Double Strike: Breaking Approximation-Based Side-Channel Countermeasures for DNNs

SemSecuElec Seminar

Lorenzo CASALINO, Maria Mendez RÉAL, Jean-Christophe PRÉVOTET, Rubén SALVADOR

CentraleSupélec, IRISA, Inria (SUSHI Team)
name.surname@irisa.fr
Lab-STICC, Univ. Bretagne-Sud, UMR 6285
maria.mendez-real@univ-ubs.fr
Univ. Rennes, INSA Rennes, CNRS, IETR-UMR 6164
jean-christophe.prevotet@insa-rennes.fr

20 Mar. 2026

ANR-21-CE39-0018

Agenda

1. Background
2. Methodology
3. Results
4. Conclusions

Table of Contents

Deep Neural Networks (DNNs)

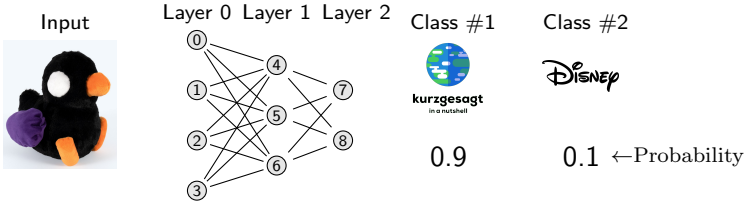
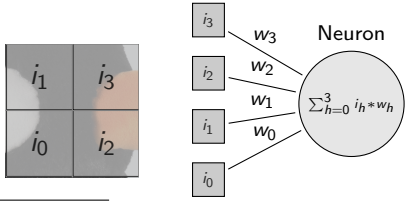


Figure: A DNN brand classifier[1].



[1]Duck and Kurzgesagt Logo belong to Kurzgesagt

The *Weight*(s) of an Economic Damage

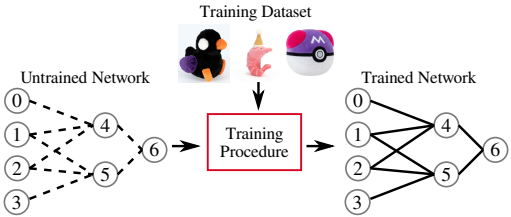


Figure: The training process.[2]

DNN Training is Expensive

- Expensive hardware (e.g., GPUs), time-intensive (e.g., days)

Weights Piracy

A non-negligible **economic damage**

[2]Duck (Kurzgesagt), Shrimp (Jellycat London), Masterball (Nintendo)

Side-Channel Analysis

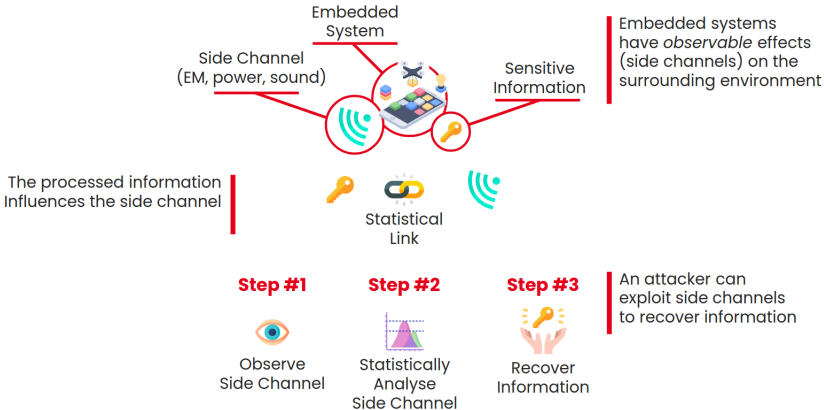
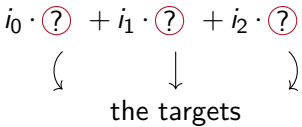
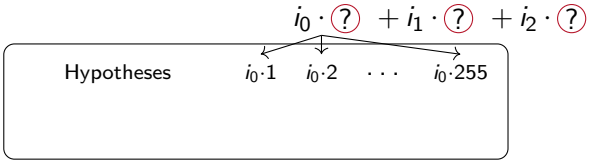


Figure: Side-Channel Analysis.

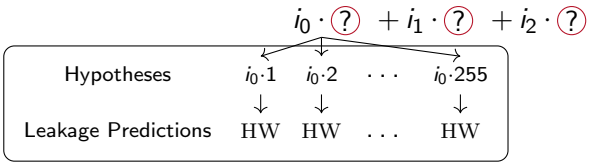
Side-channel Driven Weight Extraction



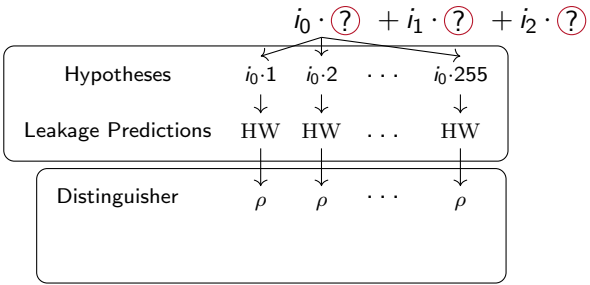
Side-channel Driven Weight Extraction



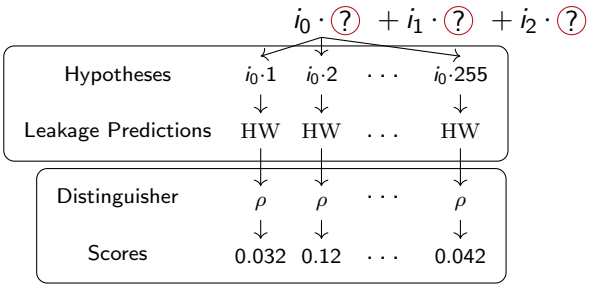
Side-channel Driven Weight Extraction



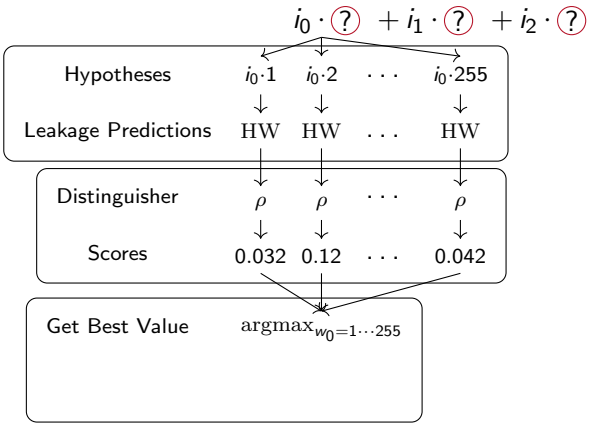
Side-channel Driven Weight Extraction



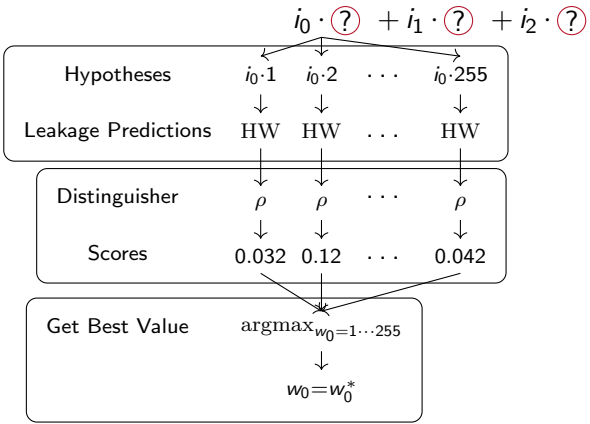
Side-channel Driven Weight Extraction



Side-channel Driven Weight Extraction



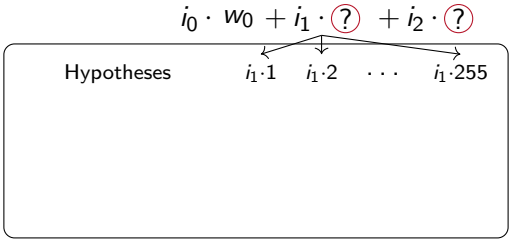
Side-channel Driven Weight Extraction



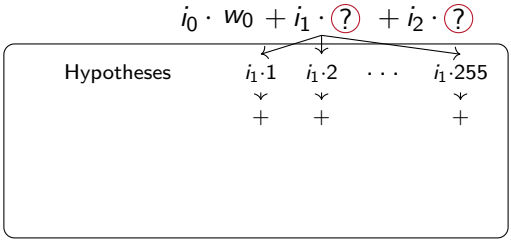
Side-channel Driven Weight Extraction

$$i_0 \cdot w_0 + i_1 \cdot \textcircled{?} + i_2 \cdot \textcircled{?}$$

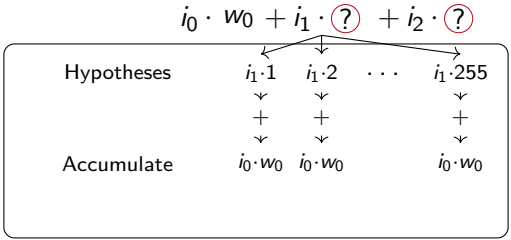
Side-channel Driven Weight Extraction



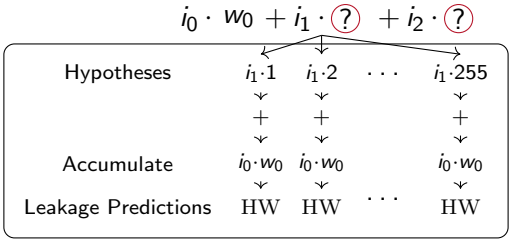
Side-channel Driven Weight Extraction



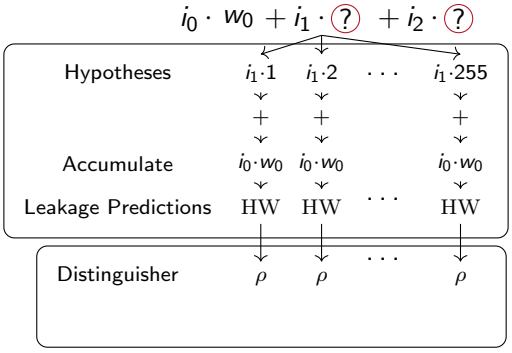
Side-channel Driven Weight Extraction



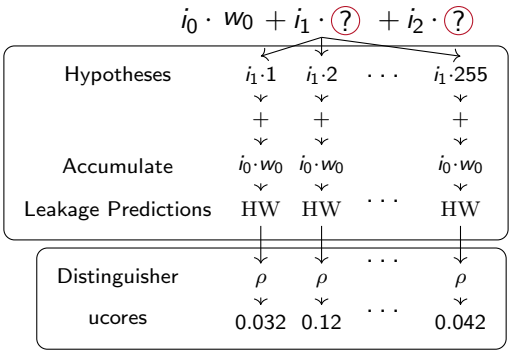
Side-channel Driven Weight Extraction



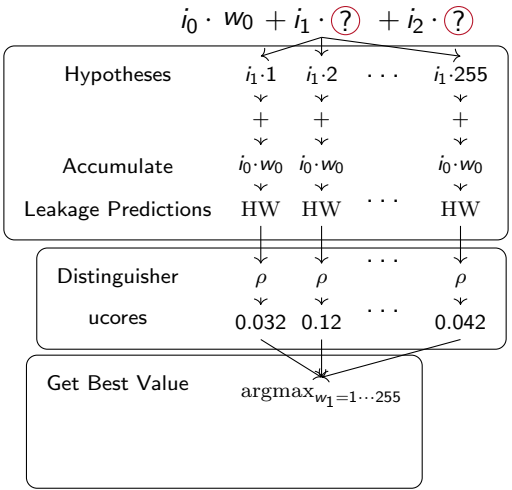
Side-channel Driven Weight Extraction



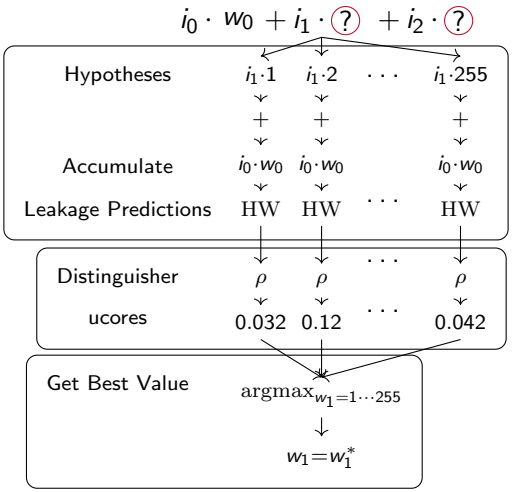
Side-channel Driven Weight Extraction



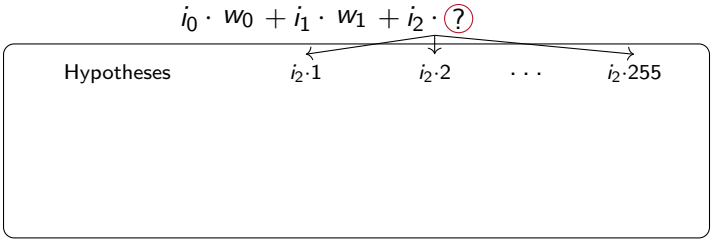
Side-channel Driven Weight Extraction



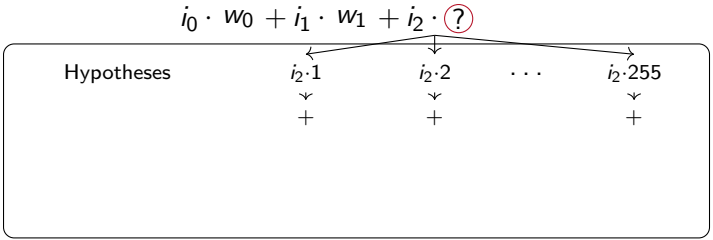
Side-channel Driven Weight Extraction



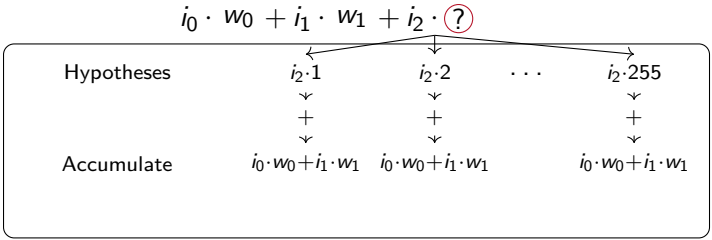
Side-channel Driven Weight Extraction



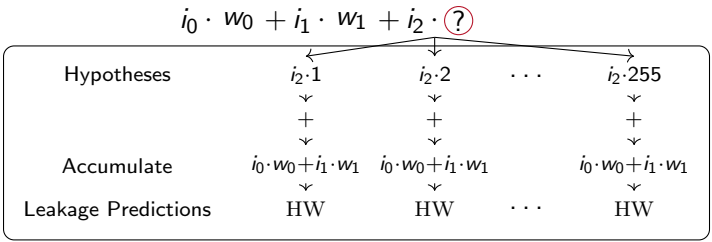
Side-channel Driven Weight Extraction



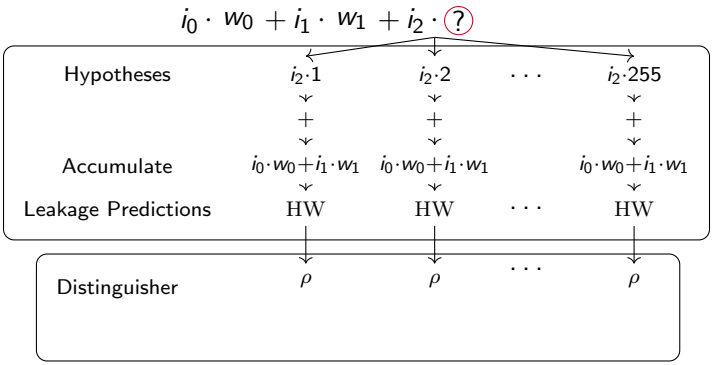
Side-channel Driven Weight Extraction



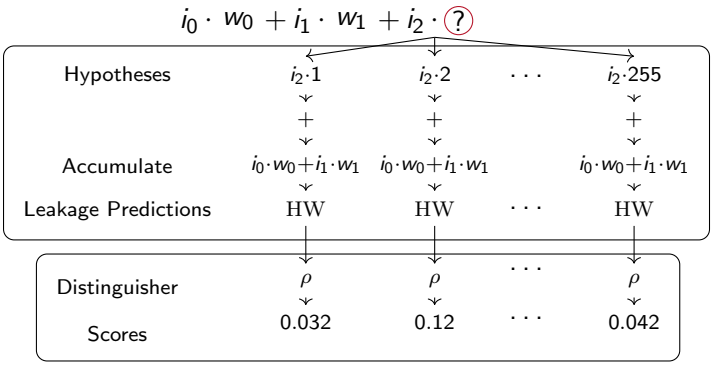
Side-channel Driven Weight Extraction



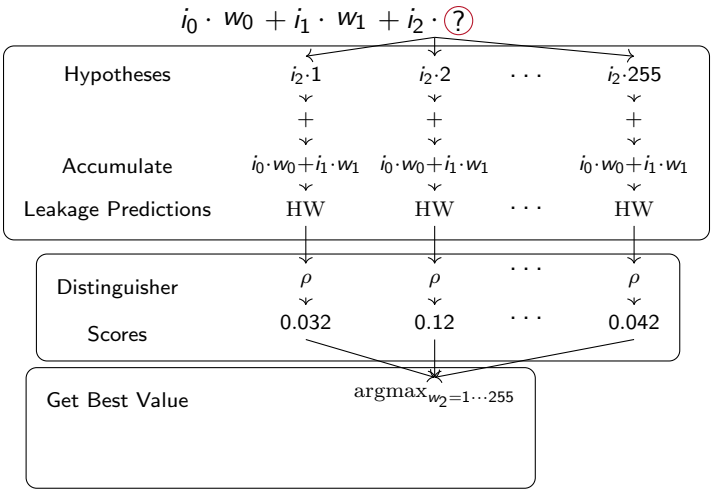
Side-channel Driven Weight Extraction



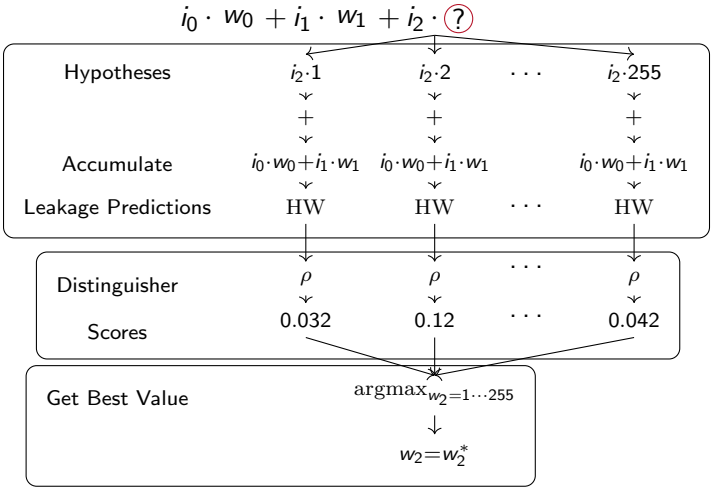
Side-channel Driven Weight Extraction



Side-channel Driven Weight Extraction



Side-channel Driven Weight Extraction



Side-channel Driven Weight Extraction

$$i_0 \cdot w_0 + i_1 \cdot w_1 + i_2 \cdot w_2$$

Countermeasures

Masking

Replace the weight-dependent signal with N random ones

Shuffling

Randomly shuffle operations to bury weight-dependent signal in signal noise

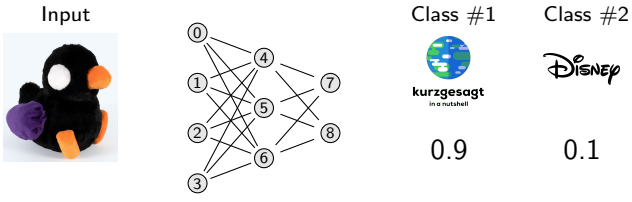
DNN-Tailored Countermeasures

- Current defences come from cryptanalysis
- But DNNs $\neq$ cryptosystems!
- DNNs exhibit particular characteristics (e.g., error resilience)

Approximate Computing

A Definition [Leo+25]

Intentional insertion of errors to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.



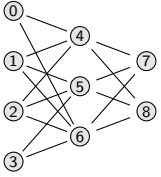
Original Network

Approximate Computing

A Definition [Leo+25]

Intentional insertion of errors to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Input



Class #1

kurzgesagt
in a nutshell

0.7

Class #2

Disney

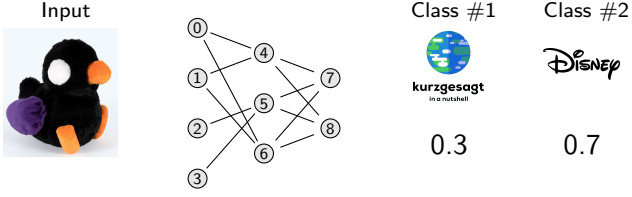
0.3

Pruned Network

Approximate Computing

A Definition [Leo+25]

Intentional insertion of errors to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.



Approximate Computing and Side-Channel Behaviour

A Definition [Leo+25]

Intentional insertion of error to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Approximate-Computing (AxC)-based Countermeasures

- Approximation changes the application behaviour

Approximate Computing and Side-Channel Behaviour

A Definition [Leo+25]

Intentional insertion of error to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Approximate-Computing (AxC)-based Countermeasures

- Approximation changes the application behaviour
- Approximation changes the application's *side-channel signature*

Approximate Computing and Side-Channel Behaviour

A Definition [Leo+25]

Intentional insertion of error to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Approximate-Computing (AxC)-based Countermeasures

- Approximation changes the application behaviour
- Approximation changes the application's *side-channel signature*
- What role does AxC play in side-channel resilience? [Jap+25; Zha+26]

Approximate Computing and Side-Channel Behaviour

A Definition [Leo+25]

Intentional insertion of error to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Approximate-Computing (AxC)-based Countermeasures

- Approximation changes the application behaviour
- Approximation changes the application's *side-channel signature*
- What role does AxC play in side-channel resilience? [Jap+25; Zha+26]
  - Core question of the ANR "ATTILA" project

Approximate Computing and Side-Channel Behaviour

A Definition [Leo+25]

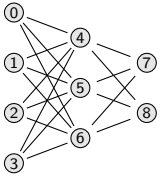
Intentional insertion of error to **trade** *tunable* **accuracy** *loss* **for** *valuable* **resource gains**.

Approximate-Computing (AxC)-based Countermeasures

- Approximation changes the application behaviour
- Approximation changes the application's *side-channel signature*
- What role does AxC play in side-channel resilience? [Jap+25; Zha+26]
  - Core question of the ANR "ATTILA" project
- Recently considered as a side-channel countermeasure [Din+25]

Pixel Importance

Input



Class #1

kurzgesagt
in a nutshell

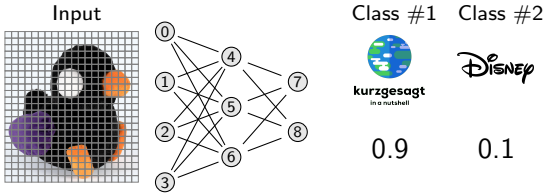
0.9

Class #2

Disney

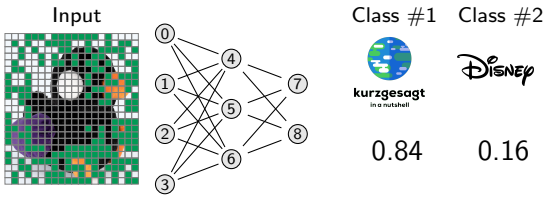
0.1

Pixel Importance



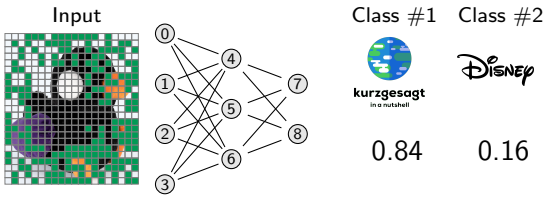
- What happens if we *prune* (skip) pixels?

Pixel Importance



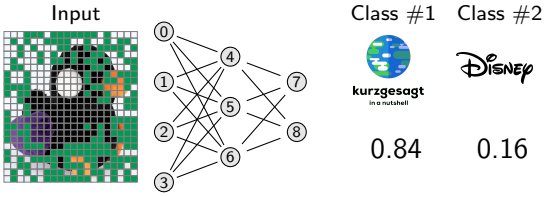
- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction

Pixel Importance



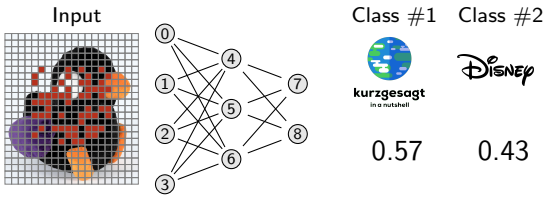
- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction
  - Non-important weight

Pixel Importance



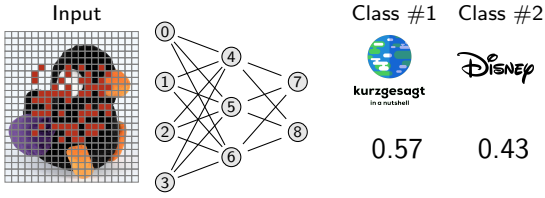
- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction
  - Non-important weight
  - Non-important Multiply-and-Accumulate (NIMAC)

Pixel Importance



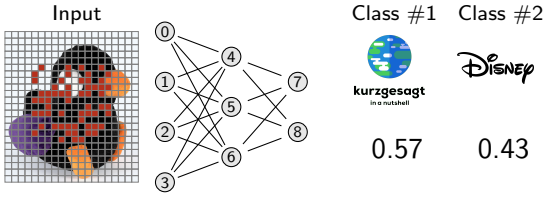
- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction
  - Non-important weight
  - Non-important Multiply-and-Accumulate (NIMAC)
- Important pixel: important accuracy reduction

Pixel Importance



- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction
  - Non-important weight
  - Non-important Multiply-and-Accumulate (NIMAC)
- Important pixel: important accuracy reduction
  - Important weight

Pixel Importance

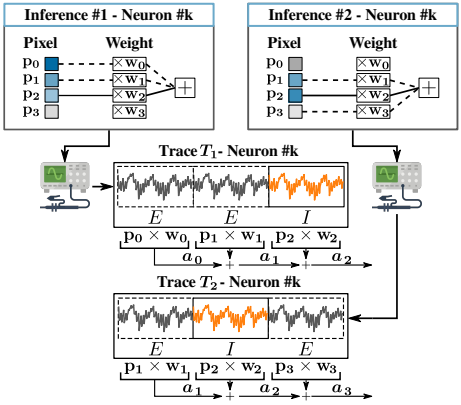


- What happens if we *prune* (skip) pixels?
- Non-important pixel: negligible accuracy reduction
  - Non-important weight
  - Non-important Multiply-and-Accumulate (NIMAC)
- Important pixel: important accuracy reduction
  - Important weight
  - Important MAC (IMAC)

MACPruning

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

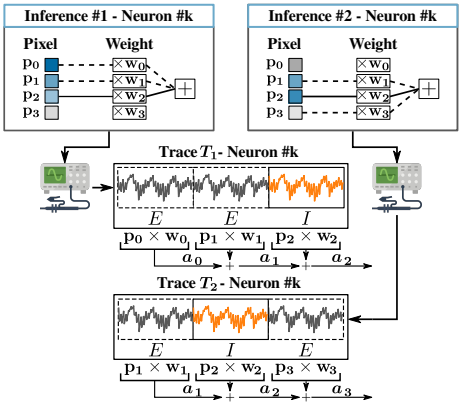


MACPruning

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

Effects



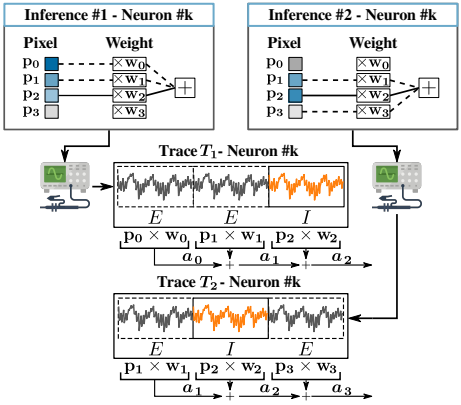
MACPruning

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

Effects

- **Incorrect leakage predictions**,



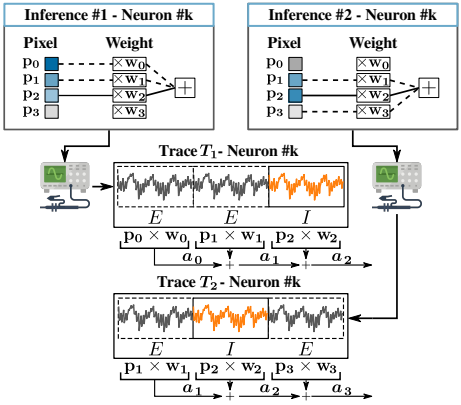
MACPruning

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

Effects

- **Incorrect leakage predictions**,
- **Desynchronised traces**.



MACPruning

The Idea

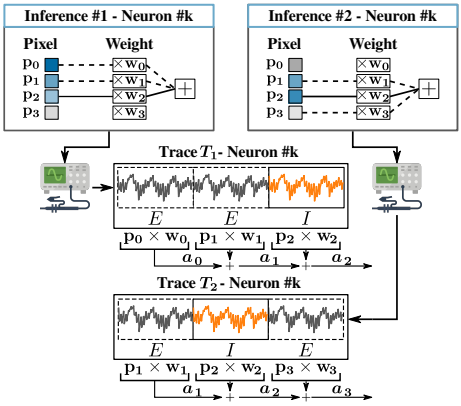
At each inference, randomly skip (*prune*) non-important pixels.

Effects

- **Incorrect leakage predictions**,
- **Desynchronised traces**.

Security Implications

Successful weight recovery requires exponential number of traces.



MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
   |
   |
   |
   |
   |
   |
   |
   |
   |
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
   |
   |
   |
   |
   |
   |
   |
   |
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
   |
   |
   |
   |
   |
   |
   |
   |
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,
- randWords: random binary vector,

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
   |
   |
   |
   |
   |
   |
   |
   |
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,
- randWords: random binary vector,
- Compute pixel if and only if:

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,
- randWords: random binary vector,
- Compute pixel if and only if:
  - i-th pixel is important (Line 3–5) or,

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
6      else if randWords[i] then
7          m ← inputs[i] · weights[i];
8          acc ← acc + m;
10 end
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,
- randWords: random binary vector,
- Compute pixel if and only if:
  - i-th pixel is important (Line 3–5) or,
  - i-th pixel must be executed (Line 6–8),

MACPruning Implementation

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
6      else if randWords[i] then
7          m ← inputs[i] · weights[i];
8          acc ← acc + m;
9      else
           /* Skip MAC                    */
10 end
11 return acc
```

The Idea

At each inference, randomly skip (*prune*) non-important pixels.

The Implementation Logic

- IaPAM: pixel $\mapsto$ importance,
- randWords: random binary vector,
- Compute pixel if and only if:
  - i-th pixel is important (Line 3–5) or,
  - i-th pixel must be executed (Line 6–8),
- Control-flow-dependent check.

Control-flow Dependency and Side-channel Information

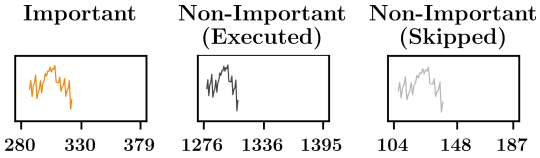
Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
   |
   |
   |
   |
   |
   |
   |
   |
9  |  else
   |  |  /* Skip MAC                    */
10 end
11 return acc
```

Control-flow Dependency and Side-channel Information

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

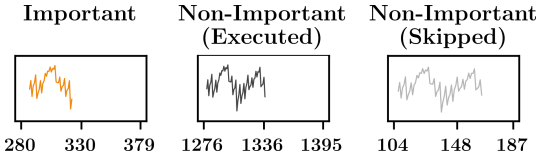
```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
9      else
           /* Skip MAC                    */
10 end
11 return acc
```



Control-flow Dependency and Side-channel Information

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

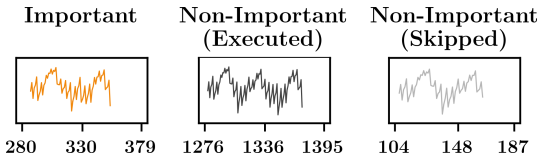
```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
6      else if randWords[i] then
9      else
           /* Skip MAC                    */
10 end
11 return acc
```



Control-flow Dependency and Side-channel Information

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

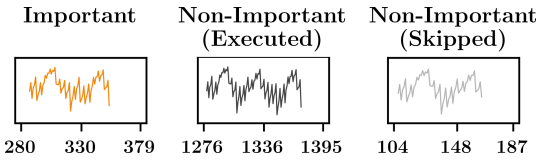
```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
6      else if randWords[i] then
7          m ← inputs[i] · weights[i];
8          acc ← acc + m;
9      else
           /* Skip MAC                    */
10 end
11 return acc
```



Control-flow Dependency and Side-channel Information

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
6      else if randWords[i] then
7          m ← inputs[i] · weights[i];
8          acc ← acc + m;
9      else
           /* Skip MAC                    */
10 end
11 return acc
```



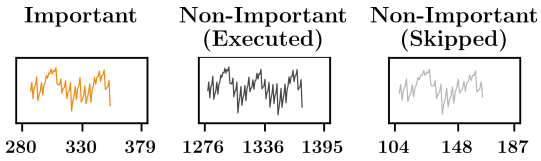
Side-Channel Information

Control-flow dependency hints each pixel' importance

Control-flow Dependency and Side-channel Information

Input : *inputs*: $(w \times h)$-array of pixels
weights: $(w \times h)$-array of weights
IaPAM: $(w \times h)$-array of bits
randWords: $(w \times h)$-array of bits
Output: *acc*: accumulator

```
1  acc ← 0;
2  for i from 0 to length(inputs) do
3      if IaPAM[i] then
4          m ← inputs[i] · weights[i];
5          acc ← acc + m;
6      else if randWords[i] then
7          m ← inputs[i] · weights[i];
8          acc ← acc + m;
9      else
           /* Skip MAC                    */
10 end
11 return acc
```



Side-Channel Information

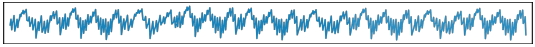
Control-flow dependency hints each pixel' importance

Question

What are the security implications?

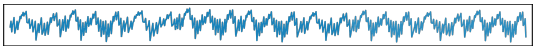
Control-flow Dependency and Security Implications

Given a side-channel trace ...

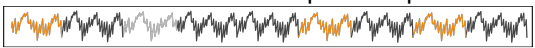


Control-flow Dependency and Security Implications

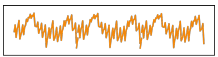
Given a side-channel trace ...



... we extract the important pixels.



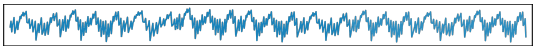
i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8



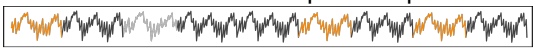
i_0 i_5 i_7

Control-flow Dependency and Security Implications

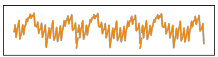
Given a side-channel trace ...



... we extract the important pixels.



i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8



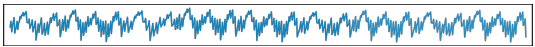
i_0 i_5 i_7

↓

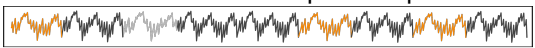
Circumvent trace desynchronisation

Control-flow Dependency and Security Implications

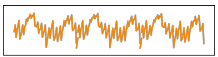
Given a side-channel trace ...



... we extract the important pixels.



i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8



i_0 i_5 i_7

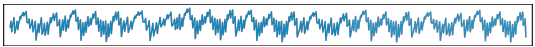
↓

Circumvent trace desynchronisation

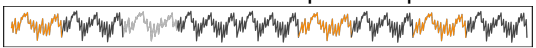
... we remove skipped pixels.

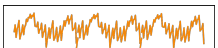
Control-flow Dependency and Security Implications

Given a side-channel trace ...



... we extract the important pixels.





↓

Circumvent trace desynchronisation

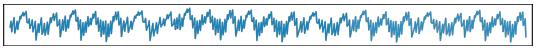
... we remove skipped pixels.

↓

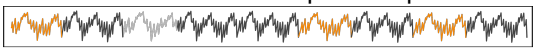
Compute correct leakage predictions

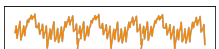
Control-flow Dependency and Security Implications

Given a side-channel trace ...



... we extract the important pixels.





↓

Circumvent trace desynchronisation

... we remove skipped pixels.

↓

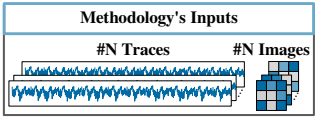
Compute correct leakage predictions

Research Question

Can we circumvent the MACPruning countermeasure?

Table of Contents

A Preprocessing Methodology to Circumvent MACPruning



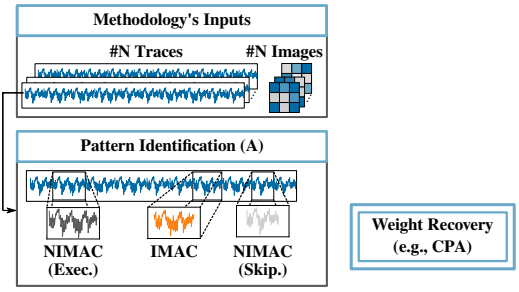
Goal

Recover the Important Weights.

Methodology

Figure: Preprocessing Methodology.

A Preprocessing Methodology to Circumvent MACPruning



Goal

Recover the Important Weights.

Methodology

A. Identify Side-channel Patterns;

Figure: Preprocessing Methodology.

A Preprocessing Methodology to Circumvent MACPruning

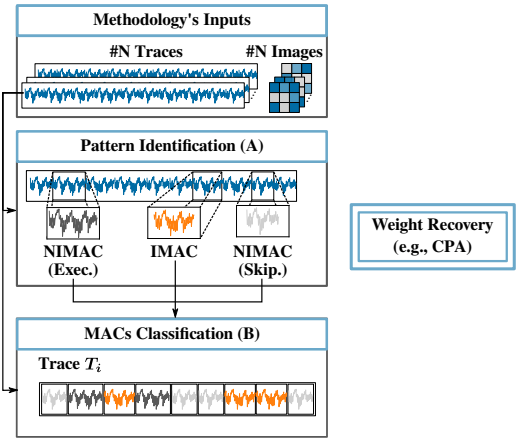


Figure: Preprocessing Methodology.

Goal

Recover the Important Weights.

Methodology

A. Identify Side-channel Patterns;

B. Identify MAC;/Inputs importance in traces;

A Preprocessing Methodology to Circumvent MACPruning

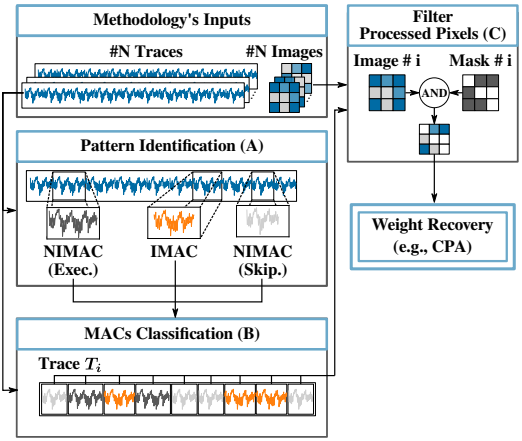


Figure: Preprocessing Methodology.

Goal

Recover the Important Weights.

Methodology

A. Identify Side-channel Patterns;

B. Identify MAC;/Inputs importance in traces;

C. Remove skipped pixels from images;

A Preprocessing Methodology to Circumvent MACPruning

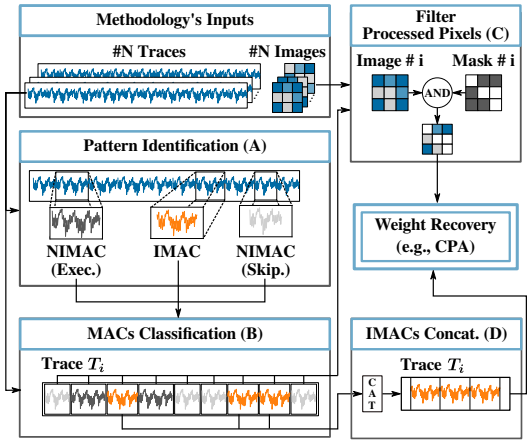


Figure: Preprocessing Methodology.

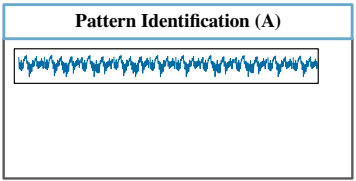
Goal

Recover the Important Weights.

Methodology

A. Identify Side-channel Patterns;
B. Identify MAC;/Inputs importance in traces;
C. Remove skipped pixels from images;
D. Preserve only the important MACs.

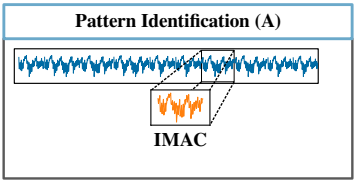
Step A – Pattern Identification



Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:

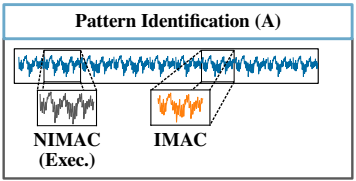
Step A – Pattern Identification



Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);

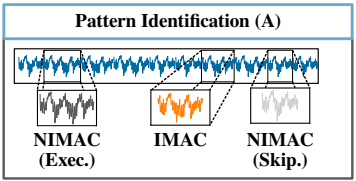
Step A – Pattern Identification



Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);

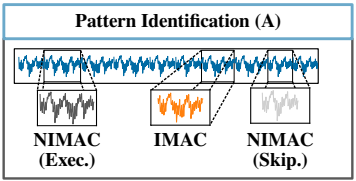
Step A – Pattern Identification



Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs

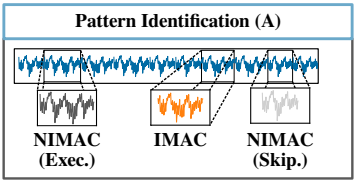
Step A – Pattern Identification



Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs
- Manual or automated search

Step A – Pattern Identification

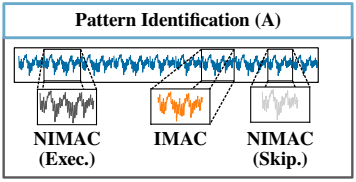


Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs
- Manual or automated search

Multiple patterns

Step A – Pattern Identification



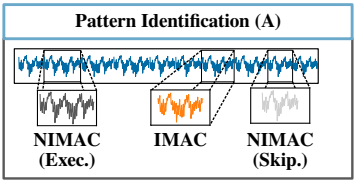
Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs
- Manual or automated search

Multiple patterns

- **Caveat:** one MAC may have multiple patterns

Step A – Pattern Identification



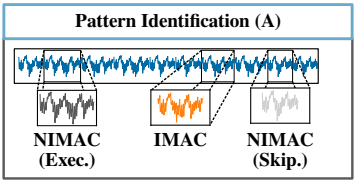
Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs
- Manual or automated search

Multiple patterns

- **Caveat:** one MAC may have multiple patterns
  - Multiple instructions issue
  - Out-of-order architectures

Step A – Pattern Identification



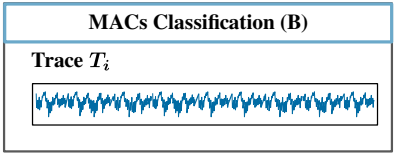
Pattern Identification

- Given one (or more) trace(s)...
- Identify a pattern for:
  - Important MACs (IMAC);
  - Non-important *executed* MACs (NIMAC);
  - Non-important *skipped* MACs
- Manual or automated search

Multiple patterns

- **Caveat:** one MAC may have multiple patterns
  - Multiple instructions issue
  - Out-of-order architectures
- We assume to have 1 pattern per MAC (scalar architecture)

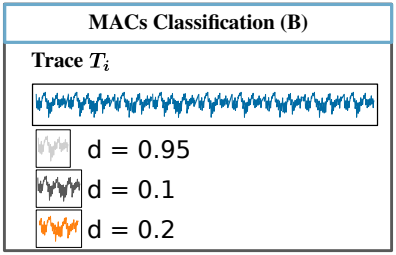
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

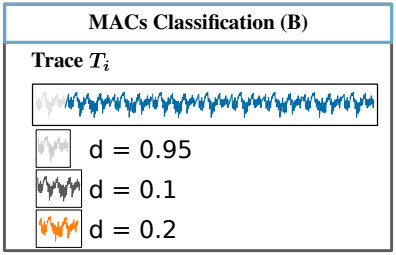
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

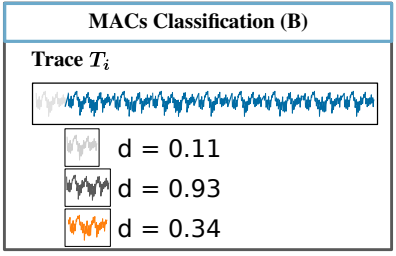
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

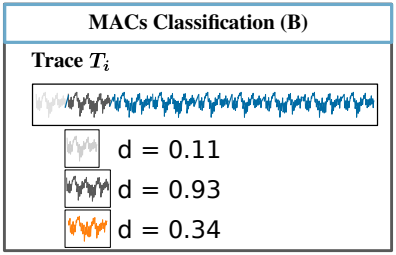
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

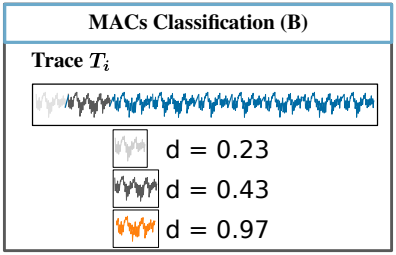
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

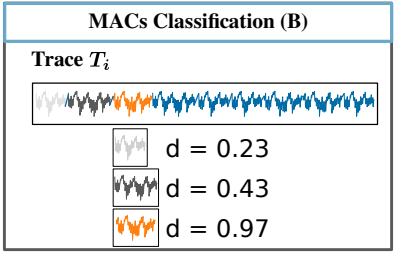
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

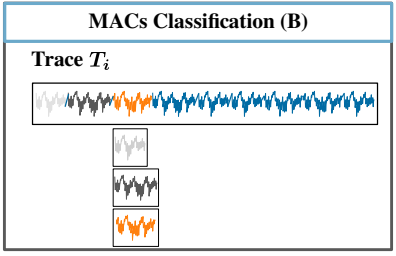
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

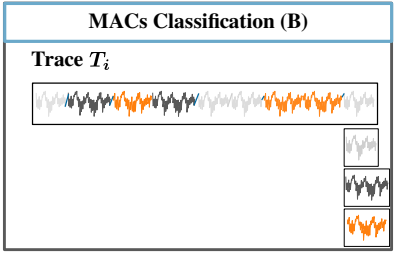
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

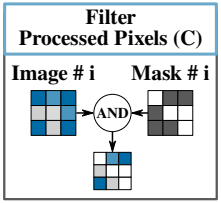
Step B – MACs Classification



Sliding-Window Pattern Matching

1. Define an acceptance threshold t;
2. Sliding each pattern p on each trace sample s;
3. For each pattern p, compute a distance score $d_p = d(p, T_i)$;
4. Assign to sample s pattern p if and only if:
   - $\forall p' \neq p : d_p > d_{p'}$
   - If $d(p, T_i) > t$, assign pattern to sample s;
5. Repeat from 2;

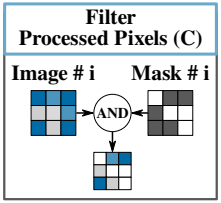
Step C – Filter Processed Pixels



Filter Processed Pixels

- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

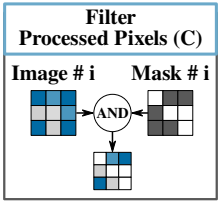
Step C – Filter Processed Pixels



Filter Processed Pixels

- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

Step C – Filter Processed Pixels

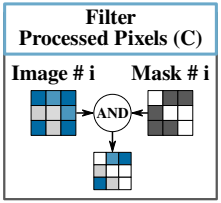


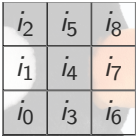
i_2	i_5	i_8
i_1	i_4	i_7
i_0	i_3	i_6

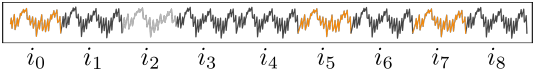
Filter Processed Pixels

- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

Step C – Filter Processed Pixels



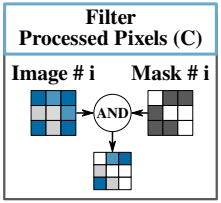


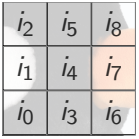


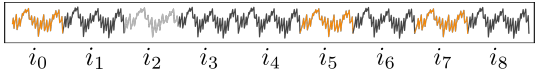
Filter Processed Pixels

- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

Step C – Filter Processed Pixels





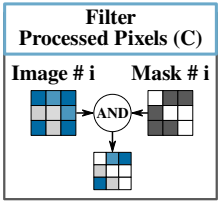


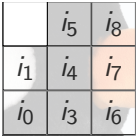
Filter Processed Pixels

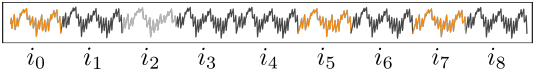
- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

To recover w_5:

Step C – Filter Processed Pixels





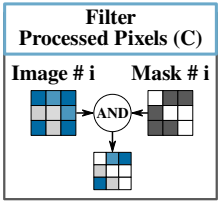


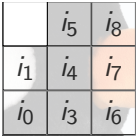
Filter Processed Pixels

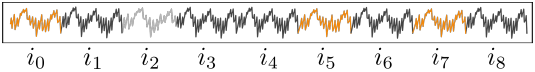
- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

To recover w_5:

Step C – Filter Processed Pixels







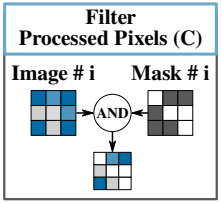
Filter Processed Pixels

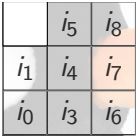
- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

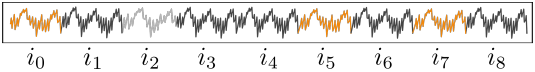
To recover w_5:

$$w_0 \cdot i_0 + i_1 \cdot w_1 + i_2 \cdot w_2 + i_3 \cdot w_3 + i_4 \cdot w_4 + i_5 \cdot w_5$$

Step C – Filter Processed Pixels







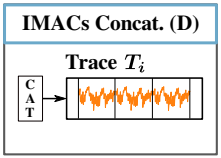
Filter Processed Pixels

- Remove skipped pixels/MACs
- Allow for correct hypotheses computation

To recover w_5:

$$w_0 \cdot i_0 + i_1 \cdot w_1 + \cancel{i_2 \cdot w_2} + i_3 \cdot w_3 + i_4 \cdot w_4 + i_5 \cdot w_5$$

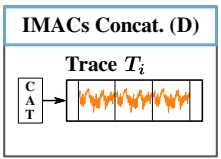
Step D – IMACs Concatenation

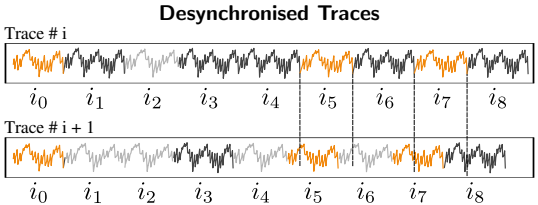


IMACs Concatenation

- Our goal: recover the Important MACs

Step D – IMACs Concatenation

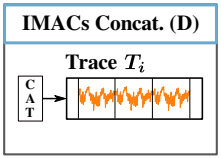




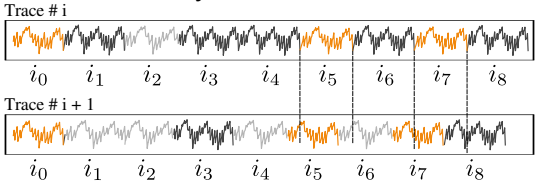
IMACs Concatenation

- Our goal: recover the Important MACs

Step D – IMACs Concatenation



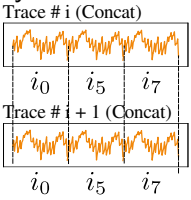
Desynchronised Traces



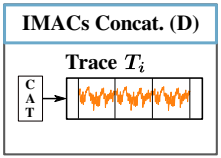
IMACs Concatenation

- Our goal: recover the Important MACs
- Preserve only the related patterns

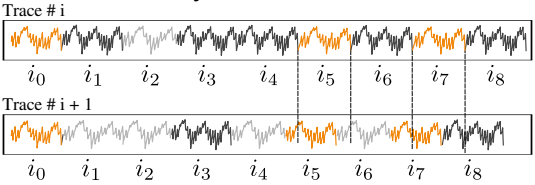
Synchronised Traces



Step D – IMACs Concatenation



Desynchronised Traces



IMACs Concatenation

- Our goal: recover the Important MACs
- Preserve only the related patterns
- By-product: traces vertically aligned

Synchronised Traces

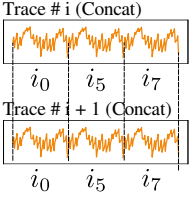


Table of Contents

Experimental Setup

Deep Neural Network model

- Multi-Layer Preceptron
- 5 neurons in the first layer
- 32 pixels per neuron

Side-channel Setup

- ARM Cortex-M4 (Clk = 7.37MHz)
- Chipwhisperer Lite (Sampling rate = x4 Clk)

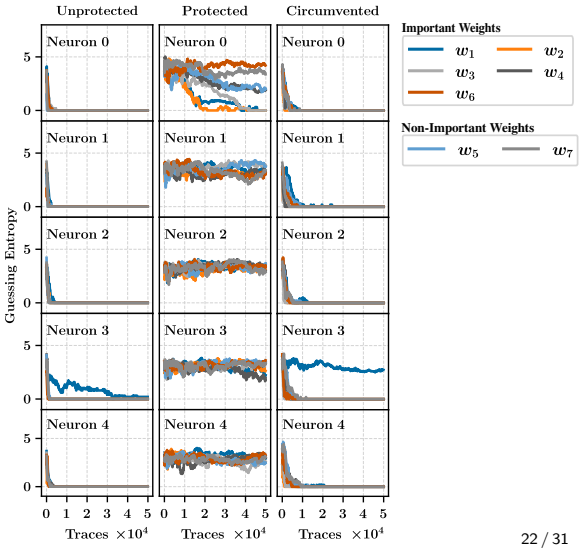
Experimental Procedure and Results

Experimental Procedure

1. Randomly and fix Important Pixels/MACs
2. Generate 5 sets of 50k random images
3. Recover 5 sets of 50k side-channel traces
4. Attack each set with Correlation Power Analysis
   - For each set, sort w_i's values in descending order
5. Compute the *Guessing Entropy (GE)* of w_i:
   - Average position of the true weight value
   - If $GE(w_i) = 0 \rightarrow$ true weight scores the best

Results

- Recovered all the important weights



Summary Pt. 1

MACPruning:
Exponentially increases number of traces for successful weight extraction

Summary Pt. 1

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Summary Pt. 1

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Summary Pt. 1

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Summary Pt. 1

Question:
What if we remove the control-flow dependency?

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPRUNING implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPRUNING implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel i

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPruning implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPruning implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*
  - Side-channel secure by hypothesis

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPRUNING implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

1 $acc \leftarrow 0$;
2 **for** i **from** 0 **to** `length`$(inputs)$ **do**
3 $cond \leftarrow$ `toSkip`$(IaPAM[i], randWord, i)$;
4 $fn \leftarrow cond{*}addr($`exec`$())+(1-cond){*}addr($`skip`$())$;
5 $acc \leftarrow fn(inputs[i], weights[i], acc)$;
6 **end**
7 **return** acc;

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*
  - Side-channel secure by hypothesis
- `exec()`: execute the MAC and return updated `acc`

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPruning implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*
  - Side-channel secure by hypothesis
- `exec()`: execute the MAC and return updated `acc`
  - Process both important and non-important pixels

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPruning implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*
  - Side-channel secure by hypothesis
- `exec()`: execute the MAC and return updated `acc`
  - Process both important and non-important pixels
- `skip()`: simply return `acc`

MACPruning: Fundamentally Broken?

Algorithm 4: Pseudo-code of a generic inference procedure enhanced with a control-flow-free MACPruning implementation.

Input : $inputs$: $(w \times h)$-array of pixels
$weights$: $(w \times h)$-array of weights
$IaPAM$: $(w \times h)$-array of bits
$randWord$: $(w \times h)$-array of bits
Output: acc: accumulator

```
1 acc ← 0;
2 for i from 0 to length(inputs) do
3 |   cond ← toSkip(IaPAM[i], randWord, i);
4 |   fn ← cond*addr(exec())+(1−cond)*addr(skip());
5 |   acc ← fn(inputs[i], weights[i], acc);
6 end
7 return acc;
```

A Branchless MACPruning

- `toSkip`: check whether to process pixel *i*
  - Store result in *cond*
  - Side-channel secure by hypothesis
- `exec()`: execute the MAC and return updated `acc`
  - Process both important and non-important pixels
- `skip()`: simply return `acc`
- `exec()` **and** `skip` **different by design**

MACPruning: Fundamentally Broken?

Another Side-Channel Leakage

- `exec()` and `skip` different by design

MACPruning: Fundamentally Broken?

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour

MACPruning: Fundamentally Broken?

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel

MACPruning: Fundamentally Broken?

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels

MACPruning: Fundamentally Broken?

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$

$$\mathrm{IaPAM} = \{1, 1, 0, 1, 0, 1\}.$$

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_4 = \{S, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$

$$\mathrm{IaPAM} = \{1, 1, 0, 1, 0, 1\}.$$

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_4 = \{S, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$

$$\mathrm{IaPAM} = \{0, 1, 0, 1, 0, 1\}.$$

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_4 = \{S, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$

$$\mathrm{IaPAM} = \{0, 1, 0, 1, 0, 1\}.$$

- We identify important weights,

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

MACPruning: Fundamentally Broken?

$$\mathrm{Seq}_1 = \{\boldsymbol{E}, \boldsymbol{E}, E, S, S, \boldsymbol{E}\},$$
$$\mathrm{Seq}_2 = \{\boldsymbol{E}, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_3 = \{\boldsymbol{E}, \boldsymbol{E}, S, E, E, \boldsymbol{E}\},$$
$$\mathrm{Seq}_4 = \{S, \boldsymbol{E}, S, S, E, \boldsymbol{E}\},$$

$$\mathrm{IaPAM} = \{0, 1, 0, 1, 0, 1\}.$$

- We identify important weights,
- Confidence bound on the number of traces.

Another Side-Channel Leakage

- `exec()` and `skip` different by design
  - Different side-channel behaviour
- We know when we process a pixel
  - `exec()`: process (non-)important pixels
  - `skip()`: skip (non-)important pixels
- Seq_i: classification of trace i
  - Each item is a pixel
  - E: (non-)important pixel executed
  - S: non-important pixel skipped

Table of Contents

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Branchless MACPruning:
Remove CFD-induced side-channel leakage

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Branchless MACPruning:
Remove CFD-induced side-channel leakage

Still, we are pruning:
Control-flow depends on the pixel's importance

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Branchless MACPruning:
Remove CFD-induced side-channel leakage

Still, we are pruning:
Control-flow depends on the pixel's importance

Pruning itself is leaking:
We can still rely on our preprocessing methodology

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Branchless MACPruning:
Remove CFD-induced side-channel leakage

Still, we are pruning:
Control-flow depends on the pixel's importance

Pruning itself is leaking:
We can still rely on our preprocessing methodology

Results:
Pruning cannot be used as it is

Summary Pt. 2

MACPruning:
Exponentially increases number of traces for successful weight extraction

MACPruning Implementation:
Control-flow dependency (CFD) on pixel's importance

Our Work:
Preprocessing methodology to exploit the CFD-induced side-channel leakage

Results:
Extracted important weights with comparable number of traces of an unprotected MLP

Branchless MACPruning:
Remove CFD-induced side-channel leakage

Still, we are pruning:
Control-flow depends on the pixel's importance

Pruning itself is leaking:
We can still rely on our preprocessing methodology

Results:
Pruning cannot be used as it is

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Unless we solve the pruning-induced leakage

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Unless we solve the pruning-induced leakage

Can Approximate Computing Underpin a Countermeasure?

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Unless we solve the pruning-induced leakage

Can Approximate Computing Underpin a Countermeasure?

Maybe

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Unless we solve the pruning-induced leakage

Can Approximate Computing Underpin a Countermeasure?

Maybe

Approximate Computing is about *accuracy–performance* trade-off

Everything is Lost?

Cannot We Use (MAC)Pruning at all?

No

Unless we solve the pruning-induced leakage

Can Approximate Computing Underpin a Countermeasure?

Maybe

Approximate Computing is about *accuracy–performance* trade-off

Secure Approximate Computing might trade off accuracy for *security*

Thank You!

Questions?

Bibliography I

[Din+25] Ruyi Ding et al. "MACPruning: Dynamic Operation Pruning to Mitigate Side-Channel DNN Model Extraction". In: *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). May 2025, pp. 12–22. DOI: 10.1109/HOST64725.2025.11050072. URL: https://ieeexplore.ieee.org/document/11050072 (visited on 10/05/2025).

Bibliography II

[Jap+25] Aditya Japa et al. "Security of Approximate Neural Networks against Power Side-channel Attacks". In: *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 2025 62nd ACM/IEEE Design Automation Conference (DAC). June 2025, pp. 1–7. DOI: 10.1109/DAC63849.2025.11133333. URL: https://ieeexplore.ieee.org/document/11133333/ (visited on 09/29/2025).

[Leo+25] Vasileios Leon et al. "Approximate Computing Survey, Part I: Terminology and Software & Hardware Approximation Techniques". In: *ACM Comput. Surv.* 57.7 (Mar. 5, 2025), 185:1–185:36. ISSN: 0360-0300. DOI: 10.1145/3716845. URL: https://dl.acm.org/doi/10.1145/3716845 (visited on 04/22/2025).

Bibliography III

[Zha+26] Lu Zhang et al. "ApproPower: Characterizing the Power Side-Channel Features of Approximate Multipliers with Symbolic Path Analysis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2026), pp. 1–1. ISSN: 1937-4151. DOI: 10.1109/TCAD.2026.3656474. URL: https://ieeexplore.ieee.org/abstract/document/11359193 (visited on 01/26/2026).

Microarchitectural Leakage

Unsuccessful Attacks?

- Not possible to recover the weight w_7
- Why?
- Solution: consider extended traces

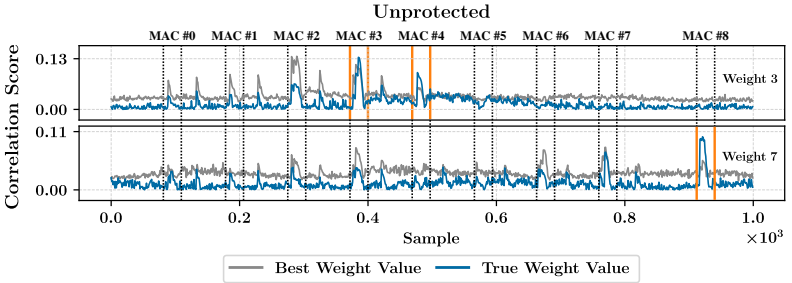


Figure: Microarchitectural Leakage.

Recovery of Non-Important Weights

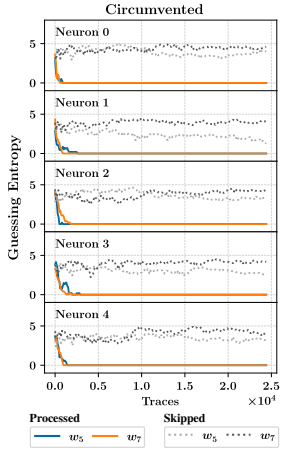


Figure: GE Filtered Traces.

Information carrying

- Partition traces in two sets:
  - *Processed*: non-important weight w_i processed
  - *Skipped*: non-important weight w_i skipped
- Compute GE on both sets
- Filtered traces carry information on non-important weights