Singularization: An Efficient Alternative to AES for Safeguarding Model Weights

Robert Poenaru

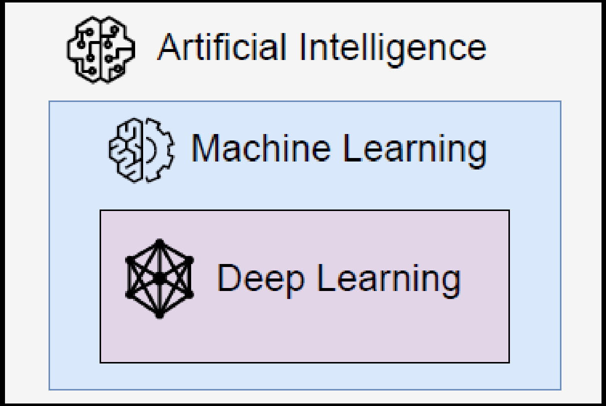
Orange Services, Bucharest (RO)

orange™

Outline

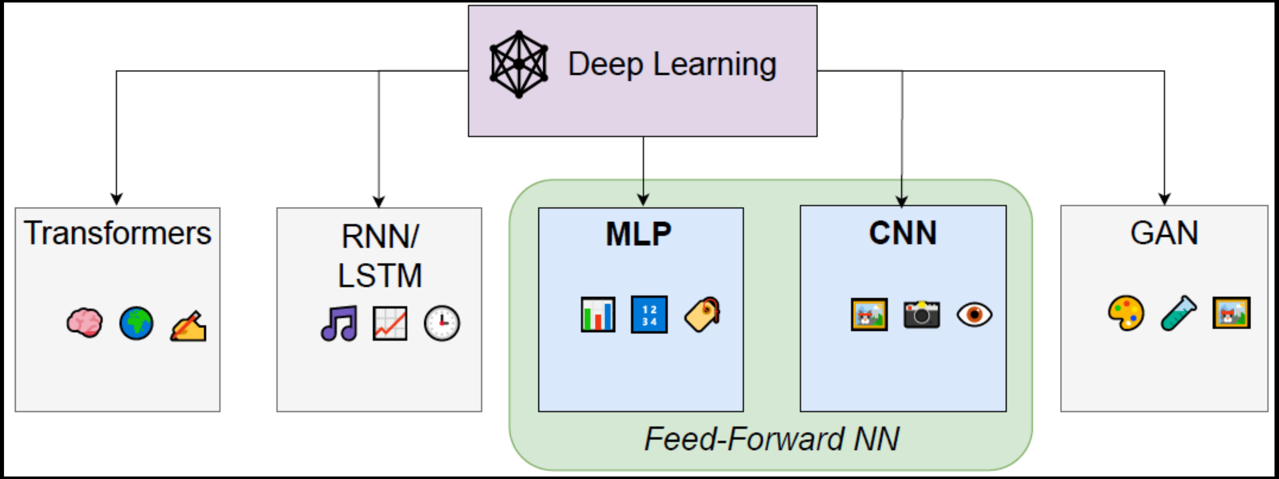
Artificial Intelligence (AI)

- AI is a crucial tool in online systems
- Machine Learning (ML) enables AI in systems
- Deep Learning (DL) is a subset of ML, used to solve specific tasks
  - Predictive modelling
  - Computer Vision
  - Voice recognition
  - Text predictions (NLP)



Artificial Neural Networks (ANN)

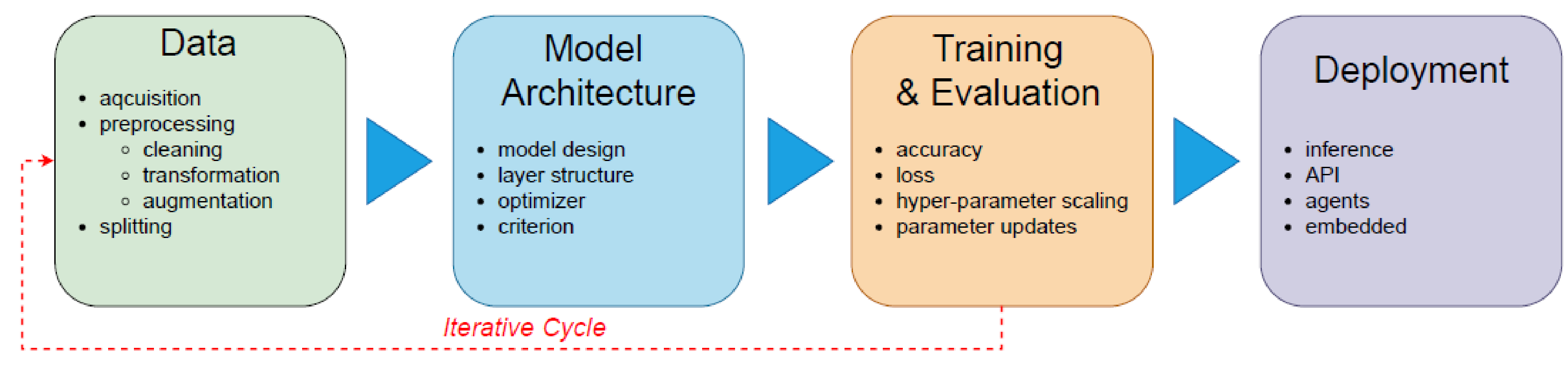
- ANN are the building blocks of Deep Learning.



- Trained via backpropagation and optimized with gradient descent.

Deep Learning Workflow

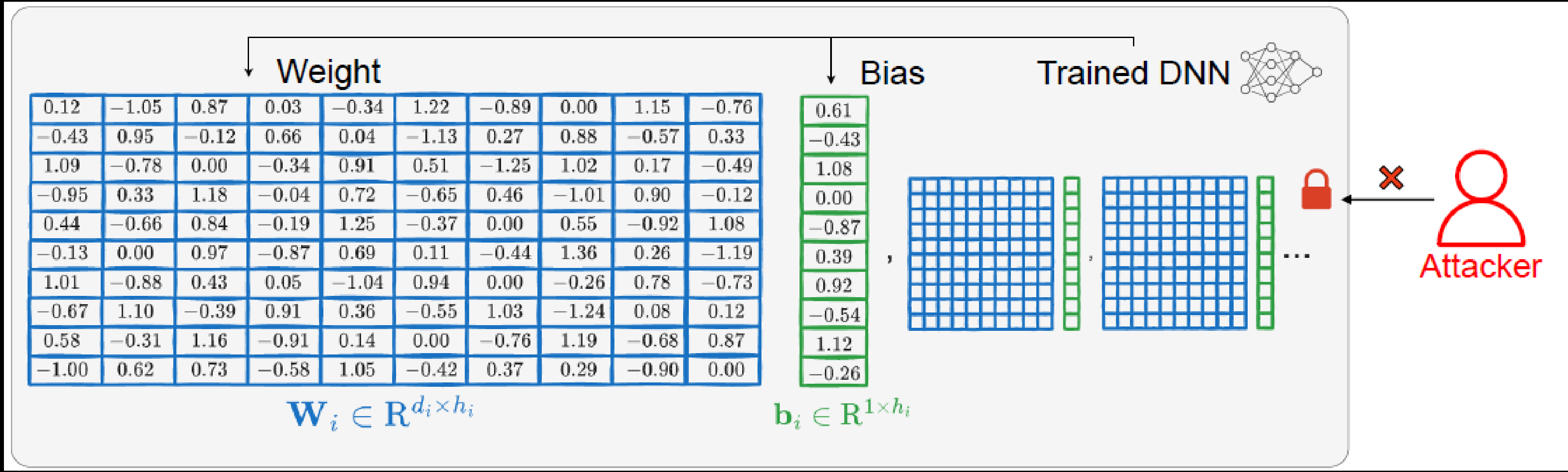
- Development of deep neural networks is an iterative cycle of design, training, and optimization.



- The iterative cycle is non-trivial: large amount of proprietary data, patented technology, computing energy, human resources.

Deep Learning Workflow II

- Final product in the DNN lifecycle; a collection of real-valued parameters: weights and biases.
- They constitute a form of intellectual property with strategic and commercial value → safeguarding these parameters is essential



DNN Protection by Obfuscation

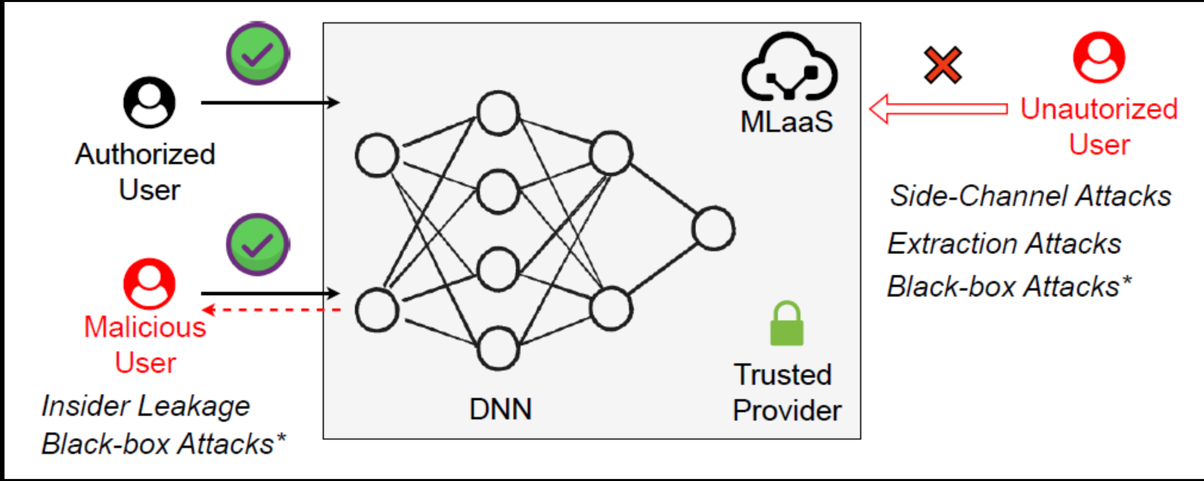
In an ideal scenario, a DNN model should be protected both in terms of architectural design and parameters.

	Protecting Parameters	Protecting Architecture
Goal	Prevent leakage or misuse of trained weights.	Hide the model design from attackers or competitors.
Why	Weights represent costly training (data, compute, expertise).	Design may reveal task-specific innovations or proprietary knowledge.
SOTA	DNN watermarking [1], Fully Homomorphic Encryption [2], Differential Privacy [3],	NN Obfuscation [4], Code Obfuscation [5]

[1] D. Rouhani et al. (2019): DeepMarks, DeepSigns
[2] A. Stoian et al. (2023): ConcreteML - Deep Neural Networks for FHE
[3] Wang et al. (2023): Practical DP for Deep Learning
[4] Gong et al. (2021): ModelObfuscator
[5] Zhang et al. (2023): NeurObfuscator

Threat Model

SCENARIO: A business deploys a DNN to the cloud (MLaaS), where authorized users can use for inference.



THREAT: Malicious and unauthorized users can perform attacks to extract the model parameters (parameter piracy).

Proposed Scenario

- Practical Use-case
  - Prevent parameter stealing from a trained DNN through an obfuscation method that minimizes the attack surface.

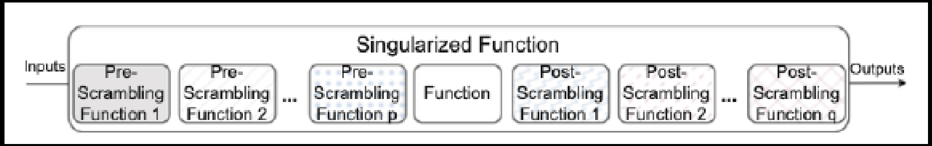
- Key characteristics
  - The proposed method aims at the following goals:
    - lightweight - the solution should not significant introduce overhead
    - straight-forward & self-consistent - simplistic mechanism
    - plug-and-play - no need for 3rd party libraries or frameworks
    - backwards-compatible - can be applied to pre-existing MLaaS
    - maintainability, scalability

Singularization - A Novel MTD Approach

Position Paper: Strengthening Applets on Legacy SIM Cards with Singularization, a New Moving Target Defense Strategy

Chrystel Gaber[1,2(✉)], Gilles Macariot-Rat[1,2], Simona David[1,2], Jean-Philippe Wary[1,2], and Alain Cuaboz[3]

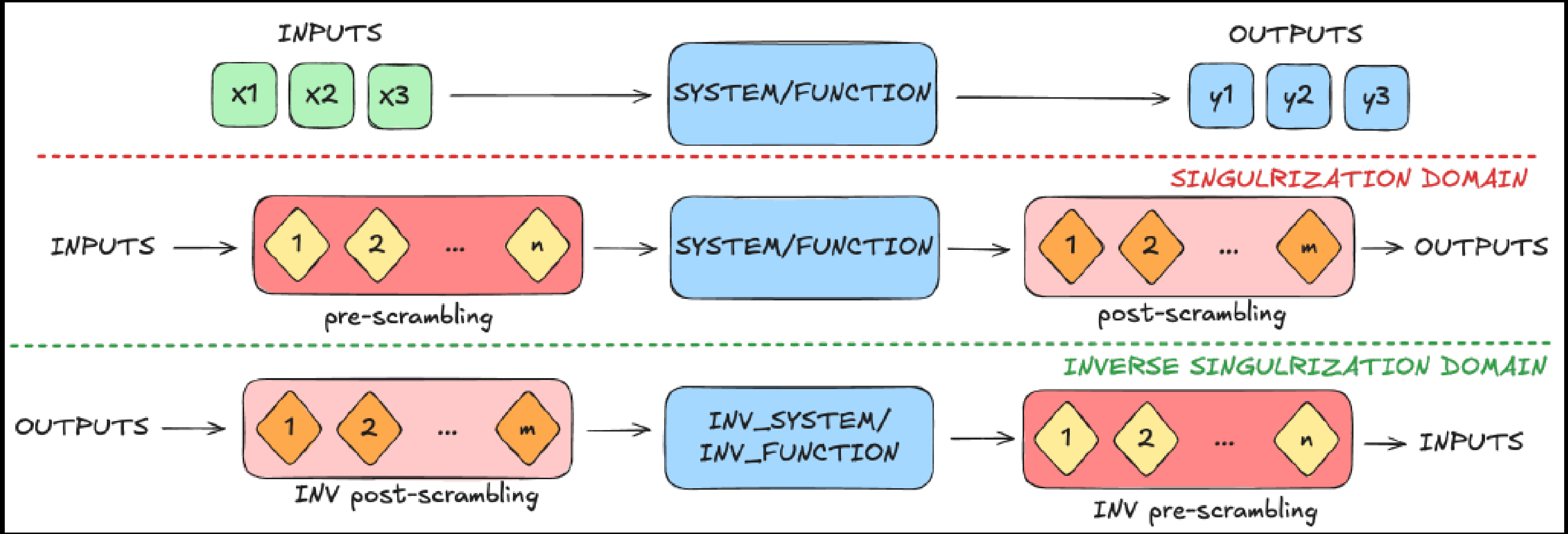
[1] Orange Innovation, Paris, France
chrystel.gaber@orange.com
[2] Orange Innovation, Bucharest, Romania
[3] Viaccess, Paris, France



- C. Gaber et. al. (2024) proposed a method for enhancing the security/robustness of an existing system, without needing to perform a full replacement of the underlying system.
- Singularization relies on encoding the inputs and outputs of a security function (e.g., cryptographic methods, code obfuscation).
- The scale and granularity of the encodings are much diverse than existing MTD methods.

Singularization as an Obfuscation Method

- Singularization does not change the system/function itself, it rather scrambles its input and output.
- Each function instance employs unique pre- and post-scrambling procedures at the input/output level.



Singularization in Neural Networks

The concept of unique scrambling functions [6] can be extended to DNN parameters (real-valued matrices) through an obfuscation via permutation mechanism.

Permuting Matrices

- In a recent work [7], it was shown that a DNN can have several types of weight permutation procedures that can be applied to its layers.
- The only mechanism of interest here: line-wise + column-wise permutations.
- Empirical results showed that weight permutation leads to random guessing for a DNN.

[6] C. Gaber et. al. (2024): Singularization: a New Moving Target Defense Strategy

[7] R. Poenaru & M. Plesa (2025): Presentation at ICMLC-2025

Singularization Formalism

Let $W \in \mathbb{R}^{5x5}$ be a trained weight:

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} \\ W_{31} & W_{32} & W_{33} & W_{34} & W_{35} \\ W_{41} & W_{42} & W_{43} & W_{44} & W_{45} \\ W_{51} & W_{52} & W_{53} & W_{54} & W_{55} \end{bmatrix}$$

and two operators P_{line} (**line-wise permutations**) and Pc_{ol} (**column-wise permutations**), matrices $\in \mathbb{R}^{5x5}$. Then, **singularization** will be:

$$W_{line} = P_{line}\, W$$
$$W_{sing} = W_{line}\, Pc_{ol} = P_{line}\, W\, Pc_{ol}$$

W_{sing} is defined as the singularized weight.

Permutation Example: Line-wise and Column-wise

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix}$$

$$P_{\mathsf{line}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad P_{\mathsf{col}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{W}_{\mathsf{sing}} = P_{\mathsf{line}} \mathbf{W} P_{\mathsf{col}} = \begin{bmatrix} w_{33} & w_{31} & w_{35} & w_{32} & w_{34} \\ w_{13} & w_{11} & w_{15} & w_{12} & w_{14} \\ w_{43} & w_{41} & w_{45} & w_{42} & w_{44} \\ w_{53} & w_{51} & w_{55} & w_{52} & w_{54} \\ w_{23} & w_{21} & w_{25} & w_{22} & w_{24} \end{bmatrix}$$

Singularization Formalism II

Inverse Singularization Transformation

- The singularization procedure $\mathbf{W} \xrightarrow{\text{sing.}} \mathbf{W}_{\text{sing}}$ is **invertible**.
- The **de-singularization** process $\mathbf{W} \xleftarrow{\text{de-sing.}} \mathbf{W}_{\text{sing}}$ is valid for P_{line}^{-1} and P_{col}^{-1}.
- Permutation matrices are **orthogonal**: $P^{-1} = P^\top$, therefore:

$$\mathbf{W} = P_{\text{line}}^{-1}\,\mathbf{W}_{\text{sing}}\,P_{\text{col}}^{-1} = P_{\text{line}}^{\top}\,\mathbf{W}_{\text{sing}}\,P_{\text{col}}^{\top}$$

Singularization Keys

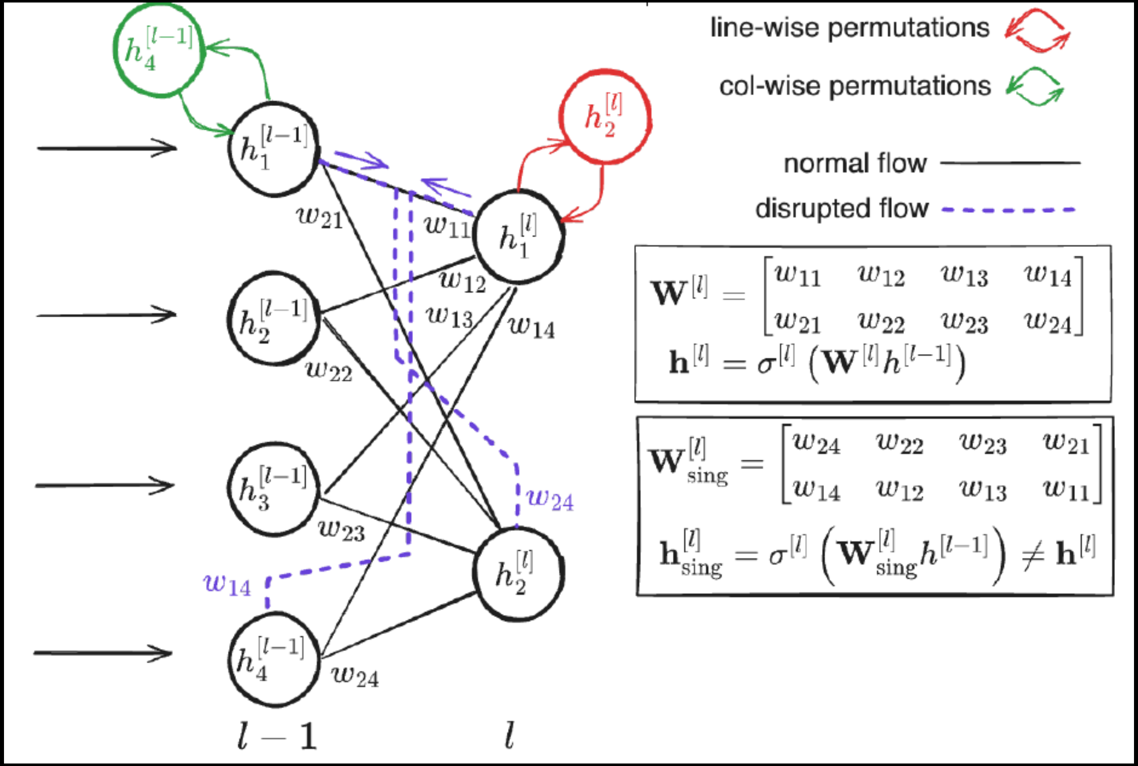
For a weight $\mathbf{W}$, its **singularization keys** are defined by the set:

$$\{P_{\text{line}}, P_{\text{line}}^{-1}, P_{\text{col}}, P_{\text{col}}^{-1}\}$$

allowing for both **singularization** and **de-singularization**.

Singularization at Inference

Since DNN weights will be permuted, a mismatch in the learned data flow will cause the model accuracy to drop.

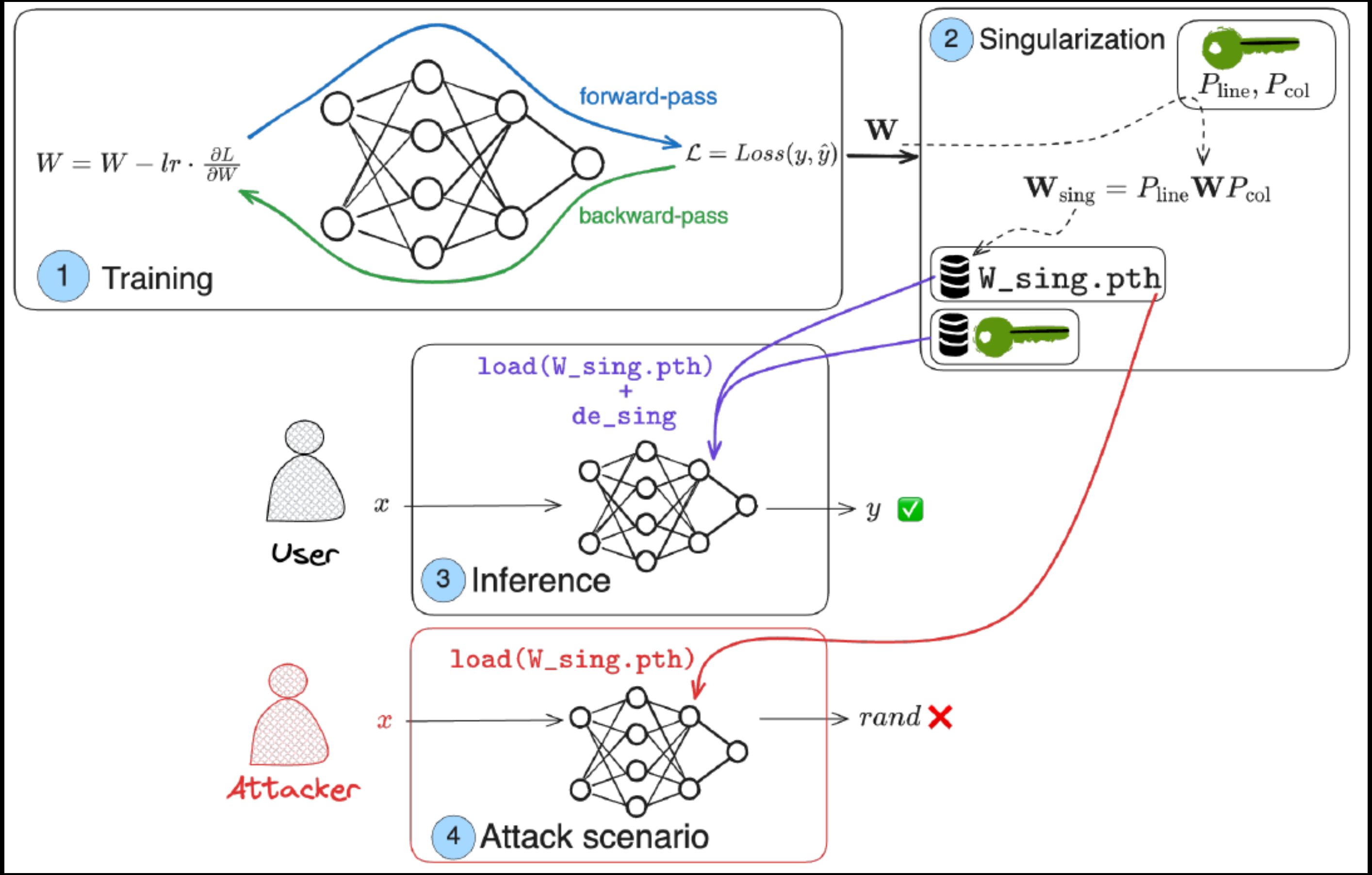


MLaaS with Singularization

Workflow for a DNN

- At training: optimize the weights with SGD for all layers and generate singularization keys after training is done.
- Save a model checkpoint on disk, but with the singularized weights instead of the ’plain’ ones.
- At inference: load singularized weights into memory and perform de-singularization during the forward-pass.
- Attack scenario: an unauthorized user does not have knowledge about singularization keys, loading only the singularized weights.

MLaaS with Singularization II



Results for MLP

Let L^3 Net be a deep neural network (DNN) defined as:

$$f(X) = f^{(3)} \circ f^{(2)} \circ f^{(1)}(X),$$

where each layer transformation $f^{(l)}$ is defined as:

$$f^{(1)}(H^{(0)}) = \mathrm{ReLU}(W^{(1)} H^{(0)}),$$
$$f^{(2)}(H^{(1)}) = \mathrm{ReLU}(W^{(2)} H^{(1)}),$$
$$f^{(3)}(H^{(2)}) = W^{(3)} H^{(2)}.$$

Results for MLP II

The model L3 Net was trained until a target accuracy, then the weights were singularized, and model was re-evaluated several times.

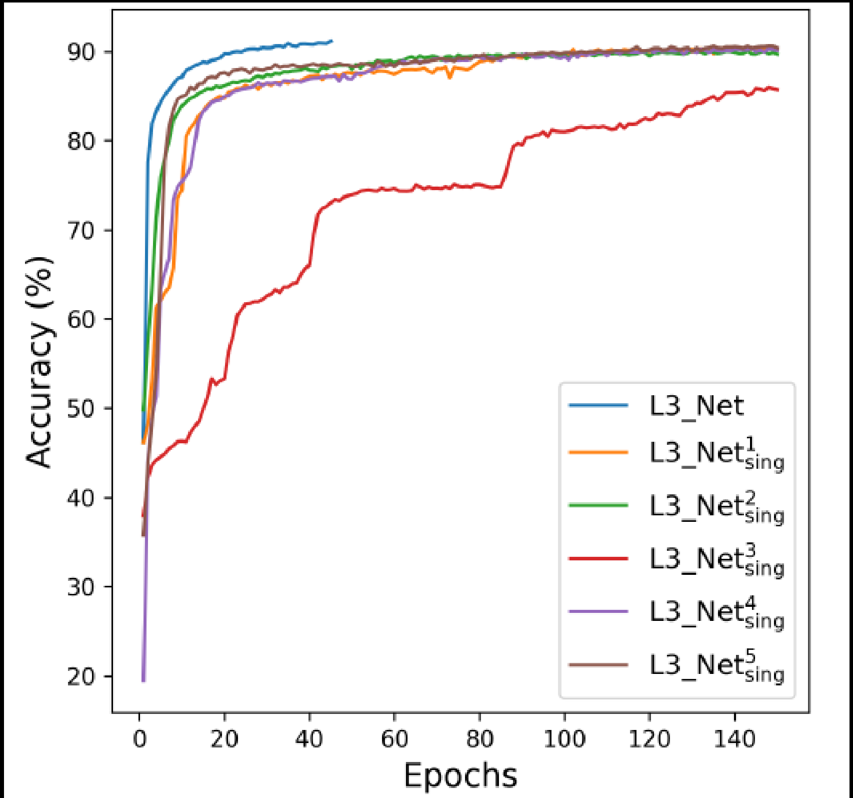
Model	Accuracy (%)	Loss (Cross-Entropy)
L3_Net trained	91.0 (target acc.)	0.321
$\text{L3\_Net}_{\text{sing}}$ (Test 1)	8.64	10.201
$\text{L3\_Net}_{\text{sing}}$ (Test 2)	10.40	8.852
$\text{L3\_Net}_{\text{sing}}$ (Test 3)	11.81	8.620
$\text{L3\_Net}_{\text{sing}}$ (Test 4)	13.05	12.021
$\text{L3\_Net}_{\text{sing}}$ (Test 5)	11.24	4.757

Untrained L3 Net acc: 10.14%

Singularization and Retraining (MLP)

Testing the Robusntess

- **Performance** of L3_Net$_{\text{sing}}$ is similar to *random guessing*.
- If the singularized **weights are extracted**, an **attacker** might try to **retrain** the model.
- **Retraining** after **permutations** shows the challenge of **recovering the original model**.

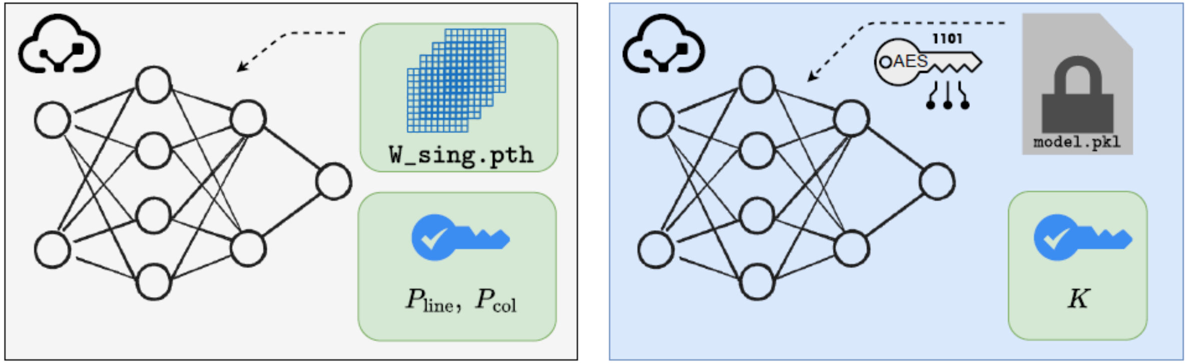


In a fine-tuning attack scenario, the attacker's efforts exceeds that of training from scratch (In terms of the number of epochs, under similar training configuration.)

Singularization vs. Encryption (MLP)

Weight permutation is benchmarked against standard AES encryption.

Property	Singularization	AES Encryption
Model Storage	Weights in plaintext but permuted	Weights are fully encrypted ciphertext
Key Requirement	Requires storing singularization (permutation) keys	Requires encryption/decryption keys
Key space (brute-force)	$\prod_{l=1}^{k}(d_l! \times d_{l-1}!)$, **potentially larger than AES**	2^{128} to 2^{256}, but **cryptanalysis-resistant** under modern assumptions
Execution	Can load weights directly and inference works via de-singularization	Cannot use model until decrypted
Security	Reversible if permutation is known	Secure under AES assumptions



Singularization vs. Encryption II (MLP)

For the implementation of AES encryption [9], the cryptography [10] library was used.

Experiments	Singularization (ms)	AES (ms)	Performance Boost
Iteration 1	0.43	2.71	**6.3x**
Iteration 2	0.36	2.48	**6.9x**
Iteration 3	0.37	2.38	**6.4x**
Iteration 4	0.34	2.58	**7.6x**
Iteration 5	0.46	2.22	**4.8x**

On average, singularization is 6.4x faster than AES.

[9] AES-256 in CTR mode with a 256-bit key and 128-bit nonce.
[10] https://pypi.org/project/cryptography/

Results for CNN

Let Conv Net be a 4-convolutional layer architecture:

$$f(X) = f^{(6)} \circ f^{(5)} \circ f^{(4)} \circ f^{(3)} \circ f^{(2)} \circ f^{(1)}(X),$$

where each layer transformation $f^{(l)}$ is defined as:

$$
\begin{aligned}
f^{(1)}(H^{(0)}) &= \mathrm{ReLU}(W^{(1)} * H^{(0)}),\\
f^{(2)}(H^{(1)}) &= \mathrm{MaxPool}(\mathrm{ReLU}(W^{(2)} * H^{(1)})),\\
f^{(3)}(H^{(2)}) &= \mathrm{MaxPool}(\mathrm{ReLU}(W^{(3)} * H^{(2)})),\\
f^{(4)}(H^{(3)}) &= \mathrm{MaxPool}(\mathrm{ReLU}(W^{(4)} * H^{(3)})),\\
f^{(5)}(H^{(4)}) &= \mathrm{ReLU}(W^{(5)} H^{(4)}),\\
f^{(6)}(H^{(5)}) &= W^{(6)} H^{(5)}.
\end{aligned}
$$

Singularization vs. Encryption (CNN)

- Timing benchmark for Singularization and AES encryption on Conv_Net [11].

Experiment	Singularization (ms)	AES (ms)	Performance Boost
Iteration 1	0.58	11.30	**19.5x**
Iteration 2	0.54	13.00	**24.1x**
Iteration 3	0.70	16.60	**23.7x**
Iteration 4	0.72	14.10	**19.6x**
Iteration 5	1.47	22.10	**15.0x**

- On average, singularization is 21x faster than AES.

DNN complexity

The **benefit** of *singularization* over the standard encryption is the scalability with larger and more complex networks.

[11] AES-256 in CTR mode with a 256-bit key and 128-bit nonce.

Conclusions

- Singularization was introduced as an obfuscation strategy for the parameters of a DNN.
- Empirical evidence shows that the permutations introduce enough disruption, similar to a random DNN.
- Several attack scenarios are dealt with:
  - black-box attacks (limited)
  - extraction attacks (strong)
  - fine-tuning attacks (costly, exceeds full training from scratch)
- Singularization provides negligible overhead on the DNN workflow
- Faster as compared to standard encryption schemes (AES).

Thank you for your attention!

Questions?

Sincere gratitude to the conference organizers, partners, and AMTD Workshop organizers (Simona David, Mihail Plesa, and Florentin Vizireanu, Dan Stanescu).