AN12867

Security for Machine Learning Package

Rev. 1.0 — 16 June 2025 Application note

Document information

Information	**Content**
Keywords	AN12867, eIQ, machine learning
Abstract	With the wide-scale deployment of machine-learning models, both safety and security issues have become a significant threat. This document describes some of these issues and the countermeasures made available in eIQ to mitigate their impact. It is focused on adversarial attacks, model cloning, and model inversion.

1 Introduction

With the wide-scale deployment of machine-learning models, both safety and security issues have become a significant threat. This document describes some of these issues and the countermeasures made available in eIQ to mitigate their impact. It is focused on adversarial attacks, model cloning, and model inversion.

Users concerned about security must also utilize the i.MX security features available at the SoC level. Enabling such features can benefit the general system security. For further details, see the processor security reference manual document available in the SoC documentation page at www.nxp.com.

2 Adversarial examples

Adversarial attacks represent a security and safety issue in the practical large-scale deployment of machine learning (Biggio, et al.), (Szegedy, et al. 2013). These are "*inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence*" (Goodfellow, Shlens and Szegedy 2014). Hence, you can create specifically crafted inputs (video, images, or sound examples) which try to mislead the machine-learning model such that it misclassifies a road-sign (safety concern) or circumvents voice or face authentication (security concern).

For example, the following two images are classified differently by an Inception v3 network trained for the ImageNet dataset:

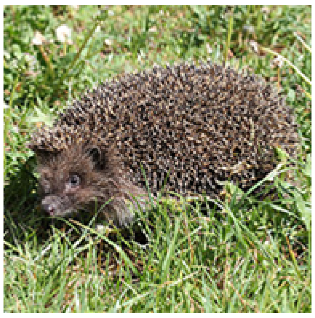


Figure 1. Top 3 classes: 90.5 % porcupine/hedgehog, 2.1 % marmot, 1.0 % beaver

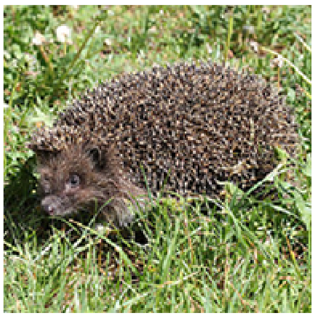


Figure 2. Top 3 Classes: 99.0 % banana, 0.1 % pineapple, 0.05 % porcupine/hedgehog

Source (public domain): https://commons.wikimedia.org/wiki/File:Erinaceus_roumanicus_2013_G5.jpg

Figure 2 is intentionally modified to be misclassified as a banana. Even though the changes are not visible to the naked eye, the neural net has an even higher confidence that the second image is a banana than that the original image was a hedgehog.

The eIQ security package includes model-hardening functionality which makes these types of attacks significantly more difficult. This functionality can harden a model against such carefully selected perturbations without modifying the trained machine learning model. The main premise is to transform the model's input in ways which ensure that the classification outcome on real inputs remains unchanged, while the adversarial perturbations of an attack no longer work. By computing multiple transformed inputs, evaluating the model on all of them, and combining the outputs using a majority vote, it becomes more difficult to mislead the target system.

Such transformations are application-specific. The eIQ security package provides code for models which have images as inputs. Specifically, the provided transformations are blurring, rotation, noise injection, and JPEG compression. A detailed description of the API is available in the Doxygen documentation in the */api-docs* subfolder. A complete example is in the *examples/ax_hardening.cpp* file.

Classifying Figure 2 (the adversarial example) with a model that performs both input rotation and noise injection as the transformations and then taking the most frequent result (including the original second image output) correctly identifies this image as a hedgehog. The rotated version is correctly identified with 93 % confidence, which is even higher than the unmodified image.

Note that it is necessary to carefully select the right parameters for these transformations. The radius of the blurring depends on the size of the images and the objects in the image. The angle of rotation depends on the rotation tolerance of the model. Even with this additional hardening, it is still possible to create adversarial examples. However, it requires more time to construct them and the search space must be increased when compared to a model which does not employ this technique.

The eIQ ml-security package includes an example of hardening against adversarial examples, applying two transformations to the inputs - rotation by 3 degrees and bilateral blurring with a window size of 5. The values were determined experimentally. Because rotated objects may have different semantic meaning, only a small rotation is used. For example, hedgehogs on their back might have a different meaning. The blurring is chosen such that the important features of the hedgehog are preserved, but small changes made by the adversary are smoothed over. The original image and the transformed images are classified by the network, followed by a voting of the most occurring class. This example uses the same hedgehog images as above - class 335 is a porcupine/hedgehog, class 955 is a banana.

The example output is shown below. Both images are classified as a porcupine/hedgehog after applying the techniques described in this section.

```
# ./ax_sample
Using hedgehog image
Top 3 for original image
1: Class 335, Confidence: 0.905243
2: Class 337, Confidence: 0.0213407
3: Class 338, Confidence: 0.0109267
Top 3 for rotated image
1: Class 335, Confidence: 0.947023
2: Class 337, Confidence: 0.0062071
3: Class 338, Confidence: 0.00346831
Top 3 for blurred (bilateral) image
1: Class 335, Confidence: 0.812905
2: Class 338, Confidence: 0.0487511
3: Class 337, Confidence: 0.043767
Classification by votes: 335
Using adversarial example
Top 3 for original image
1: Class 955, Confidence: 0.990323
2: Class 954, Confidence: 0.00107823
3: Class 335, Confidence: 0.000476602
Top 3 for rotated image
1: Class 335, Confidence: 0.935247
2: Class 337, Confidence: 0.0106328
```

```
3: Class 338, Confidence: 0.0096081
Top 3 for blurred (bilateral) image
1: Class 335, Confidence: 0.719883
2: Class 337, Confidence: 0.0393073
3: Class 338, Confidence: 0.0233211
Classification by votes: 335
```

3 Model cloning

Machine learning models are susceptible to model extraction using retraining attacks (Tramèr et al. 2016, Correia-Silva et al. 2018), where an adversary exploits the information gained by querying the model with selected inputs and uses that as training data for a counterfeit model. Model extraction attacks enable an adversary to copy the functional behavior of the model into a clone, which can then be used to undermine pay-per-query mechanisms or the competitive edge of the creator of the original model. In addition, the attack can also be used to create a copy which can be inspected to gather additional information for other attacks, like the adversarial examples attack.

The eIQ ml-security package hardens machine-learning models against model extraction by perturbing the confidence values in the model output. This reduces the leaked information about the model's functional behavior. Therefore, an adversary must perform an increased number of queries to the model to gain sufficient information about the model to reproduce it. Since a successful model extraction attack requires more queries, the attack becomes more invasive and hence increases the spent effort of the adversary. Additionally, the increased queries become easier to detect and to subsequently take effective measures upon.

We offer several strategies to perturb the confidence information produced by the machine-learning model. All strategies keep the top-1 accuracy unchanged by maintaining the number 1 rank intact (the output class with the highest confidence). The strategies add noise by adding or subtracting values from a user-defined interval to the confidence levels of the predictions such that model extraction attacks become as effective as when they are performed on only the top label, while some useful information about the remaining ranks is preserved for the legitimate user.

There are two functions offered that perturb the predicted confidence levels without affecting the rank of the class that has the highest confidence: *addNoise* and *addPseudoNoiseSin*. The former adds random noise to each of the confidence values. The latter adds noise according to a function chosen and experimentally validated: $\text{confidence}_{new} = \text{confidence}_{old} + 0.05 + 0.1 \times \sin(50 \times (\text{confidence}_{old} - 0.09))$. It is shown in Figure 3 and results in a considerable increase in the number of queries required for successful model extraction. Although this results in a reduced precision of the model's prediction confidence, this reduction is limited. If the addition of noise causes the first ranked class to change, the original top class is swapped back into the top position by the algorithm. Therefore, neither *addNoise* nor *addPseudoNoiseSin* affects the top-1 accuracy. The functions support normalization, which scales the new confidence levels such that they sum to 1.

Note: *The addNoise function actually adds random noise, thus giving a different output when run repeatedly with the same input, while addPseudoNoiseSin gives consistent output.*

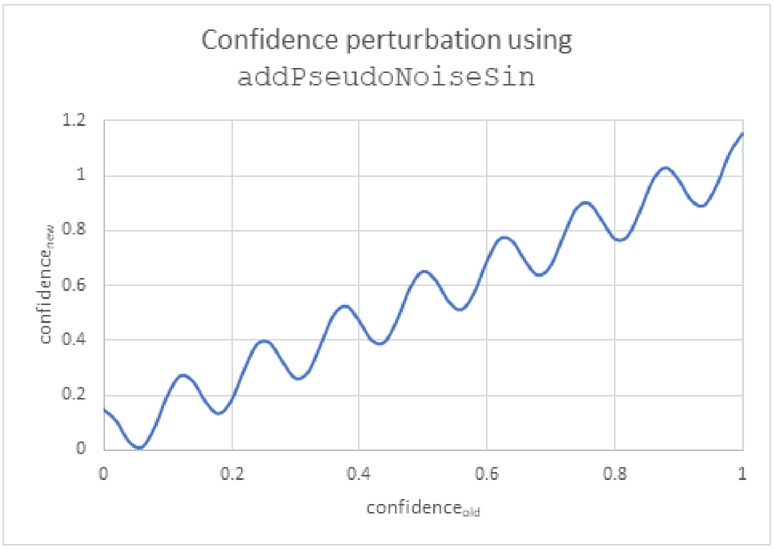


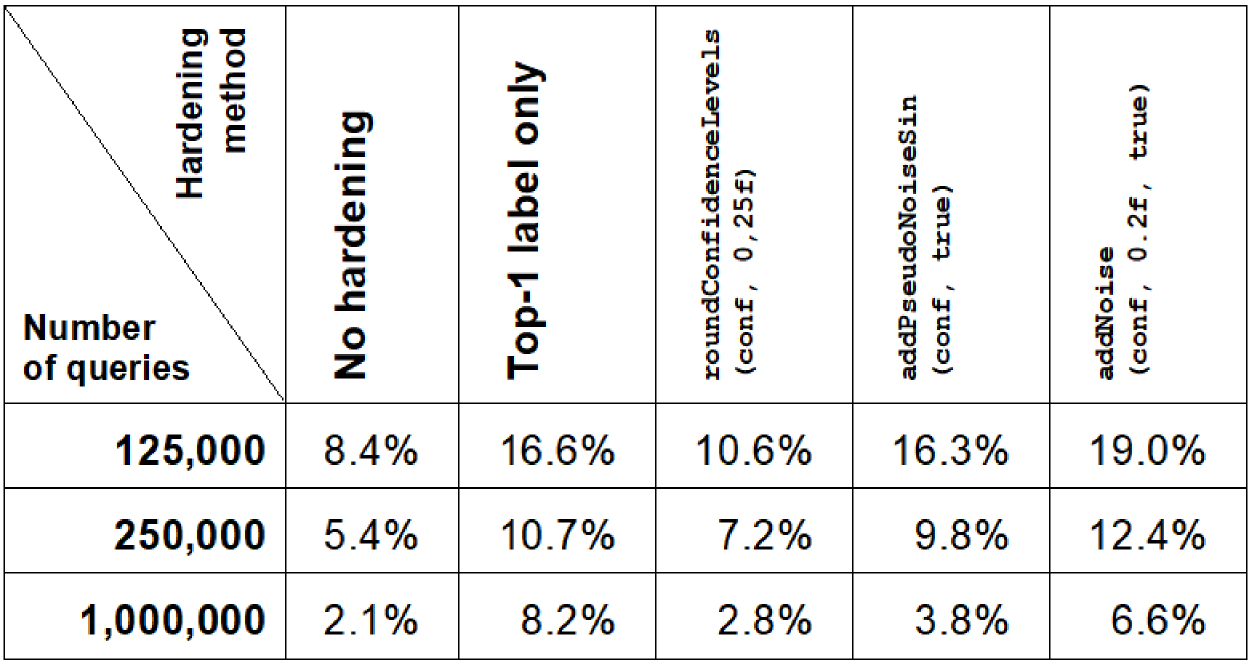
Figure 3. Remapping the original confidence levels to confidence levels with addPseudoNoiseSin

An additional function for perturbing the confidence levels is *roundConfidenceLevels*. It rounds the confidence levels to the nearest user-defined step. Contrary to the other two functions, this function may affect the top-1 accuracy, because the highest confidence level may be shared by multiple classes.

All three of these functions can be used as follows:

```
// calculate confidence levels with the neural network GoogLeNet
cv::Mat confidences = googleNet.forward();
bool applyNormalization = true;
// add noise to the confidence levels with a range of [-0.1, 0.1]
addNoise(confidences, 0.2f, applyNormalization);
// add pseudo noise to the confidence levels
addPseudoNoiseSin(confidences, applyNormalization);
// round the confidences to quarters precision
roundConfidenceLevels(confidences, 0.25f);
```

The effectiveness of each of the offered perturbation functions is shown in Figure 4. It shows the loss in top-1 accuracy of a model cloned using the 2018 attack by Correia-Silva et al. compared to the top-1 accuracy of the original model. The loss is given for attacks with 125,000, 250,000, and 1,000,000 queries to the original that the clone is trained on. “Labels only” is added as a reference and shows the loss in top-1 accuracy for the clone if the confidence levels are disregarded and only the label of the first ranked class is used for the model-extraction attack. This shows that the described countermeasure offers some protection against the attack without removing all information given to a legitimate user about the remaining confidence levels, as a model that would output only the top-1 label.



Hardening method / Number of queries	No hardening	Top-1 label only	`roundConfidenceLevels (conf, 0,25f)`	`addPseudoNoiseSin (conf, true)`	`addNoise (conf, 0.2f, true)`
125,000	8.4%	16.6%	10.6%	16.3%	19.0%
250,000	5.4%	10.7%	7.2%	9.8%	12.4%
1,000,000	2.1%	8.2%	2.8%	3.8%	6.6%

Figure 4. Loss of clone top-1 accuracy compared to original model top-1 accuracy

The choice of which hardening method to use (or to use no method) depends on the use case in which a model is deployed. A tradeoff between the protection against model extraction and the precision of the model is identified. If the precision of the confidence levels is not a requirement, then a model is best protected with the *addNoise* method with a large parameter for the range (for example, ≥0.2). If a use-case requires higher precision of the confidence levels, then the *addPseudoNoiseSin*, *addNoise* with a small range (for example, <0.2), or *roundConfidenceLevels* methods with a small range (for example, <0.2) can suffice. The parameter range of the noise for *addPseudoNoiseSin* is limited to interval [-0.05 , 0.15].

Note: *The described attacks require access to the input and output of the model.*

4 Model inversion

When machine learning is used for privacy-sensitive applications or when the data used for training contains privacy-sensitive information, this sensitive information can be learned by the model. This means that private information can be contained inside the trained model as part of the learned internal parameters. Therefore, model inversion attacks (Fredrikson, et al. 2015) may become an issue. In such an attack, an adversary uses information obtained by querying the model to extract privacy-sensitive information from the model. However, this requires the attacker to have access to the model through reverse engineering or similar techniques.

An example application for model-inversion attacks is an adversary attempting to extract the faces of individuals from a model used for face recognition. Another example application is an adversary trying to infer sensitive medical information about a person based on some easily available attributes and a machine-learning model trained to predict medical conditions or drug dosages (Fredrikson, et al. 2014).

To counter the threat of exposing privacy-sensitive information, the literature recommends limiting the accuracy of confidence values returned by the model. Generally, that means that similar countermeasures can be used to harden a model against model inversion attacks as the ones recommended to harden a model against the model extraction attack (as shown in Section 3), because both types of attacks are based on information leaked through a fine-grained confidence output.

The eIQ ml-security package allows you to harden pre-trained machine learning models against these attacks by rounding the confidence levels included in the output of the model to a given step size. Experimental results (from Fredrikson, et al. 2015) show that these types of attacks can be mitigated in this way, while maintaining the confidence levels accurate enough to be useful for the intended model's application. However, model inversion is not the focus of as much research and as many academic publications as the other threats detailed in this chapter. Therefore, it is possible that stronger attacks (using more queries or combined model extraction-inversion attacks) can allow an attacker to extract confidential information even from a hardened model.

The following code example shows how the rounding function can be applied to the output of a model:

```
// calculate confidence levels with the neural network
cv::Mat confidences = googleNet.forward();
// round the confidence levels
roundConfidenceLevels(confidences, 0.05f);
```

The countermeasure of adding small perturbations to the confidence output (as recommended to harden models against model extraction in Section 3) can also be used to harden models against model inversion.

The following example code shows how these output transformations can be applied to the output of a model:

```
// calculate confidence levels with the neural network
cv::Mat confidences = googleNet.forward();
bool applyNormalization = true;
// add noise to the confidence levels with a range of [-0.1, 0.1]
addNoise(confidences, 0.2f, applyNormalization);
// add pseudo noise to the confidence levels
addPseudoNoiseSin(confidences, applyNormalization);
```

For further details on these countermeasures, see Section 3.

5 eIQ ml-security library usage

The library can be included in an i.MX board image by adding `IMAGE_INSTALL_append = " ml-security-staticdev"` to *conf/local.conf*, assuming that the appropriate layers and recipes are available, which should hold for recent versions of the general BSPs. Detailed usage of the library is described in more detail in the API documentation included with the library. In that case, it should be available for static linking from both the toolchain and the final image. It is unlikely that the static library in the image is used, but it may be useful for quick testing and it takes up less than 3 MB of space.

On an image that includes the eIQ ml-security package, the library should be located in */usr/lib/libml-security.a* and the include files in */usr/include/ml-security/*. The pre-built examples and their source should be located in */usr/share/ml-security/examples*.

Using the Yocto SDK Toolchain for a supporting image should allow rebuilding the examples as follows:

```
mkdir /home/user/examples-build
cd /home/user/examples-build
cmake $SDK_SYSROOT/usr/share/ml-security/examples
make
```

This results in a binary which works on a device running the accompanying image. Using the examples requires the *test_images* folder from the *examples* directory and the *test_model/inception_v3.pb* file. This model can be downloaded and converted using the *get_model.sh* script in the examples folder, but it requires Bash, Python, and Tensorflow and it is intended for use on a Ubuntu host.

6 References

- Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Pavel Laskov Nedim Srndic, Giorgio Giacinto, and Fabio Roli. 2013. "Evasion attacks against machine learning at test time." *Machine Learning and Knowledge Discovery in Databases – European Conference, ECML PKDD 2013.* Springer. 387-402.
- Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. 2014. "Explaining and harnessing adversarial examples." *International Conference on Learning Representations.*
- Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2013. "Intriguing properties of neural networks." *International Conference on Learning Representations.*
- Jacson Rodrigues Correia-Silva, Rodrigo F. Berriel, Claudine Badue, Alberto F. de Souza, and Thiago Oliveira-Santos. 2018. “Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data” *2018 International Joint Conference on Neural Networks (IJCNN),* Rio de Janeiro, 2018, pp. 1-8. DOI:https://doi.org/10.1109/IJCNN.2018.8489592
- Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. “Stealing Machine Learning Models via Prediction APIs.” *25th USENIX Secur. Symp. USENIX Secur. 16*, Austin, TX, USA, August 10-12, 2016. MI (2016). DOI:https://doi.org/10.1103/PhysRevC.94.034301
- Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. “Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures”. *CCS '15 Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Pages 1322-1333*, Denver, Colorado, USA, October 12-16, 2015. DOI: https://doi.org/10.1145/2810103.2813677
- Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page and Thomas Ristenpart. 2014. “Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing”. *Proceedings of the 23rd USENIX Security Symposium,Pages 17-32 ,* San Diego, CA, USA, August 20–22, 2014.

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

8 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN12867 v.1.0	16 June 2025	• Updated to the current NXP look and feel.
AN12867 v.0	May 2020	• Initial version.

Legal information

Definitions

Disclaimers

Trademarks

Contents

Date of release: 16 June 2025
Document identifier: AN12867