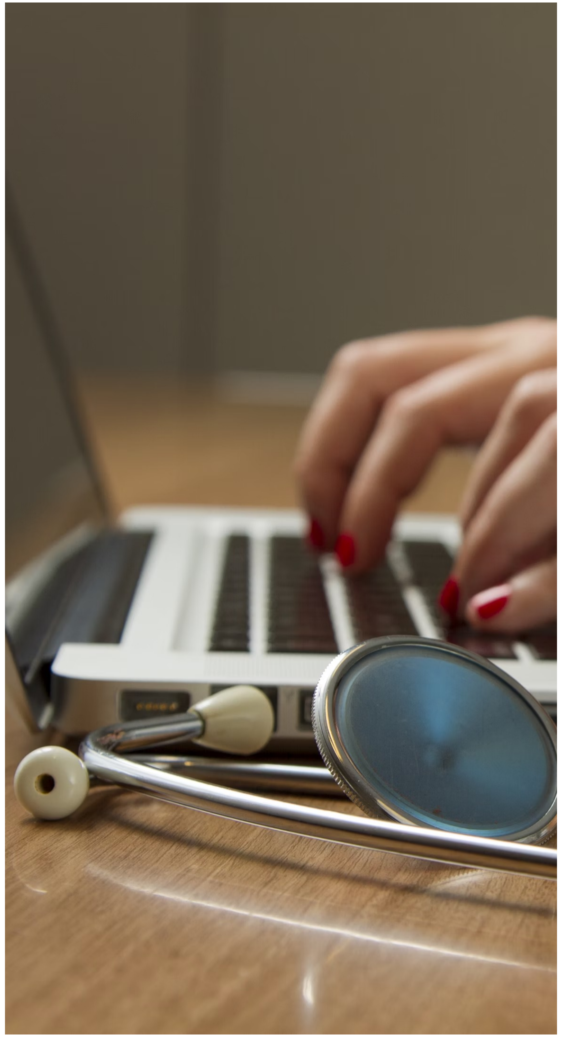
NDSS
SYMPOSIUM/2025

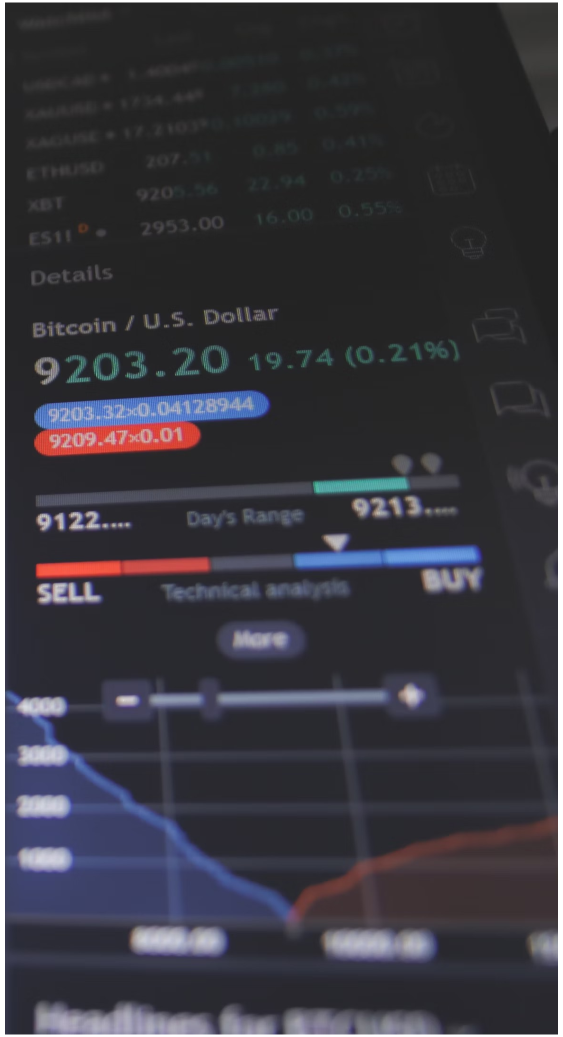
BitShield: Defending Against Bit-Flip Attacks on DNN Executables

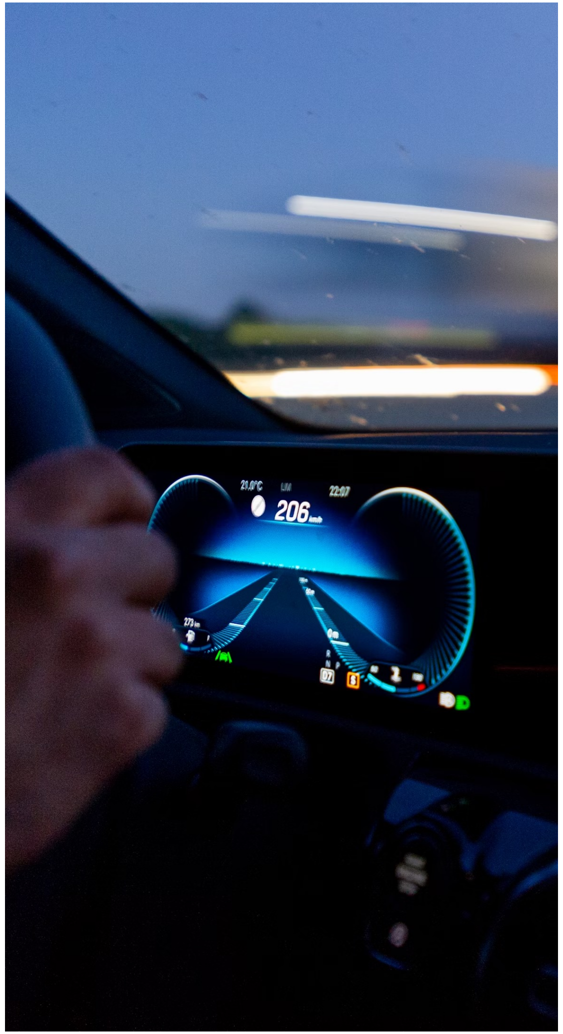
Yanzuo Chen[†], Yuanyuan Yuan[†], Zhibo Liu[†], Sihang Hu[‡], Tianxiang Li[‡], Shuai Wang[†]

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

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




ETHUSD 207.51 0.85 0.41%
XBT 9205.56 22.94 0.25%
2953.00 16.00 0.55%
Details
Bitcoin / U.S. Dollar
9203.20 19.74 (0.21%)
9203.32×0.04128944
9209.47×0.01
9122....
Day's Range
9213....
SELL
Technical analysis
BUY
More
4000
3000
2000
1000


206

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.

“High-Level” Attacks on AI/ML Systems

- Adversarial examples
- Backdoor
- Data poisoning
- ⋮

- Model stealing
- Model replication
- Membership inference
- ⋮

Bit-Flip Attacks (BFAs)

Bit-Flip Attacks (BFAs)

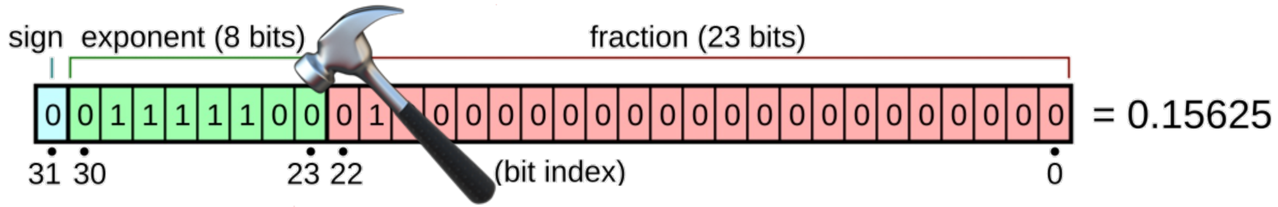
- Flipping data bits in the memory (DRAM)

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

BFAs on DNN Models

- Yes, researchers have done this
  - Model weights: IEEE 754 (full-precision) or integers (quantised)
  - Flipped bits $\rightarrow$ distorted weights $\rightarrow$ altered inference results
  - And there've also been many defences

$0 \rightarrow 1$



Not the Whole Story…

(Enter DNN executables)

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

BFAs on DNN Executables?

- DNN executables: compiled code (e.g., DNN operators)

- Current offensive research: attack surface overlooked
  - Only consider flips in model weights, not in code →

- Current defensive research: can't protect them
  - Only protect weight integrity & may be bypassed →

Note: We published the first offensive paper to close the gap also at NDSS 2025.

Dangerous Bit Flips in Code

	Model	Dataset	#Vuln	%Vuln
1	ResNet50	CIFAR10	12070	3.52
2	ResNet50	MNIST	13156	3.83
3	ResNet50	Fashion	14223	4.14
4	ResNet50	ImageNet	22008	4.79
5	GoogLeNet	CIFAR10	28926	2.97
6	GoogLeNet	MNIST	30401	3.13
7	GoogLeNet	Fashion	24381	2.51
8	DenseNet121	CIFAR10	40514	2.79
9	DenseNet121	MNIST	45369	3.13
10	DenseNet121	Fashion	44800	3.09
11	Q-ResNet50	CIFAR10	15846	2.17
12	Q-GoogLeNet	CIFAR10	11588	0.84
13	Q-DenseNet121	CIFAR10	13944	0.52
14	Avg.	-	-	2.88

- Pervasive
- Single-bit corruption
- Equally impact quantised models
  - (Previously considered more robust)

BFAs on DNN Executables?

- DNN executables: compiled code (e.g., DNN operators)

- Current offensive research: attack surface overlooked
  - Only consider flips in model weights, not in code →

- Current defensive research: can't protect them
  - Only protect weight integrity & may be bypassed →

Note: We published the first offensive paper to close the gap also at NDSS 2025.

Unprotected DNN Executables: An Example

Addr	Opcode bytes	x86 assembly instructions
0x98	F7 FE	idiv esi
0x9A	89 C3	mov ebx, eax
0X9C	44 8D 7F 01	lea r15d, [rdi + 0x1]
0xA0	44 0F AF F8	imul r15d, eax

(a) Assembly code before BFA.

0x98	F7 FE	idiv esi
0x9A	C9	leave ;; releases stack frame
0X9B	C3	ret ;; return to caller
0xA0	44 8D 7F 01	lea r15d, [rdi + 0x1]
0XA4	44 0F AF F8	imul r15d, eax

(b) Assembly code after BFA.

DNN Exe's: A More Demanding Case

- Bit flips in…
  - Weights (still works)
  - Code (new, more dangerous)

DNN Exe's: A More Demanding Case

- Bit flips in...
  - Weights (still works)
  - Code (new, more dangerous)
- Requirements for Defence

DNN Exe's: A More Demanding Case

- Bit flips in…
  - Weights (still works)
  - Code (new, more dangerous)
- Requirements for Defence
  - Unified, generic

DNN Exe's: A More Demanding Case

- Bit flips in...
  - Weights (still works)
  - Code (new, more dangerous)
- Requirements for Defence
  - Unified, generic
  - Self-defending

DNN Exe's: A More Demanding Case

- Bit flips in…
  - Weights (still works)
  - Code (new, more dangerous)

- Requirements for Defence
  - Unified, generic
  - Self-defending
  - Highly applicable

DNN Exe's: A More Demanding Case

- Bit flips in...
  - Weights (still works)
  - Code (new, more dangerous)

- Requirements for Defence
  - Unified, generic
  - Self-defending
  - Highly applicable
  - Performant

DNN Exe's: A More Demanding Case

- Bit flips in…
  - Weights (still works)
  - Code (new, more dangerous)

- Requirements for Defence
  - Unified, generic
  - Self-defending
  - Highly applicable
  - Performant
- Need a new defence that meet all of them!

(Always has been)

A Perspective of Semantics

A Perspective of Semantics

- DNN predictions: code logic + model weights

A Perspective of Semantics

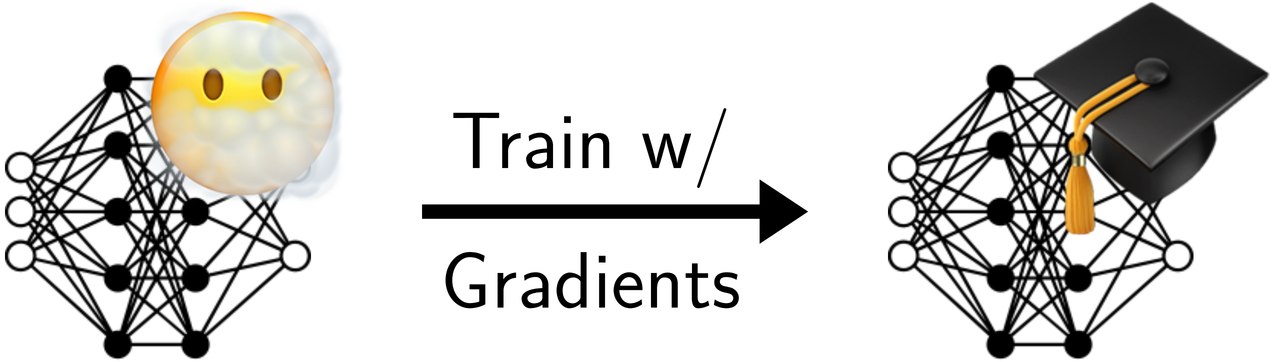
- DNN predictions: code logic + model weights
- BFAs (weights/code) are processes to change semantics

A Perspective of Semantics

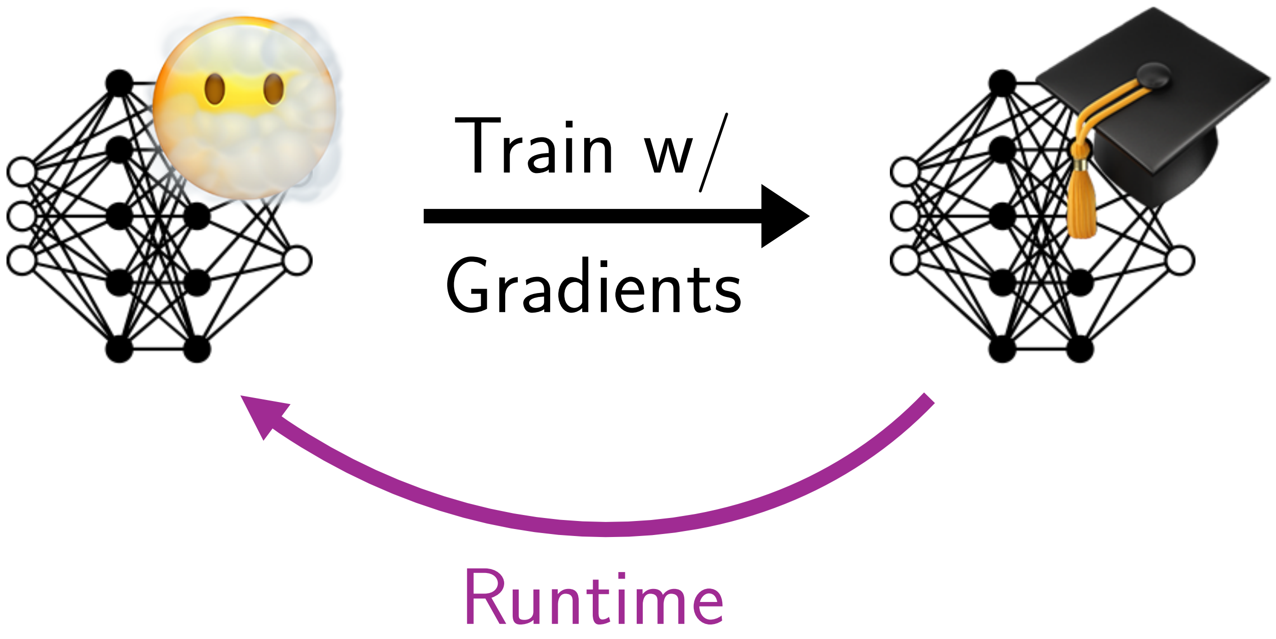
- DNN predictions: code logic + model weights
- BFAs (weights/code) are processes to change semantics
- But how to capture the semantics?

💡 Gradients

Gradients



Gradients



Capturing DNN Semantics

Capturing DNN Semantics

- Model output: y

Capturing DNN Semantics

- Model output: y
- Prepare vector $u = [\ {}^{1}/_{|u|},\ \ldots,\ {}^{1}/_{|u|}\ ]$

Capturing DNN Semantics

- Model output: y
- Prepare vector $u = [\ ^1/_{|u|}, \ldots, ^1/_{|u|}\ ]$
- Measure distance: $D_{KL}(u, y)$

Capturing DNN Semantics

- Model output: y
- Prepare vector $u = [\ {}^{1}/_{|u|}, \ldots, {}^{1}/_{|u|}\ ]$
- Measure distance: $D_{KL}(u, y)$
- Backpropagate to layer i: $\partial D_{KL}(u, y)/\partial W_i \ \rightarrow\ \ell_1$-norm

Capturing DNN Semantics

- Model output: y
- Prepare vector $u = [\ ^{1}/_{|u|}, \ldots, {}^{1}/_{|u|}\ ]$
- Measure distance: $D_{KL}(u, y)$
- Backpropagate to layer i: $\partial D_{KL}(u, y) / \partial W_i \ \rightarrow\ \ell_1$-norm

- Record normal semantics using training data

92.52% Mitigated

Weights-Based BFAs

Dealing with Code-Based BFAs

- Recall: Code flips may allow defence bypasses
- Just semantic checks are not enough

- $\Rightarrow$ Need more self-defence mechanisms

Adding Self-Defence

Adding Self-Defence

- "Avalanche effect" from cryptography
  - Slight disturbance gets amplified greatly

Adding Self-Defence

- "Avalanche effect" from cryptography
  - Slight disturbance gets amplified greatly

- Fuse code checksum into semantics calculation
  - Code flips $\rightarrow$ checksum $\rightarrow$ captured semantics

Fusing Code Checksum into Semantics

- Semantics capturing (simplified, w.l.o.g.): $o = W \star v$

NDSS SYMPOSIUM/2025

Fusing Code Checksum into Semantics

- Semantics capturing (simplified, w.l.o.g.): $o \;=\; W \star v$

$$o \;=\; \mathcal{M}^{-1}(c^*, \mathcal{M}(c_0, W)) \star v$$

Fusing Code Checksum into Semantics

- Semantics capturing (simplified, w.l.o.g.): $o = W \star v$

Masking/unmasking (inverse operations, e.g., XOR)

$$o = \boxed{\mathcal{M}^{-1}}(c^*, \boxed{\mathcal{M}}(c_0, W)) \star v$$

NDSS SYMPOSIUM/2025

Fusing Code Checksum into Semantics

- Semantics capturing (simplified, w.l.o.g.): $o = W \star v$

Masking/unmasking (inverse operations, e.g., XOR)

$$o = \boxed{\mathcal{M}^{-1}}(c^*, \boxed{\mathcal{M}}(c_0, W)) \star v$$

Runtime checksum

Embedded checksum

One More Thing

- Desirable to prevent potential damage early, if possible

- Checksum revisited: A checksum canary
  - Insert plain checksum checks!
  - $\rightarrow$ Halt execution upon mismatch

Evaluation

Threat Model & Setup

Threat Model & Setup

- All attackers are white-box, adaptive
  - Code-based attackers: Aggressive, stealthy
  - Weights-based attacker (existing SOTA)

Threat Model & Setup

- All attackers are white-box, adaptive
  - Code-based attackers: Aggressive, stealthy
  - Weights-based attacker (existing SOTA)

- 5 DRAM profiles (from existing surveys)

Threat Model & Setup

- All attackers are white-box, adaptive
  - Code-based attackers: Aggressive, stealthy
  - Weights-based attacker (existing SOTA)
- 5 DRAM profiles (from existing surveys)
- Metrics: Attack success rate, post-attack accuracy, overhead
  - Successful attack: Accuracy drop $\geq 3\%$

Results

Results

- Attack success rates
  - Code-based (both types): 100% → 0%
  - Weights-based: 96.24% → 7.48%

Results

- Attack success rates
  - Code-based (both types): 100% → 0%
  - Weights-based: 96.24% → 7.48%

- Post-attack accuracy
  - Code-based, aggressive: 12.69% → n/a
  - Code-based, stealthy: 80.37%% → n/a
  - Weights-based: 10.95% → 54.10%

Results

- Attack success rates
  - Code-based (both types): 100% $\rightarrow$ 0%
  - Weights-based: 96.24% $\rightarrow$ 7.48%

- Post-attack accuracy
  - Code-based, aggressive: 12.69% $\rightarrow$ n/a
  - Code-based, stealthy: 80.37%% $\rightarrow$ n/a
  - Weights-based: 10.95% $\rightarrow$ 54.10%

Model		Overhead (%)
ResNet50	CIFAR10	2.66
	MNIST	1.87
	Fashion	2.38
	ImageNet	8.22
	Avg.	**4.33**
GoogLeNet	CIFAR10	0.97
	MNIST	0.43
	Fashion	0.64
	Avg.	**0.68**
DenseNet121	CIFAR10	2.76
	MNIST	2.58
	Fashion	2.22
	Avg.	**2.52**
Avg.		**2.47**

Making Sense of the Results

- All code-based & 93% weights-based attacks mitigated
- ASRs decrease from 99% to 2%
- Remaining (few) successful attempts limited greatly
- Low overhead for practical use (2%)

In This Talk

- BFAs on DNN executables and challenges for defences
- Semantic-based defence to protect against old & new attacks
- Highly effective, low overhead method

Thank You!

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

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

BitShield: Defending Against Bit-Flip Attacks on DNN Executables.
By Yanzuo Chen, Yuanyuan Yuan, Zhibo Liu, Sihang Hu, Tianxiang Li, and Shuai Wang.

TABLE IV

ATTACK RESULTS ON VANILLA DNN EXECUTABLES WITHOUT PROTECTION.

Attack Success Rate (%)

Attacker Type	ResNet50				GoogLeNet			DenseNet121			Avg.
	CIFAR10	MNIST	Fashion	ImageNet	CIFAR10	MNIST	Fashion	CIFAR10	MNIST	Fashion	
Aggressive code-based	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Stealthy code-based	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Weights-based	98.80	91.60	94.00	96.00	98.80	92.40	96.80	98.00	97.20	98.80	96.24
Avg.	99.60	97.20	98.00	98.67	99.60	97.47	98.93	99.33	99.07	99.60	98.75

Accuracy after Attack (%)

Attacker Type	ResNet50				GoogLeNet			DenseNet121			Avg.
	CIFAR10	MNIST	Fashion	ImageNet	CIFAR10	MNIST	Fashion	CIFAR10	MNIST	Fashion	
Aggressive code-based	18.09	13.85	15.31	2.59	12.11	11.80	12.61	11.98	13.31	15.26	12.69
Stealthy code-based	82.17	89.90	78.54	63.46	72.30	82.44	83.75	74.19	91.83	85.14	80.37
Weights-based	18.26	11.07	10.17	2.94	12.95	10.28	10.45	10.79	11.40	11.17	10.95

TABLE V

ATTACK RESULTS ON DNN EXECUTABLES PROTECTED BY BITSHIELD.

Attack Success Rate (%)

Attacker Type	ResNet50				GoogLeNet			DenseNet121			Avg.
	CIFAR10	MNIST	Fashion	ImageNet	CIFAR10	MNIST	Fashion	CIFAR10	MNIST	Fashion	
Aggressive code-based	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Stealthy code-based	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Weights-based	16.00	1.20	3.20	6.40	24.80	1.60	1.20	2.80	12.80	4.80	7.48
Avg.	5.33	0.40	1.07	2.13	8.27	0.53	0.40	0.93	4.27	1.60	2.49

Accuracy after Attack (%)

Attacker Type	ResNet50				GoogLeNet			DenseNet121			Avg.
	CIFAR10	MNIST	Fashion	ImageNet	CIFAR10	MNIST	Fashion	CIFAR10	MNIST	Fashion	
Aggressive code-based	-	-	-	-	-	-	-	-	-	-	-
Stealthy code-based	-	-	-	-	-	-	-	-	-	-	-
Weights-based	66.35	45.64	31.25	51.54	74.84	90.00	68.72	40.00	37.31	35.38	54.10

TABLE VI
BREAKDOWN OF THE ATTACK OUTCOMES ON PROTECTED RESNET50(RN), GOOGLENET(GN), AND DENSENET121(DN).

Attacker	**Outcome**	**Models**			**Sum (Proportion)**
		RN	**GN**	**DN**	
Code-based	Profiling failed	422	1062	862	2346 (31.28%)
	SIG	386	342	526	1254 (16.72%)
	Canary	1049	96	112	1257 (16.76%)
	Accuracy	143	0	0	143 (1.91%)
	Success	0	0	0	0 (0%)
Weights-based	Profiling failed	49	24	15	88 (1.17%)
	SIG	884	657	684	2225 (29.67%)
	Accuracy	0	0	0	0 (0%)
	Success	67	69	51	187 (2.49%)
Sum		3000	2250	2250	7500 (100.00%)

TABLE VII
EFFECTS OF DIFFERENT e VALUES.

e	Model	FA (%)	MF (%)	ΔASR (%)		
				Code-based	Weights-based	Avg.
0.0	ResNet50	0.00	6.93	0.00	0.00	0.00
	GoogLeNet	0.20	6.16	0.00	0.00	0.00
0.3	ResNet50	0.00	1.32	-	-	-
	GoogLeNet	0.00	0.01	-	-	-
0.4	ResNet50	0.00	0.81	0.00	0.00	0.00
	GoogLeNet	0.00	0.00	0.00	0.00	0.00
0.5	ResNet50	0.00	0.37	0.00	0.00	0.00
	GoogLeNet	0.00	0.00	0.00	0.00	0.00
0.6	ResNet50	0.00	0.27	0.00	0.00	0.00
	GoogLeNet	0.00	0.00	0.00	0.00	0.00
1.0	ResNet50	0.00	0.00	0.00	0.00	0.00
	GoogLeNet	0.00	0.00	0.00	+8.00	+4.00

1) ResNet50 and GoogLeNet are trained on CIFAR10 and MNIST datasets.
2) FA: false alarm of test inputs, MF: mis-flag of inputs from other datasets.
3) ΔASR: changed ASR w.r.t. $e = 0.3$ in main experiments.

TABLE VIII

COMPARISON WITH PRIOR DEFENSES ON ADAPTIVE WEIGHTS-BASED ATTACKS. ONLY WEIGHTS-BASED ATTACKS ARE CONSIDERED, AS NONE OF THE PREVIOUS METHODS PROTECT AGAINST CODE-BASED BFAS.

Work	**Method**	**Performance overhead (%)**	**Acc. loss (%)**	**Mitigation rate (%)**
Aegis [46]	Enhance structure	NA (< 0)	1.24	63.76
DeepAttest [3]	Fingerprint	7.20	≤ 0.09	90.00
NeuroPots [30]	Enhance weights + fingerprint	3.93	1.38	100.00
Ours	Semantic integrity	2.47	NA (0)	92.52