-Channel Attacks

- We focus on remote *model architecture stealing* attacks

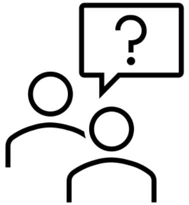
	Limitation	
Rowhammer	Leak partial information from quantized DNN	NO
Power	Rely on RAPL interface (require privileges)	NO
Cache	Need shared cache (and memory regions)	🤔

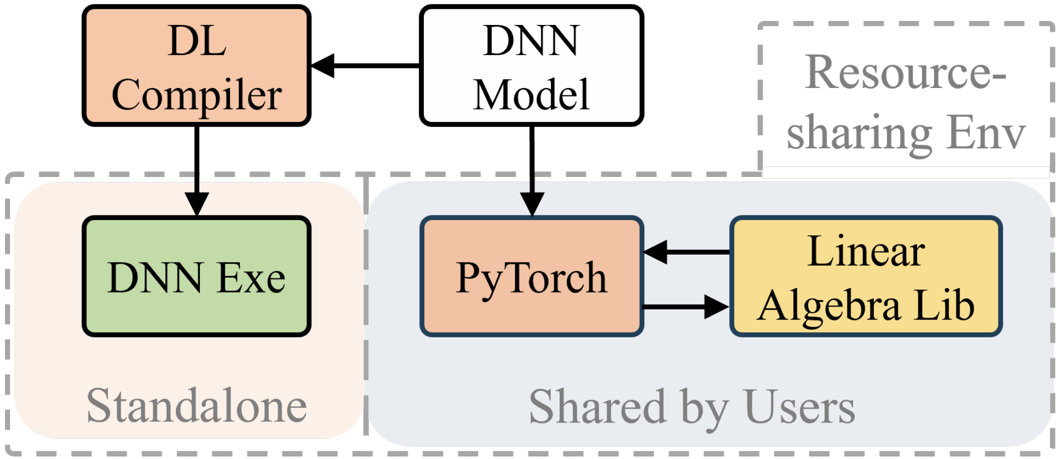
More discussion: yanzuo.ch/bh24

[CCS'24] DeepCache: Revisiting Cache Side-Channel Attacks in Deep Neural Networks Executables

Challenges

- None of existing cache side channel attacks apply to DNN executable
- Why?
  - Standalone
  - No shared memory
  - No libs for pre-analysis





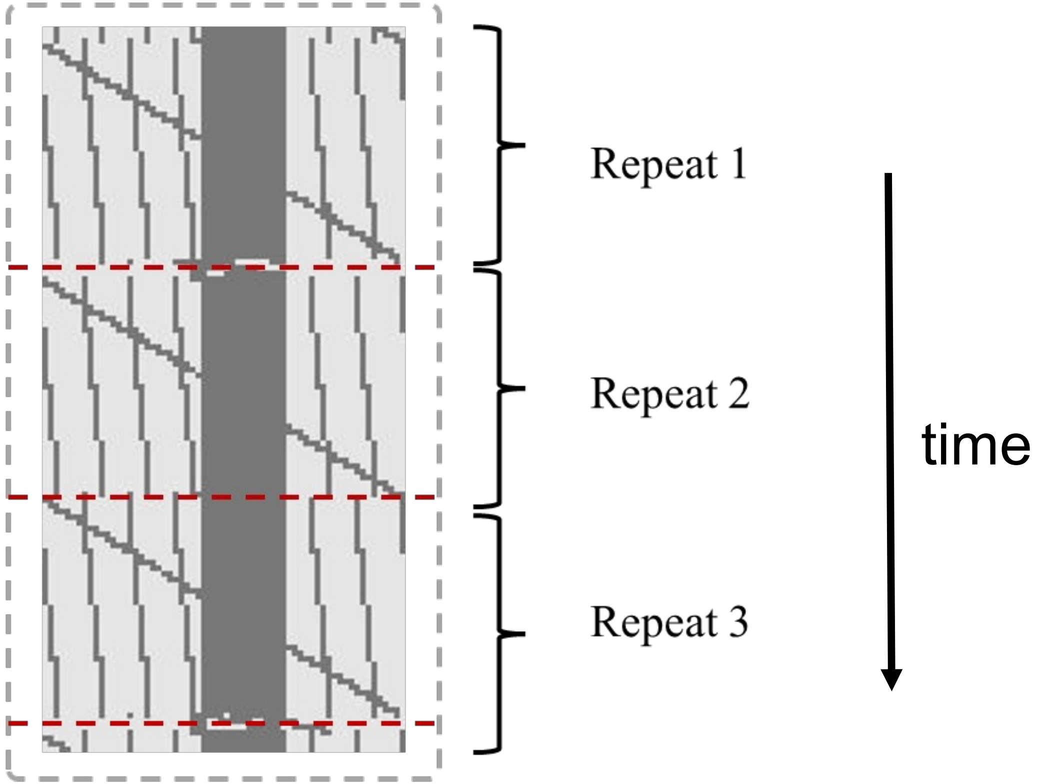
- Is DNN executable more secure?

Zoom In

- Noise free
- Simulated with Intel Pin
- Mimic Prime+Probe

Each **row** represents a cache state (e.g., 64 cache lines).

dark pixels → cache hits
light pixels → cache misses

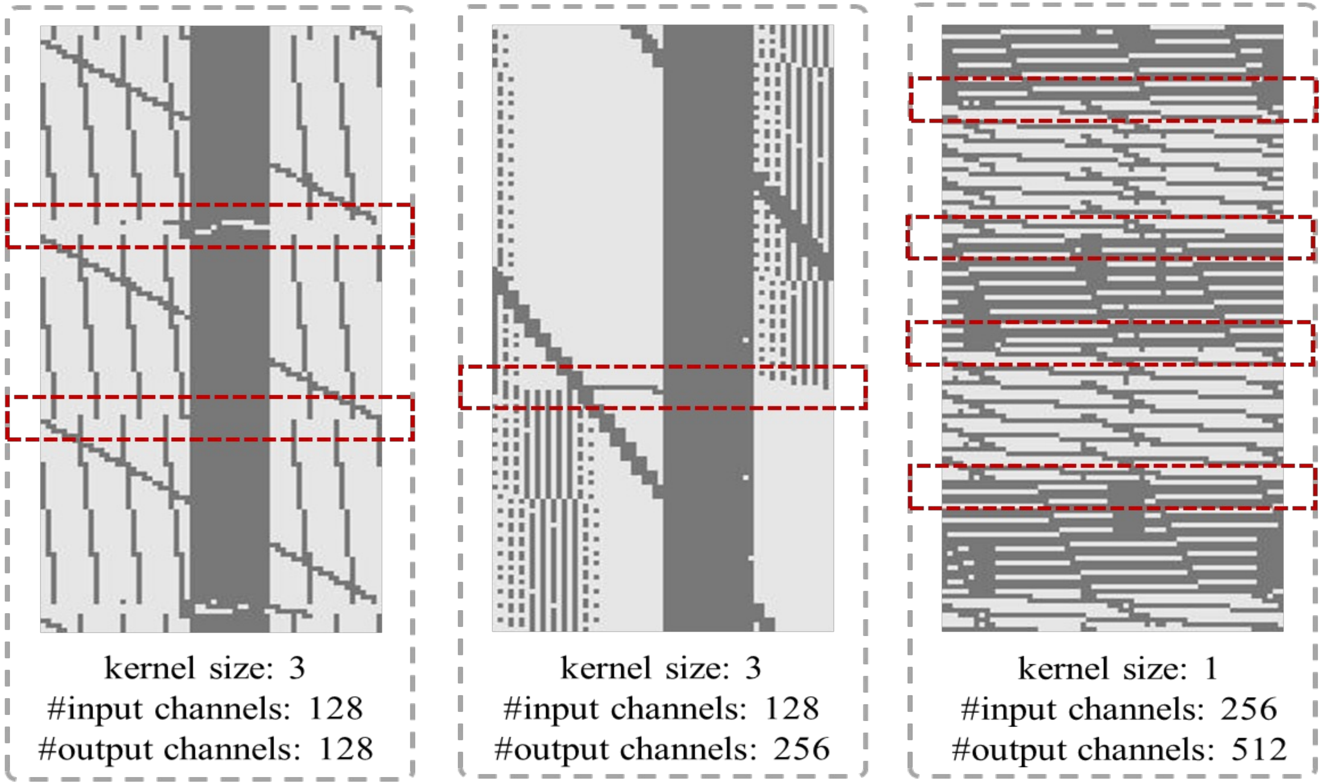


(a) Trace of a Conv compiled by **TVM**

Cache Access Patterns

Why is that?

Compiler Optimizations!



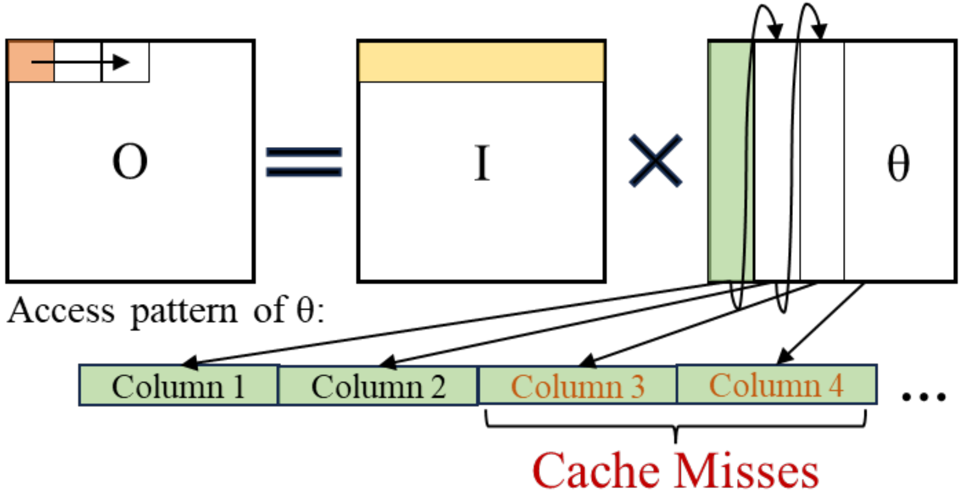
(a) Conv from ResNet18 compiled by TVM.

(b) Conv from VGG16 compiled by TVM.

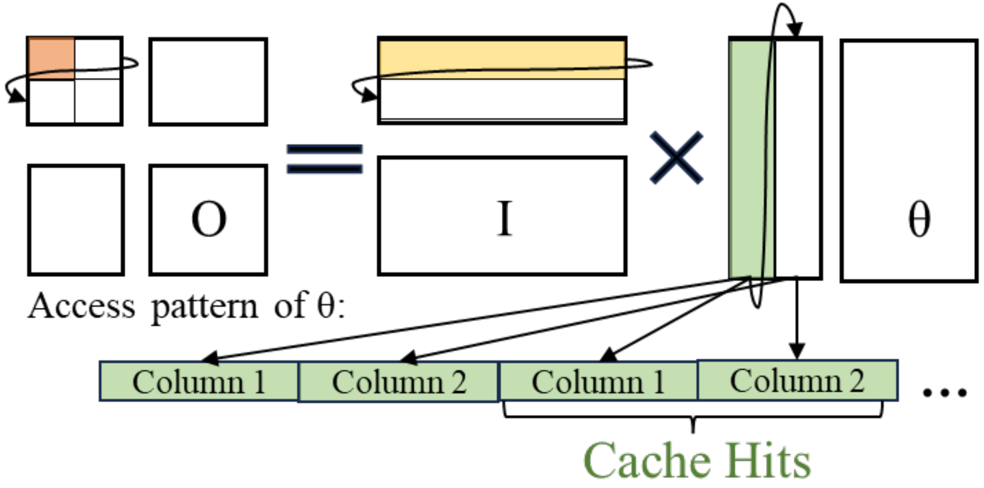
(c) Conv from ResNet18 compiled by Glow.

DL Compiler Optimizations

- Blocking
- For better memory/cache locality



(a) Matrix multiplication without blocking.

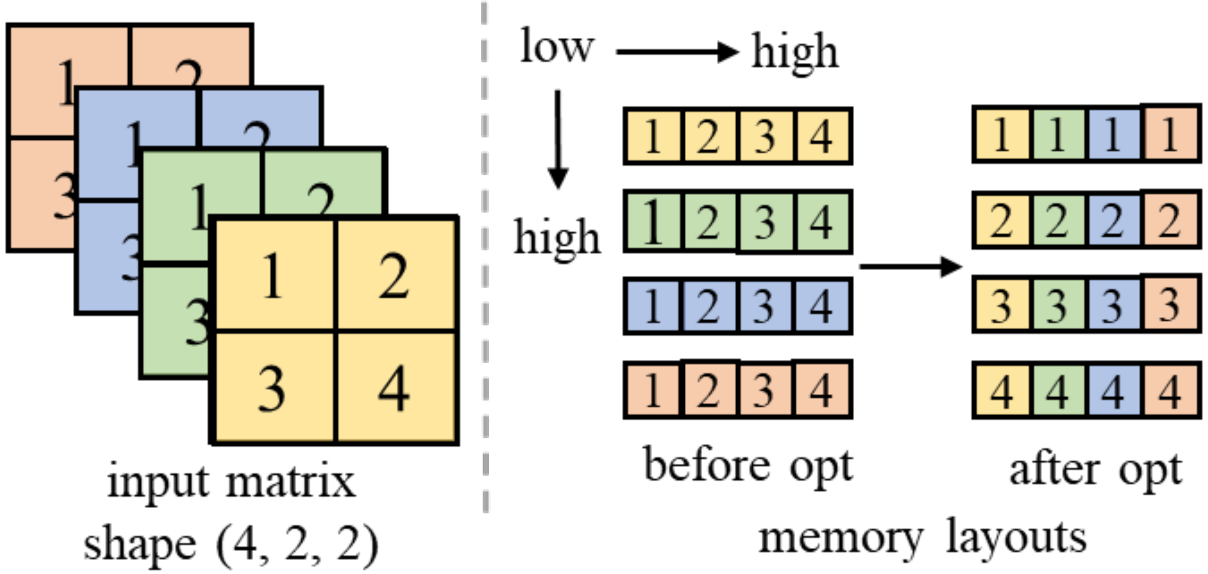


(b) Matrix multiplication with blocking.

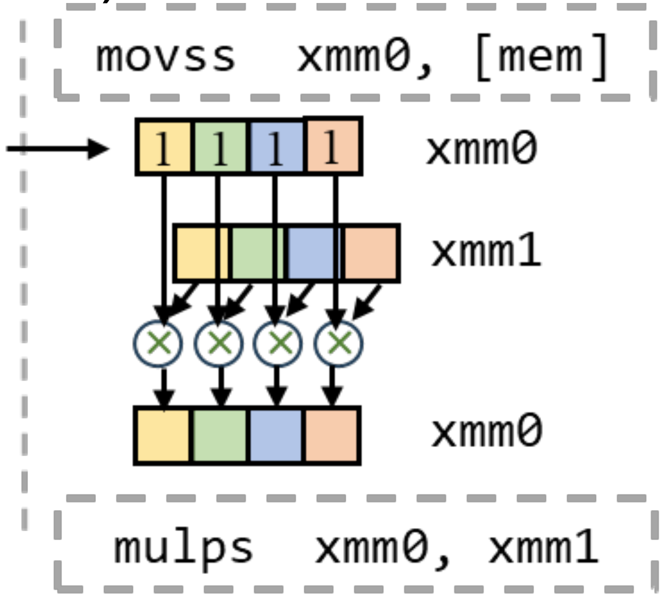
The size of cache is limited (e.g., 32KB)

DL Compiler Optimizations

- Vectorization
- Leverage **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD) extension



Memory layout optimization.



SIMD instruction example.

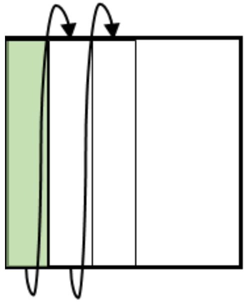
DL Compiler Optimizations

- Pseudo code illustration

➢ Convolution

➢ Naïve loop structures

➢ Sweep the whole matrix



```
1  def Conv(I, W, O):
2    # output channels
3    for oc in range(256):
4      # output height
5      for oh in range(14):
6        # output width
7        for ow in range(14):
8          # lines 2-7: each output element
9          # input channels
10         for ic in range(128):
11           # kernel height
12           for kh in range(3):
13             # kernel width
14             for kw in range(3):
15               v_1 = oh * stride + kh
16               v_2 = ow * stride + kw
17               O[1][oc][oh][ow] += \
18                 I[1][ic][v_1][v_2] * \
19                 W[oc][ic][kh][kw]
```

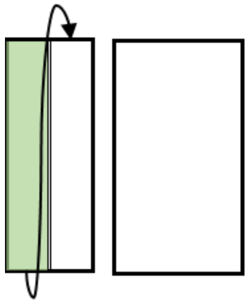
low memory locality

DL Compiler Optimizations

- Pseudo code illustration

Optimized loop structures

Loops are split and permutated



```
for oc_outer_fused_oh in range(8*14):
  for ow_outer in range(2):
    for ic_outer in range(16):
      for kh in range(3):
        for kw in range(3):
          for ic_inner in range(8):
            for ow_inner in range(7):
              for oc_inner in range(32):
                ow = ow_inner + ow_outer * 7
                oh = oc_outer_fused_oh % 8
                oc_outer = oc_outer_fused_oh / 14
                iw = ow * stride + kw
                ih = oh * stride + kh
                O[1][oc_outer][oh][ow][oc_inner] += \
                I[1][ic_outer][ih][iw][ic_inner] * \
                W[oc_outer][ic_outer][kh][kw][ic_inner][oc_inner]
```

higher memory locality

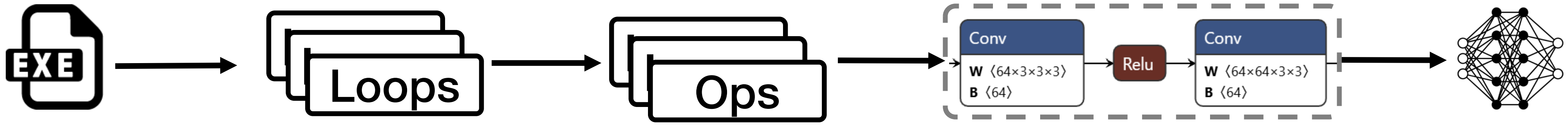
Split & permutated, original:

```
for ic in range(128)
// 128 = 16 * 8
```

Unique Loop Structures

- Compiler optimizations depend on the hyper-parameters of operators.
  - Different operator types and hyper-parameters →
  - Distinct loop structures in compiled low-level code.

- If we can determine the loop structure, we can distinguish operators.

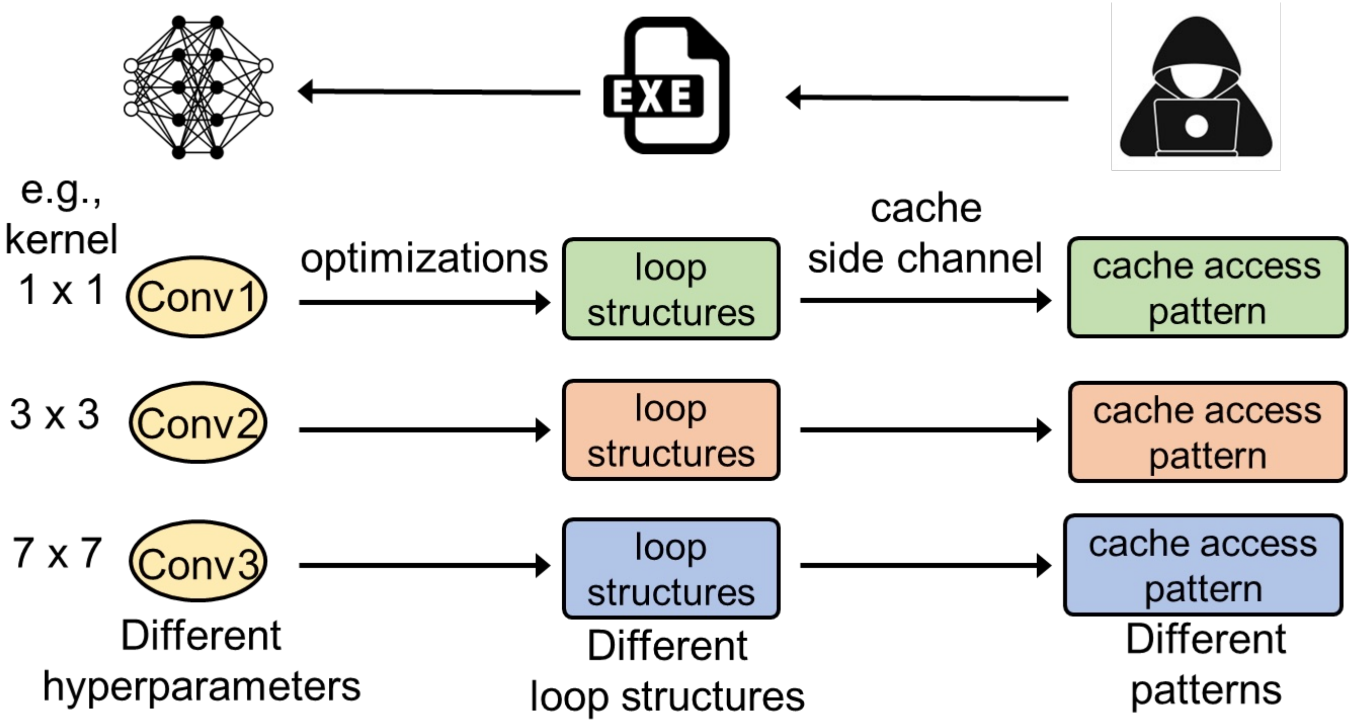


Unique Loop Structures

- DNN inference involves massive memory accesses, resulting distinguishable cache activities

- We depict binary-level code structures with $Loop_I$ (inner loop) and $Loop_O$ (outer loop)
  - $Loop_I$ denotes the repeated pattern
  - $Loop_O$ represents the frequency of a pattern's occurrence

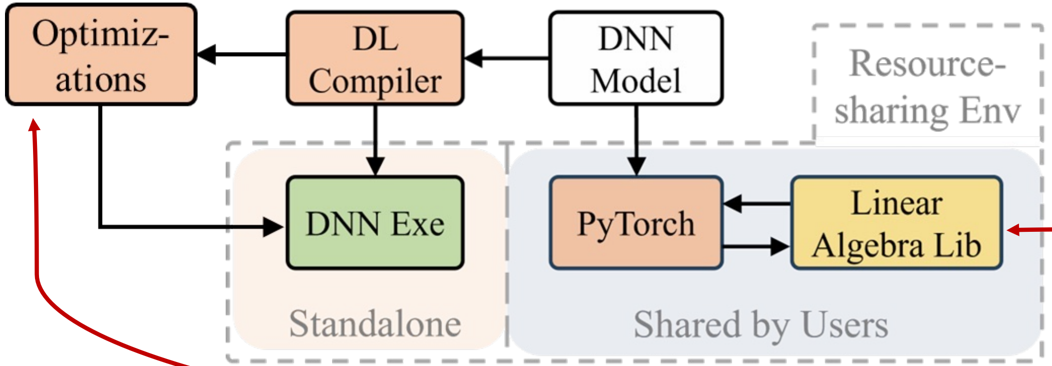
Unique Loop Structures



- ➢ There should be a one-to-one mapping relation that attacker can exploit to infer operators.

New Attacking Surface

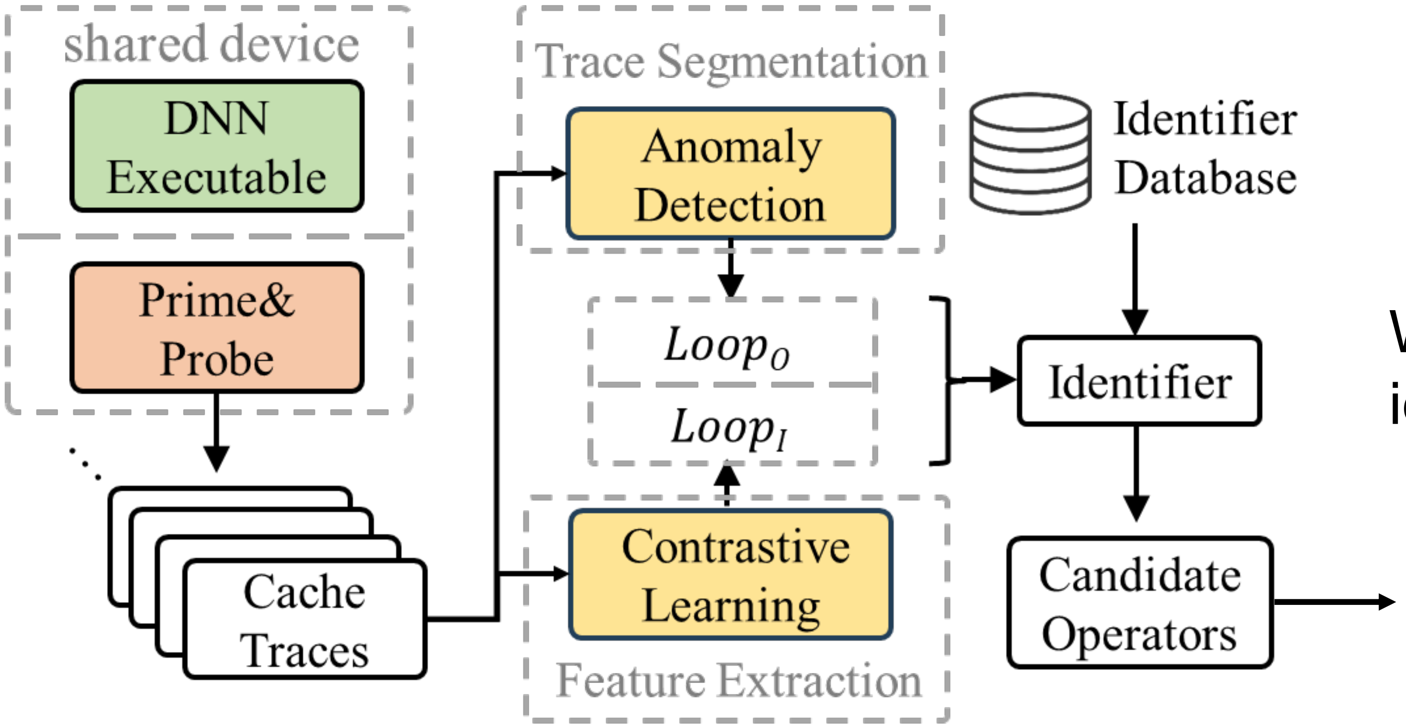
- Prior works manually locate sensitive functions in linear algebra libraries as target of cache side channels.



- Differently, we reveal that hardware- and cache-aware optimizations introduce new cache side channel leakages.

DeepCache: End-to-End DNN Architecture Stealing

- We approximate a mapping from cache access traces to loop structures

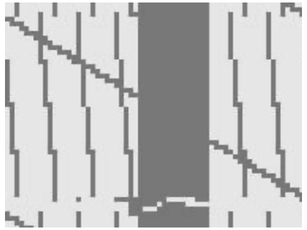


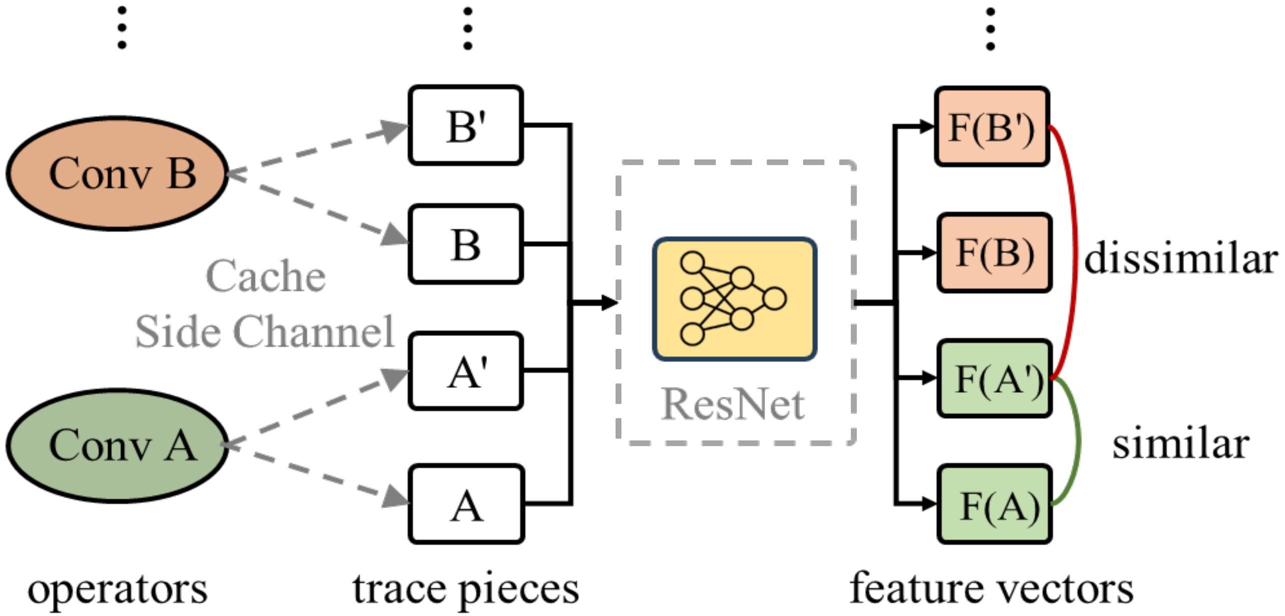
We match similar record in the identifier database.

Operator **type**, **hyperparameters**, optimized **memory layouts**…

Contrastive Learning

- Extract features cache access traces

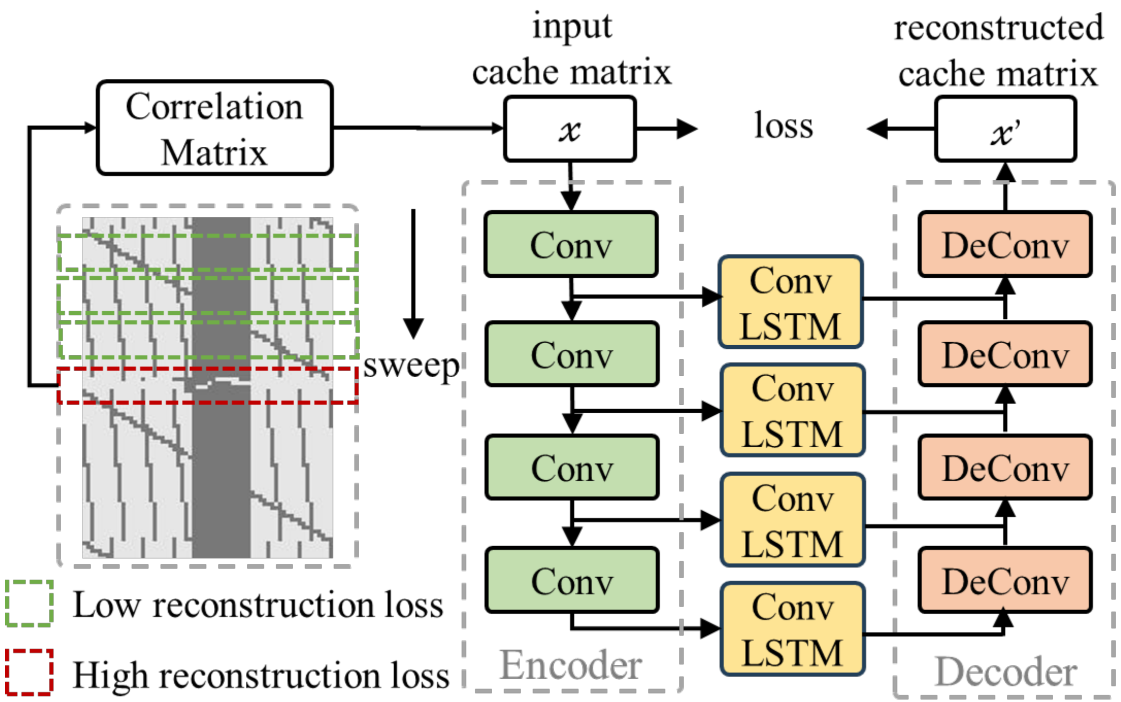
E.g., A' = 



Traces from the same operator should have similar features.
Extracted features are deemed as $Loop_l$

Trace Segmentation

- We use encoder-decoder network to segment traces



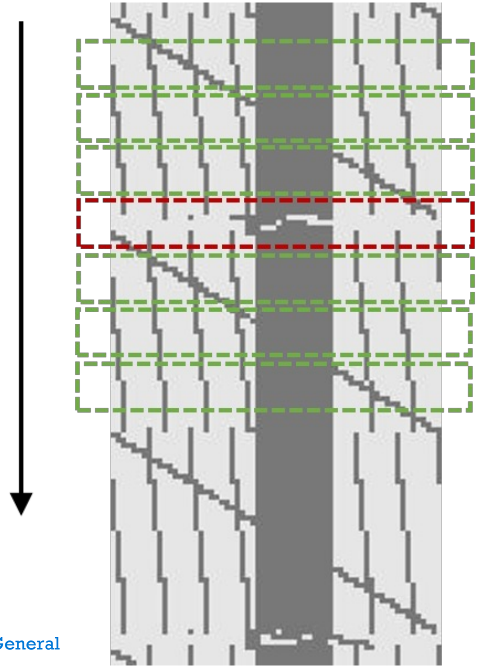
Compare recovered and original cache trace pieces

Similar:
  smooth normal patterns
Dissimilar:
  anomaly! → segment

Idea: frequent normal patterns can quickly be learned.

Trace Segmentation

Encoder: compress the information (of learned patterns)
Decoder: recover the original information (uncompress)



Success to recover → the pattern is seen before

Fail to recover → the pattern is an anomaly → segmentation point

Sweep the trace to figure out how many times the whole pattern repeated.

Evaluation

- We collect 28 real-world CNN models (372 operators) from ONNX Zoo as database
- All models are compiled with two state-of-the-art DL compilers, TVM and Glow
- ResNet18 and VGG16 as the test set
- Evaluated with L1 and LLC `Prime+Probe` attack

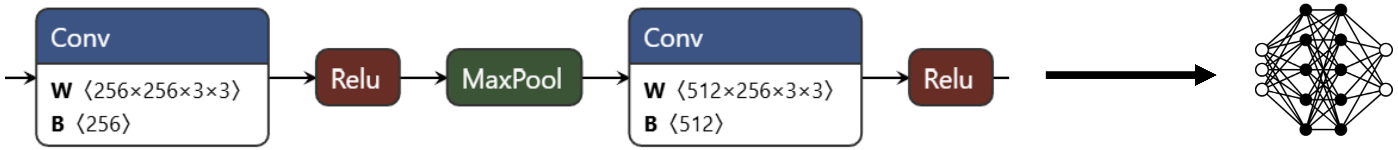
Results

- Victim

- Results

Op1 → {type: Conv, shape: [256, 256, 3, 3]}
Op2 → {type: ReLU}
Op3 → {type: MaxPool}
Op4 → {type: Conv, shape: [512, 256, 3, 3]} . . .

- Recovered



Results

- L1

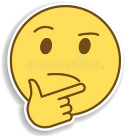
Table 4: The performance of DEEPCACHE with L1 Prime+Probe attack in recovering DNN architectures, and memory layouts.

	TVM		Glow	
	ResNet	VGG	ResNet	VGG
Operator Types	95.2%	88.2%	94.4%	81.3%
Hyperparameters	96.2%	89.5%	71.9%	87.5%
Mem Layouts	100%	100%	71.0%	100%

- LLC

Table 5: The performance of DEEPCACHE with LLC attack.

	TVM		Glow	
	ResNet	VGG	ResNet	VGG
Operator Types	95.2%	100%	100%	100%
Hyperparameters	92.6%	100%	100%	100%
Mem Layouts	91.9%	100%	100%	100%



Why is LLC attack much better?

Results

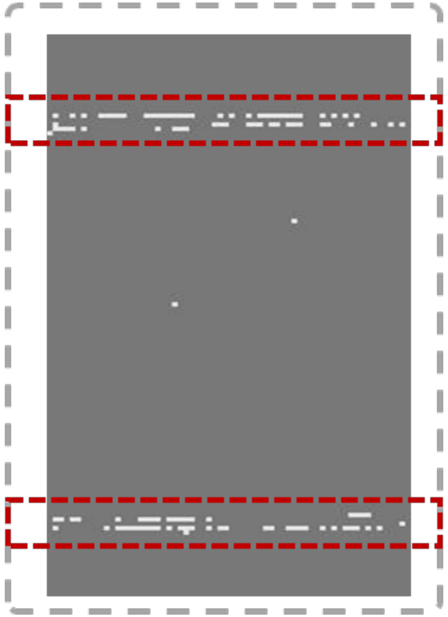
- Why does LLC attack show better accuracy than L1 attack?

- Because some operators are compiled into non-optimal binary code
  - i.e., the binary code shows low memory locality
  - consequently, low cache hit rate

- From attack's view, *non-optimal code is difficult to distinguish*

Results

- The cache trace of non-optimal code is featureless



```
for ic in range(512):
    addr1 = (W + w_idx + ic*8)
    addr2 = (W + w_idx + ic*8 + 20h)
    # read a ymmword from addr1
    # read a ymmword from addr2
    addr1 = (W + w_idx + ic*8 + 24000h)
    addr2 = (W + w_idx + ic*8 + 24020h)
    # read a ymmword from addr1
    # read a ymmword from addr2
    # floating-point multiplications
    # ...
```

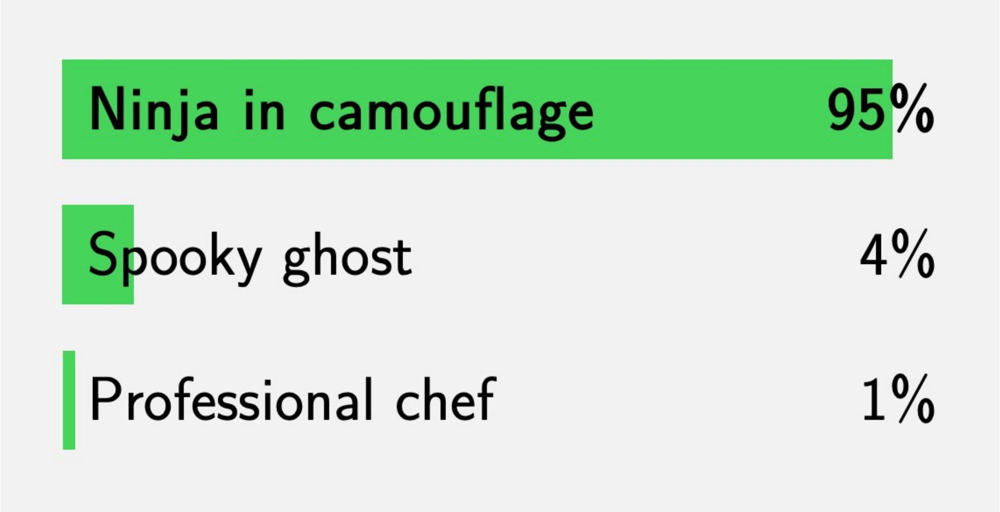
Read 64 KB mem
But L1 cache is 32 KB

Self-competing

(a) Example of featureless trace. (b) Example of non-optimal code.

Part II: Making Models Do Bad Stuff

Speaker: Yanzuo Chen


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

Crime Detector

Yes, putting pineapple on pizza is a crime. It's a violation of the sacred bond between dough, sauce, and cheese. While some may argue that the combination of sweet and savory flavors is delicious, true pizza aficionados know it's an offense to tradition.

Attacks on DNNs

- Existing: adversarial examples, data poisoning, backdoors, …
  - More pointers: yanzuo.ch/bh24

- Optimisation problem vs. Attacking through a new dimension


HIS LAPTOP'S ENCRYPTED.
DRUG HIM AND HIT HIM WITH
THIS $5 WRENCH UNTIL
HE TELLS US THE PASSWORD.
GOT IT.

xkcd.com/538

Is there a way?

Attacking DRAM Microarchitectures

- Rowhammer (🎉 Happy 10th Anniversary)
  - Software-triggered hardware bug
  - Current leakage between DRAM cells
  - Flips data bits in memory

Rowhammer in action

- ✅ DDR3
- ✅ DDR4
- ✅ ECC memory
- ✅ (New!) DDR5

- ✅ Privilege escalation
- ✅ Cross-VM attacks
- ✅ Attacking through browsers

Bit-Flip Attacks (BFAs) on DNNs

- Yes, it works
- Targets victim model weights…
  - *What if we don’t have that knowledge?*

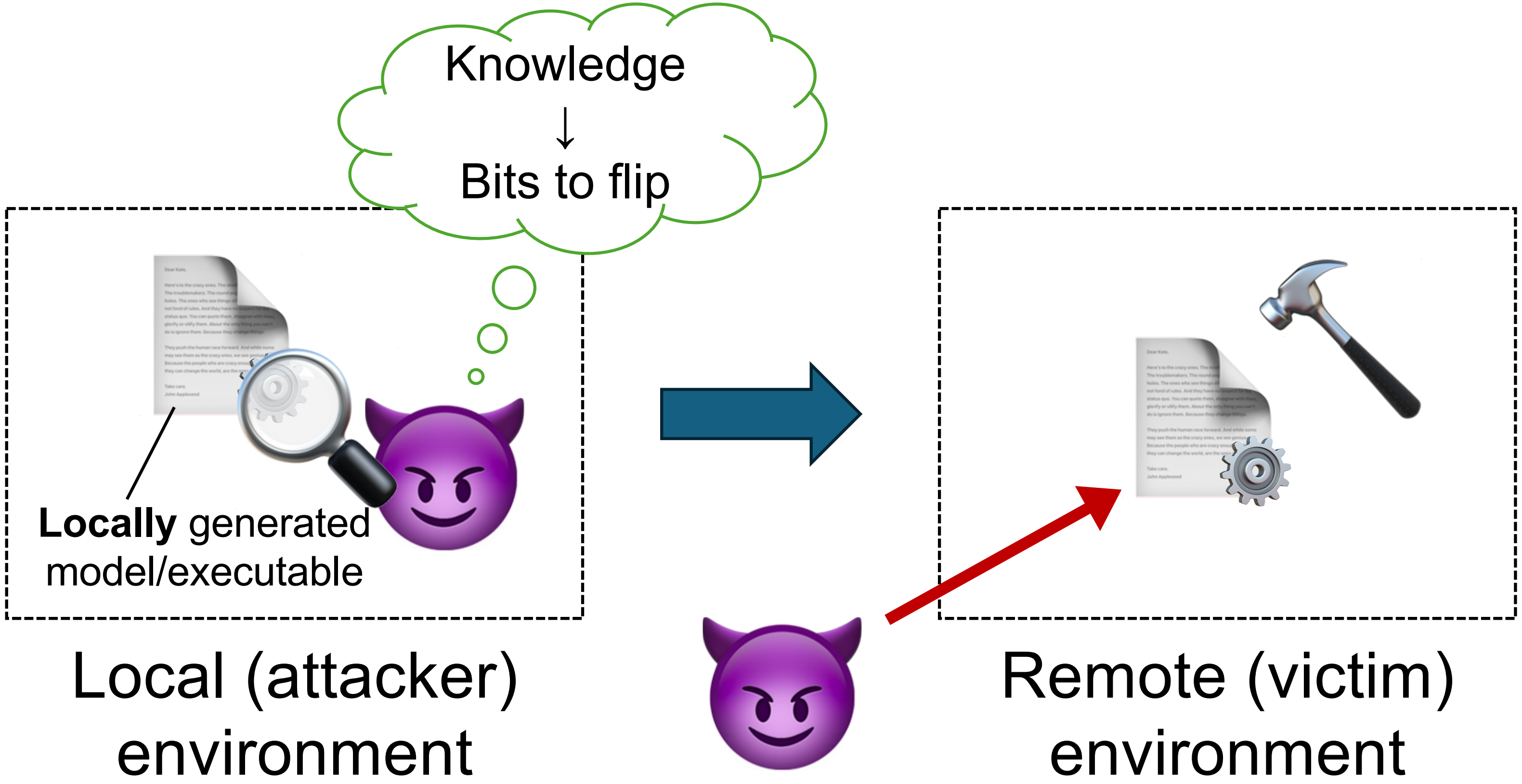
DNN "Executables"

DNN executables are compiled code

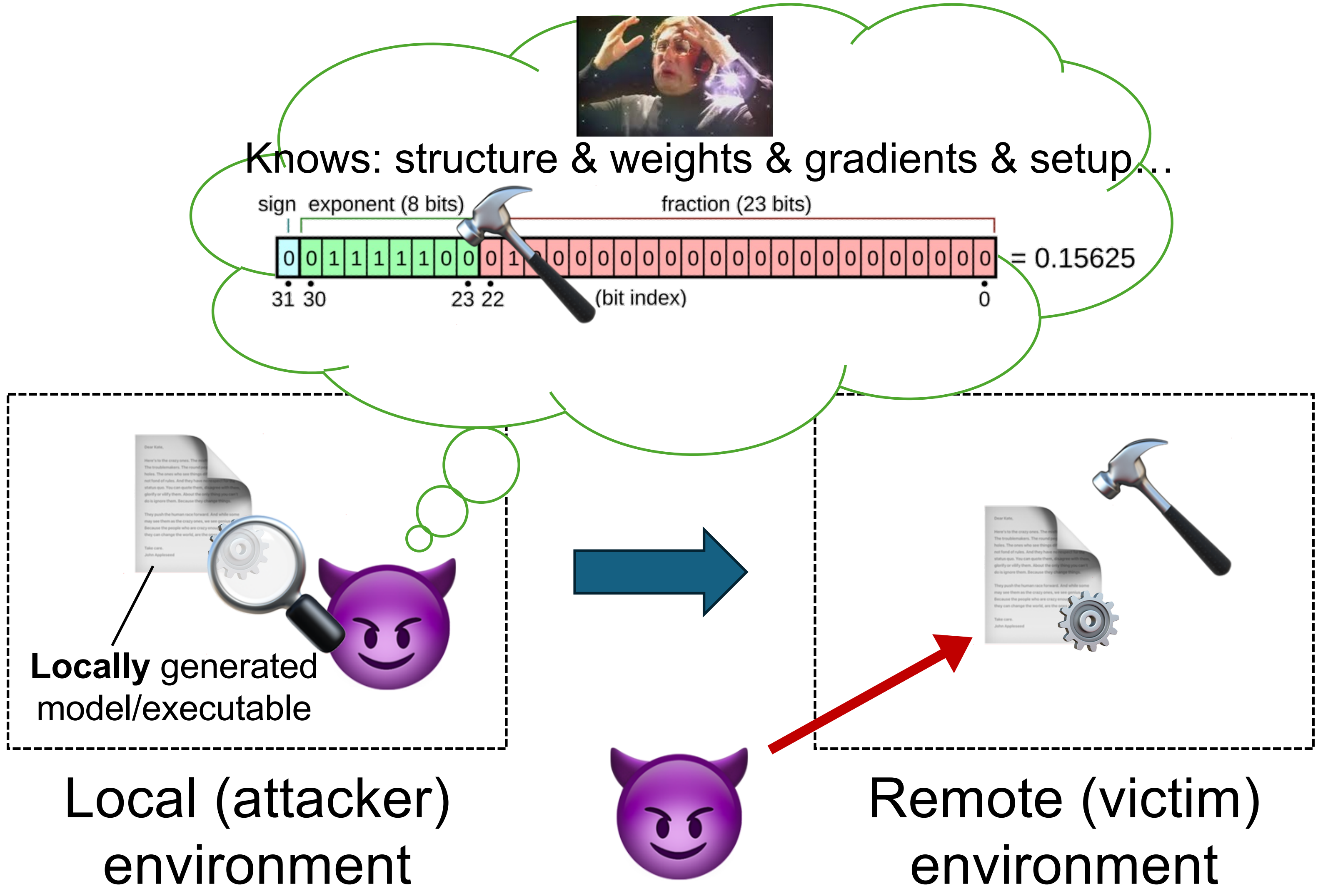
The Setup

- **Attacker objective**: deplete model intelligence via BFAs (E.g., make them random guessers)
- **Attacker knowledge**: Model structure => model executable
  - E.g., with DeepCache (Our Part I) / BTD (Zhibo@BH-USA24)
- Attacker has **no** access to victim model weights
- We figure out: **How** to find bits to flip

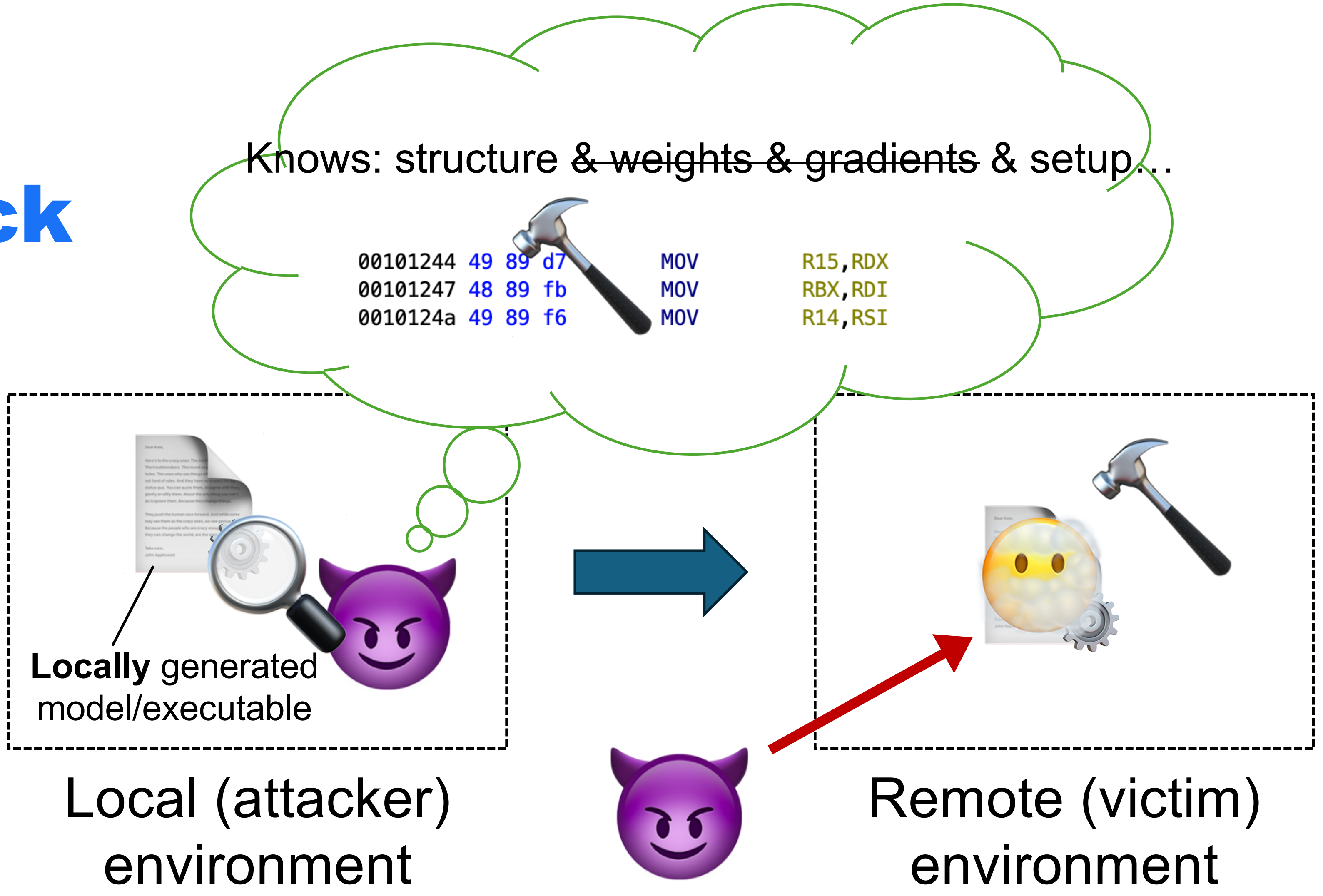
Attack Flow

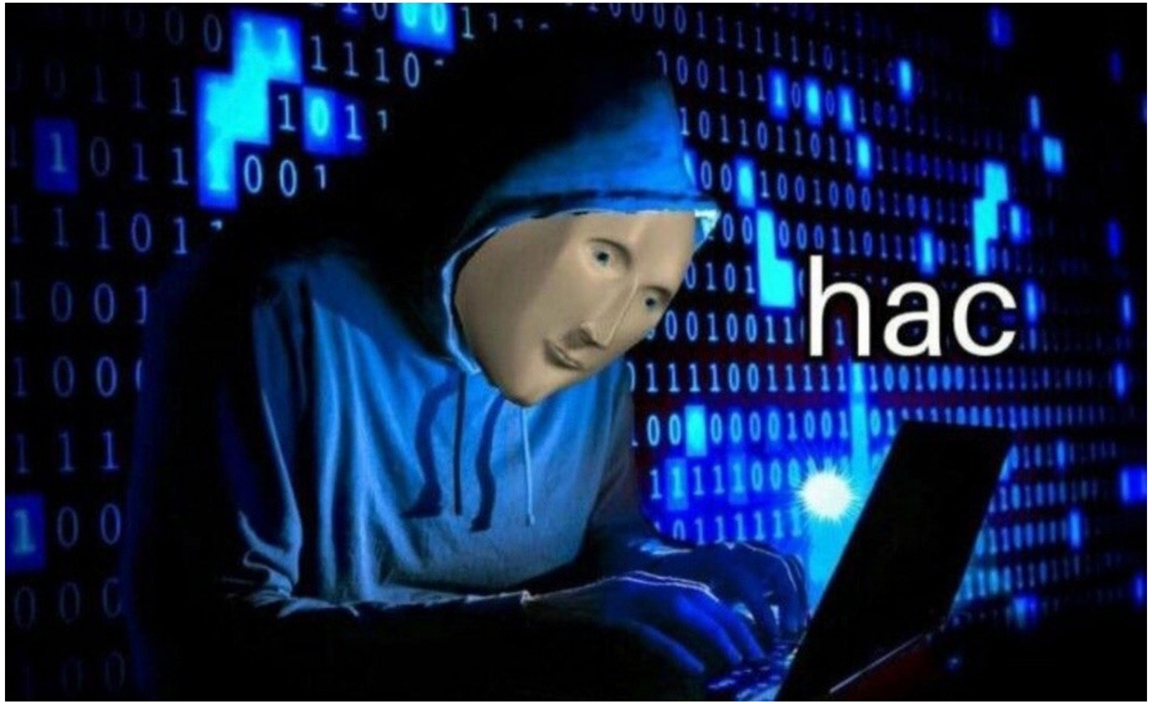

Knowledge
↓
Bits to flip
Locally generated model/executable
Local (attacker) environment
Remote (victim) environment

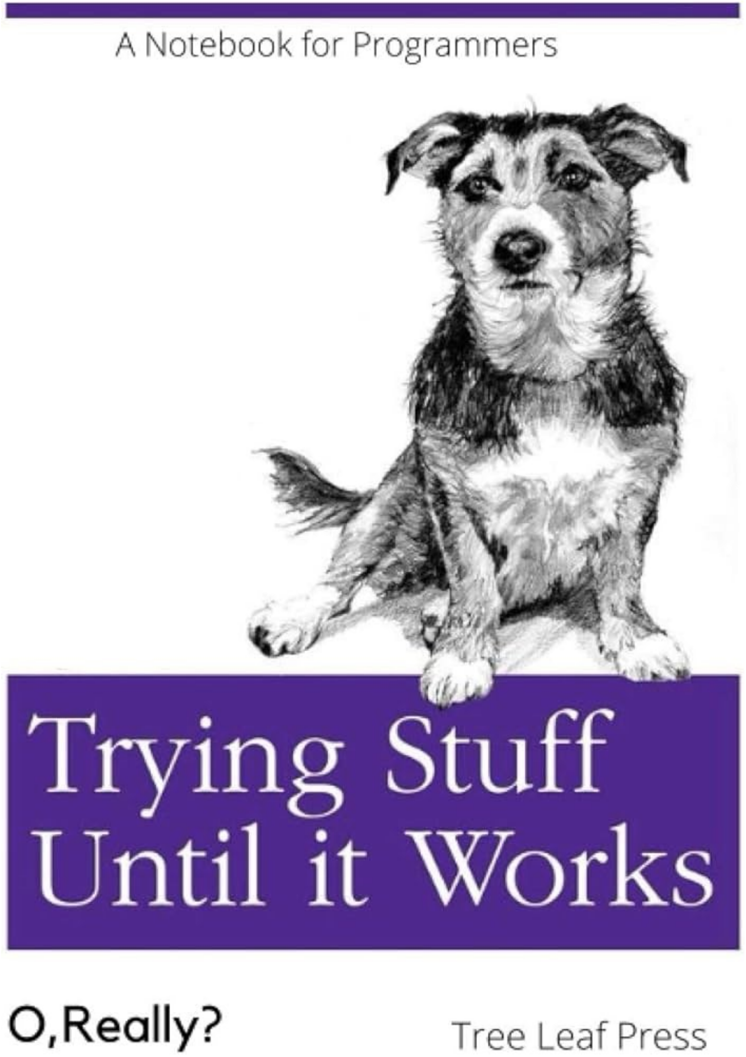
Previous Attacks



Our Attack


Knows: structure ~~& weights & gradients~~ & setup…
00101244 49 89 d7 MOV R15,RDX
00101247 48 89 fb MOV RBX,RDI
0010124a 49 89 f6 MOV R14,RSI
Locally generated model/executable
Local (attacker) environment
Remote (victim) environment


hac



- Randomly choose one bit within the code region
- Flip it
- See what happens
- 🔄 Loop

ASR: 2%

The Remaining 98%

- Most of them → Crash
- Some of them → No effect

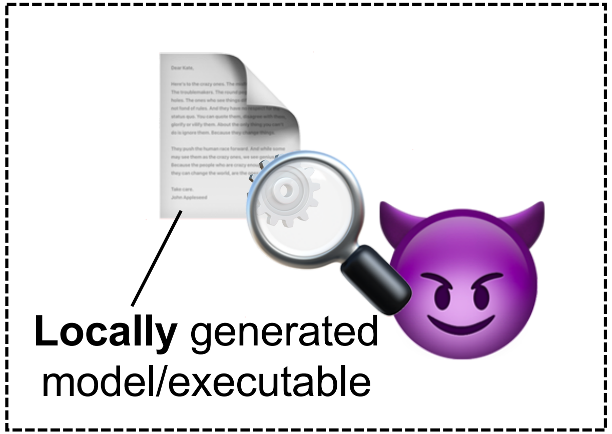
```
 segfault at 940c9 ip 00007f329a3df57b sp 00007f3299b54d10 error 6
 segfault at 73249 ip 00007f329a3df57b sp 00007f3298b52d10 error 6
 segfault at 20e09 ip 00007f329a3df57b sp 00007f329634dd10 error 6
 segfault at 523c9 ip 00007f329a3df57b sp 00007f3297b50d10 error 6
 segfault at ffffffffffffff89 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

```
                          LAB_00102476
00102476 31 c0              XOR        EAX,EAX
00102478 c5 f8 77           VZEROUPPER
0010247b c3                 RET
0010247c 0f                 ??         0Fh
0010247d 1f                 ??         1Fh
0010247e 40                 ??         40h    @
0010247f 00                 ??         00h
```

But: That 2%

Take 2: Using those 2% of bits



Local (attacker) environment

- Compile & train the model on an arbitrary dataset
  - Can't use victim dataset (we don't know it)
- Scan all bits and record those useful
- Remote: Try useful bits on victim executable

ASR: 45%

- 45% of time (or bits) lead to successful degradation
- Rest of the time: Crash or no effect

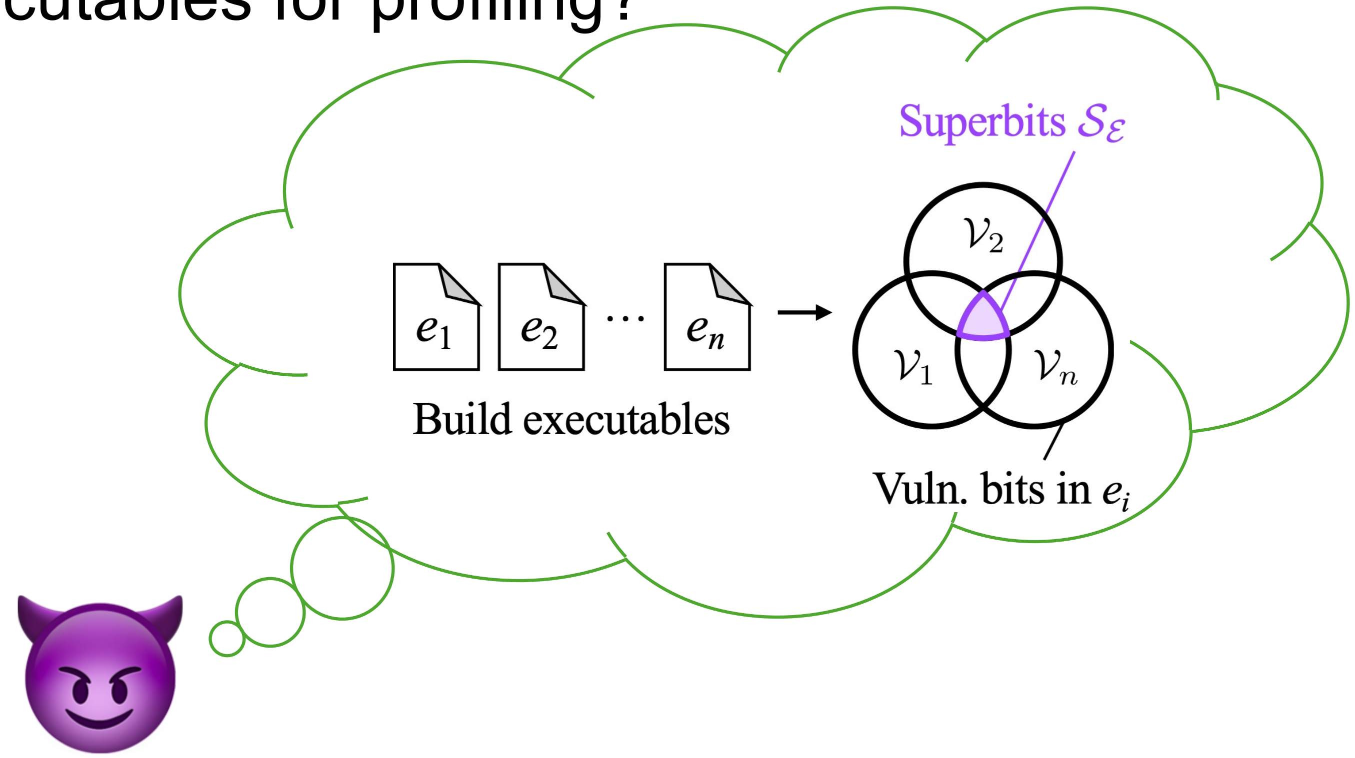
- *Why not 100% ASR?*
  - Model weights are different.

💡 Transferable vulnerable bits

45% vulnerable bits transferable to victim model,
despite different training sets

Take 3: In seek of “Superbits”

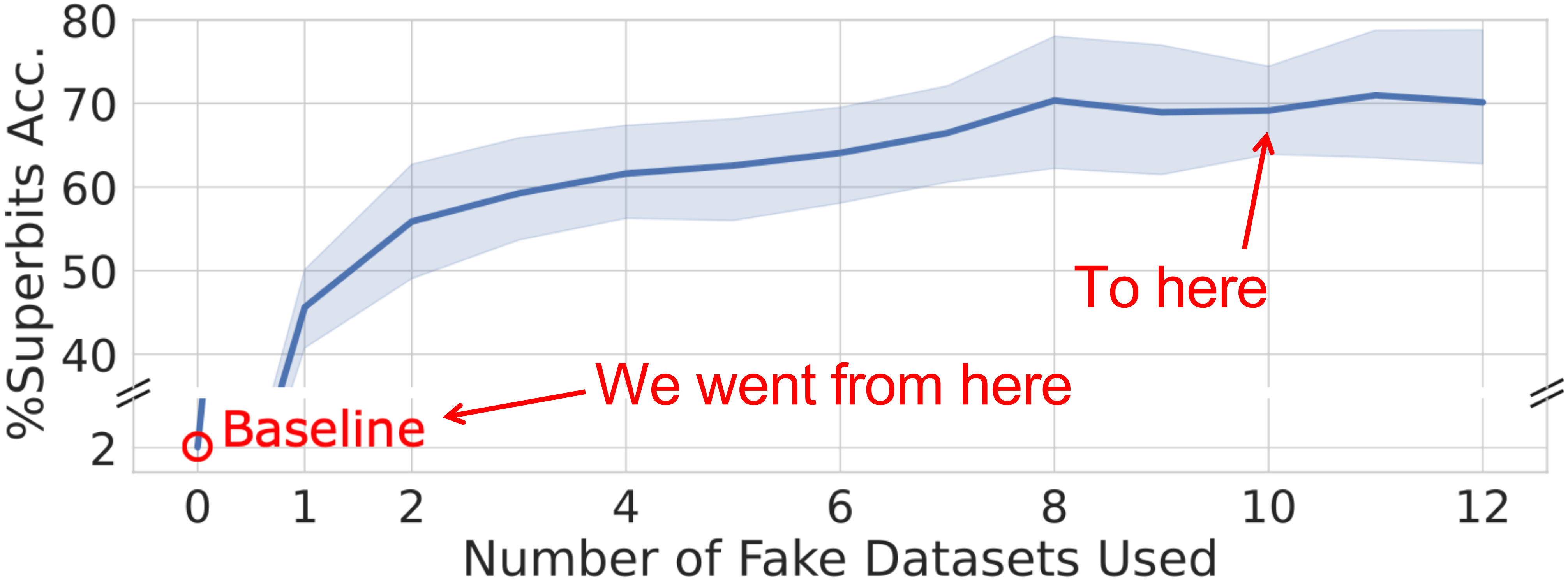
- Using *more* local executables for profiling?



Building More Local Executables

- Train them on datasets of random noise
  - Regulates weights
  - “Unbiased” choice
    - (More refs: yanzuo.ch/bh24)

ASR: 70%


%Superbits Acc.
80
70
60
50
40
2
0
1
2
4
6
8
10
12
Number of Fake Datasets Used
Baseline
We went from here
To here

Real World Experiments

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 Experiments

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

Avg: ~1.4 flips to success

Comparison: DeepHammer's Results

Dataset	Architecture	Network Parameters	Acc. before Attack (%)	Random Guess Acc. (%)	Acc. after Attack (%)	Min. # of Bit-flips
Fashion MNIST	LeNet	0.65M	90.20	10.00	10.00	**3**
Google Speech Command	VGG-11	132M	96.36	8.33	3.43	**5**
	VGG-13	133M	96.38		3.25	**7**
CIFAR-10	ResNet-20	0.27M	90.70	10.00	10.92	**21**
	AlexNet	61M	84.40		10.46	**5**
	VGG-11	132M	89.40		10.27	**3**
	VGG-16	138M	93.24		10.82	**13**
ImageNet	SqueezeNet	1.2M	57.00	0.10	0.16	**18**
	MobileNet-V2	2.1M	72.01		0.19	**2**
	ResNet-18	11M	69.52		0.19	**24**
	ResNet-34	21M	72.78		0.18	**23**
	ResNet-50	23M	75.56		0.17	**23**

Avg: ~12 flips

Bonus: Case Study

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.

In this case:

- Operand of *cmp* flipped
- Hard to defend with existing methods (e.g., optimisation)

- Learn more: yanzuo.ch/bh24

Black Hat Sound Bytes

- DeepCache: Optimisations gave away model architectures
- BFA: 6x fewer flips to ruin model intelligence
- More security research on DNN executables please

Thanks!

Yanzuo Chen
ychenjo@cse.ust.hk

Zhibo Liu
zhiboliu@ust.hk

Learn More
yanzuo.ch/bh24

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