Sécurité de l'Intelligence Artificielle Embarquée : attaques Physiques contre l'Intégrité et la Confidentialité de Réseaux de Neurones

Pierre-Alain Moellic

N° d'ordre NNT : 2025UJMO0062

THÈSE de DOCTORAT
par VALIDATION des ACQUIS de L'EXPERIENCE
DE L'UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Membre de l'Université de Lyon

Ecole Doctorale N°488
SIS - Sciences Ingénierie Santé

Spécialité de doctorat : Informatique

Soutenue publiquement le 12 décembre 2025, par :

PIERRE-ALAIN MOELLIC

Sécurité de l'Intelligence Artificielle Embarquée : Attaques Physiques contre l'Intégrité et la Confidentialité de Réseaux de Neurones

---

Devant le jury composé de :

CHRISTELLE BAHIER-PORTE, Professeur des universités, Université Jean Monnet Saint-Etienne	Présidente
BRUNO ROBISSON, Docteur HDR, Commissariat à l'Energie Atomique et aux Energies Alternatives	Rapporteur
JEAN-CHRISTOPHE PREVOTET, Professeur des universités, INSA RENNES	Rapporteur
MOUNIA KHARBOUCHE-HARRARI, Docteure, STMICROELECTRONICS	Examinatrice
CEDRIC KILLIAN, Professeur des universités, Université Jean Monnet Saint-Etienne	Examinateur
LILIAN BOSSUET, Professeur des universités, Université Jean Monnet Saint-Etienne	Directeur de thèse

Remerciements

Je tiens à adresser mes sincères remerciements aux membres du jury pour avoir accepté d'évaluer ce travail de thèse et participé à la soutenance qui s'est tenue sur le Campus de l'Université Jean Monnet, à Saint-Etienne. D'une certaine manière, cette soutenance de doctorat en VAE était une triple première : pour moi évidemment, pour l'école doctorale et les membres du jury. En premier lieu, je suis extrêmement reconnaissant à M. Lilian Bossuet pour avoir très tôt accepté de m'accompagner, en tant que référent VAE et directeur de thèse, dans ce qui n'avait pas encore été fait à l'école doctorale. Ses conseils avisés, son expertise et ses connaissances des rouages de l'Université ont été importants dans la réussite de ce projet. Je remercie aussi chaleureusement mes deux rapporteurs, M. Jean-Christophe Prevotet et M. Bruno Robisson pour leurs retours précieux quant à l'amélioration de la présentation des contributions que j'avais sélectionnées dans le cadre de cette VAE. Je remercie Mme Mounia Karbouche-Harrari, M. Cédric Killian et Mme Christelle Bahier-Porte pour avoir accepté la tâche d'examinatrices et examinateur et participé à ma soutenance, avec un enthousiasme qui m'a beaucoup touché.

L'expérience que j'ai souhaité valoriser à travers les travaux présentés dans ce manuscrit n'est pas solitaire. Elle aurait été, dans ce cas, sans aucune saveur. J'ai l'immense chance de travailler dans une équipe de recherche commune entre le CEA-Leti et les Mines de Saint-Etienne, au Centre de Microélectronique à Gardanne, dans laquelle j'ai pu m'épanouir professionnellement à travers des collaborations qui sont d'une valeur inestimable à mes yeux. Je tiens donc à remercier l'ensemble du département SAS et des équipes MicroPacks. Les travaux que je présente dans ce manuscrit n'auraient tout simplement pas pu aboutir sans Jean-Max Dutertre, Jean-Baptiste Rigaud, Olivier Potin, Simon Pontié et Alexandre Menu. Grâce au soutien du CEA (et surtout, indéfectible, de M. Christophe Villemazet) j'ai pu monter plusieurs projets de recherche avec des thèses et post-doctorat associés. Ces collaborations ont toutes été de fantastiques expériences humaines et scientifiques et je tiens à remercier Rémi Bernhard, Baptiste N'Guyen, Raphaël Joud, Kevin Hector, Brice Colombier, Mathieu Dumont et Clément Gaine pour avoir partagé (et supporté !) ces quelques années à mes côtés.

Je remercie aussi du fond du coeur mes proches, ma famille et amis, qui m'accompagnent et me soutiennent depuis toutes ces années.

Résumé

L'étude de la sécurité de l'Intelligence Artificielle (IA) a débuté bien avant l'émergence du *Deep Learning* [1, 2] mais ce sont les premiers travaux sur les fameux *adversarial examples* [3, 4] qui ont amorcé une petite révolution au sein de la communauté de l'IA. Dès lors, une multitude d'études ont démontré des attaques ciblant l'intégralité du *pipeline* traditionnel d'un système de Machine Learning (e.g., apprentissage, inférence) pour altérer leur *intégrité* de fonctionnement, leur *accessibilité* ou violer leur *confidentialité*. Par rapport aux premiers travaux sur la sécurité de l'IA, le contexte a radicalement changé : le développement accéléré de ces technologies a bouleversé les pratiques d'un grand nombre de domaines dont certains extrêmement critiques (santé, énergie, finance, cybersécurité, IoT industrielle, défense...). Aussi, les premières actions de régulation, normalisation et standardisation ont été amorcées, non sans résistance, particulièrement en Europe avec l'*IA Act*. Toutes ces actions ont néanmoins révélé combien les questions de sûreté et de sécurité sont, pour une grande part, sans réponse [5] et l'effort de recherche associé est encore immense, malgré plus d'une décennie d'efforts des communautés de l'*Adversarial Machine Learning* et du *Privacy-preserving Machine Learning*.

Parmi les nombreux sujets qu'il reste à traiter, celui du déploiement à très large échelle des modèles sur des plateformes matérielles dédiées à l'IA est considéré comme une priorité [1] pour les futurs normes et standards. Les travaux rassemblés dans ce manuscrit, dans le cadre d'une Validation des Acquis de l'Expérience (VAE) pour un Doctorat, s'inscrivent dans ce contexte. Il s'agit d'apporter des premières études indispensables à la caractérisation et l'évaluation des menaces propres au déploiement de réseaux de neurones profonds sur des plateformes typiques des systèmes embarqués. Dans notre cas, nos plateformes cibles sont les microcontrôleurs.

Dans un premier temps, nous décrivons dans le chapitre 3 le contexte des travaux présentés et apportons un état de l'art des principales menaces démontrées par la communauté de la sécurité de l'IA. Ces menaces sont principalement théoriques et algorithmiques, dans le sens où elles ne prennent pas en compte les failles liées aux implémentations logicielles ou matérielles des modèles. Nous détaillons donc par la suite les spécificités de l'IA embarquée et des menaces associées aux attaques physiques : les analyses par canaux auxiliaires (*side-channel analysis - SCA*) et par injection de fautes (*fault injection analysis - FIA*). Ces deux types d'attaques physiques sont au cœur des deux chapitres suivants.

Le chapitre 4 se concentre exclusivement sur une menace qui a pris aujourd'hui une réelle importance dans la communauté de sécurité matérielle : les attaques sur les paramètres des modèles de réseaux de neurones profonds. Dans un premier temps, nous apportons une analyse critique de l'attaque de référence, la Bit-Flip Attack [6], d'abord démontrée en simulation puis via Rowhammer contre des mémoires DRAM. Dans un second temps, dans un contexte d'évaluation sécuritaire, nous proposons la première mise en pratique, par injection laser, de l'altération des valeurs des paramètres pour des modèles embarqués sur un microcontrôleur ARM Cortex-M.

Le chapitre 5 démontre ensuite comment il est possible d'exploiter les analyses *side-channel* et l'injection de fautes pour réaliser des attaques dites d'*extraction de modèles*. Ces menaces de type *reverse-engineering* visent à extraire le maximum d'informations de modèles en boîte-noire, plus particulièrement leur architecture et la valeur de leurs paramètres. Ces attaques, complémentaires d'autres stratégies, démontrent que la prise en compte d'une surface d'attaque globale est fondamentale pour évaluer avec précision les risques contre les modèles embarqués.

Nous finissons ce manuscrit (chapitre 6) en discutant des différentes contributions proposées, notamment certaines limitations et des travaux futurs directement associés. Nous proposons aussi plusieurs perspectives pour élargir le périmètre de ces études, plus particulièrement en considérant des menaces propres à l'apprentissage embarqué, qui connaît un développement extrêmement fort, poussé par la multiplication des plateformes matérielles dédiées à l'IA.

1. *Cybersecurity of AI and Standardisation*, ENISA Report, 2023 : https://www.enisa.europa.eu/publications/cybersecurity-of-ai-and-standardisation

Table des matières

Acronymes

AdvBET Adversarial Bit Error Training
API Application Programming Interface

BFA Bit-Flip Attack

CEA Commissariat à l'Energie Atomique et aux Energies Alternatives
CNN Convolutional Neural Network
CPA Correlation Power Analysis
CV Computer Vision

DL Deep Learning
DNN Deep Neural Network
DP Differential Privacy
DSP Digital Signal Processor

EMFI Electromagnetic Fault Injection

FGSM Fast Gradient Signed Method
FIA Fault Injection Analysis
FPGA Field-Programmable Gate Array
FPU Floating Point Unit

GPU Graphics Processing Unit

IA Intelligence Artificielle
IoT Internet of Things

LETI Laboratoire d'Electronique et de Technologie de l'Information
LFI Laser Fault Injection
LIST Laboratoire d'Intégration des Systèmes et des Technologies
LLM Large Language Model

MCU MicroController Unit
MIA Membership Inference Attack
ML Machine Learning
MLP Multilayer Perceptrons
MSB Most Significant Bit
MSE Mines Saint-Etienne

NLP Natural Language Processing

PGD Projected Gradient Descent

RandBET Random Bit Error Training

SCA Side-Channel Analysis
SEA Safe Error Attack

VAE Validation des Acquis de l'Expérience

Chapitre 1

Parcours professionnel, périmètre du manuscrit

Contents

1.1 Validation des Acquis de l'Expérience (VAE)

La thèse présentée dans ce document correspond à une **Validation des Acquis de l'Expérience** (VAE) auprès de l'école doctorale ED SIS (488).

La Validation des Acquis de l'Expérience est issue de la *Loi de Modernisation Sociale* du 17 janvier 2002[1]. Elle permet d'obtenir un diplôme ou autres certifications à travers l'**équivalence d'une expérience professionnelle ou extra-professionnelle**. Elle se déroule en trois étapes :

1. Une phase de candidature pour vérifier la viabilité et conformité du projet de VAE.
2. Un dossier de candidature détaillant les expériences du candidat par rapport au diplôme visé dont la validation ouvre la troisième et dernière étape.
3. Un entretien avec un jury de professionnels et/ou de formateurs.

Par souci de cohérence, dans ce manuscrit, l'expérience présentée au titre de VAE ne concerne qu'une partie de ma carrière professionnelle et de mes travaux de recherche. Néanmoins, je présente dans les sections suivantes mon parcours, mes projets de recherche, l'ensemble de mes publications scientifiques, mes activités d'enseignements ainsi qu'un panel de communications. Par la suite (Cf. section 1.6), je définis plus précisément le périmètre des travaux qui concernent ce manuscrit qui sera défendu dans la phase d'entretien avec le jury.

1.2 Parcours professionnel

Mon parcours professionnel comporte trois grandes phases, principalement au sein de la Direction de la Recherche Technologique (DRT) du Commissariat à l'Energie Atomique et aux Energies Alternatives (CEA). Comme détaillé dans la Section 1.6, ce manuscrit se concentre uniquement sur la période au CEA LETI et mes activités (actuelles) portant sur la sécurité de l'IA embarquée.

Formation. Après une classe préparatoire aux grandes écoles au lycée Chateaubriand de Rennes, j'ai intégré l'École Nationale Supérieure de Techniques Avancées (ENSTA, Paris) en 1999 dont j'ai été diplômé en 2002 avec une option en *Multimédia*, *Traitement d'Images*, *Apprentissage Automatique*. En parallèle de ma dernière année à l'ENSTA, j'ai effectué le DEA IARFA (*Intelligence Artificielle*, *Reconnaissance des Formes et Applications*) à l'Université Paris 6.

Première expérience au CEA LIST [2003-2010]. A la suite de ce double diplôme, j'ai intégré en 2003 le jeune *Laboratoire d'Ingénierie de la Connaissance Multimédia Multilingue* (LIC2M) au sein du CEA LIST (Fontenay-Aux-Roses), dirigé par Christian FLUHR. Entre 2003 et 2010, j'ai effectué au sein de ce laboratoire des activités d'ingénieur chercheur sur la recherche d'information multimédia en utilisant plus spécifiquement des approches d'apprentissage statistique supervisé et non-supervisé. Ces activités ont été réalisées dans le cadre de programmes de recherche collaborative dont j'ai assuré la coordination pour le CEA LIST et monté et encadré deux thèses, celle de Christophe MILLET[2] (*Annotation automatique d'images : annotation cohérente et création automatique d'une base d'apprentissage*, 2004-2007) et Adrian POPESCU[3] (*Structures conceptuelles pour la recherche d'images sur Internet*, 2005-2008).

Expérience R&D en entreprise [2011-2016]. En 2010, j'ai quitté le CEA LIST pour intégrer le département de recherche de la société TWENGA, alors leader français dans la recherche d'information pour le e-commerce. Pendant cette période, mes activités ont principalement porté sur la recherche de solutions de l'état de l'art en recherche d'information multimédia et multimodale mettant en œuvre du *Machine Learning* et le montage d'actions de recherche collaborative avec le milieu académique. En pleine crise concurrentielle (Cf. action de la Cour de justice de l'Union Européenne contre *Google Shopping* pour abus de position dominante), j'ai quitté cette entreprise et suis retourné au CEA en 2016.

1. https://www.legifrance.gouv.fr/loda/id/JORFTEXT000000408905 Art. 133-145
2. Dirigée par Isabelle Bloch et coencadrée par Patrick Hède
3. Dirigée par Ioannis Kanellos et coencadrée par Gregory Grefenstette

Expérience au CEA LETI [2016-...]. Fin 2016, j'ai intégré le laboratoire de recherche commune SAS (*Systèmes et Architectures Sécurisés*) entre le CEA LETI et l'école des Mines de Saint-Étienne (MSE, cursus ISMIN) au sein du Campus de Microélectronique Aix-Marseille-Provence Georges Charpak [4]. L'expertise de cette équipe (et plus généralement du service SSSEC – *Sécurité des Systèmes Électroniques et des Composants* – au sein du LETI) est la sécurité matérielle et plus particulièrement les menaces dites physiques et les protections associées. Cette équipe commune repose sur une expertise reconnue et des équipements de pointes en analyse par canaux auxiliaires (*side channel*, SCA) et injection de fautes (FIA). Si le cœur des activités de recherche de l'équipe concerne la caractérisation sécuritaire et la sécurisation de modules cryptographiques, j'ai proposé, dès 2017, d'ajouter les systèmes d'IA embarquée. Cette activité deviendra une thématique à part entière au sein de l'équipe commune et au sein du LETI/SSSEC et pour laquelle j'assure la coordination scientifique. C'est à ce titre que je suis nommé [5] "*Expert Senior*" au sein du CEA dans le domaine "Mathématiques / Informatique scientifique / Logiciel" pour la spécialité "Intelligence Artificielle, Systèmes Experts / Sécurité de l'apprentissage automatique".

Ce manuscrit se concentre exclusivement sur cette période d'activité.

1.3 Principaux projets de recherche

Pendant mon parcours au sein du CEA, j'ai pu mener plusieurs projets de recherche collaborative. Les principaux sont énumérés et brièvement décris ci-dessous :

Pour le CEA LIST :

- ImagEVAL (2005-2007, Techno-Vision, Ministère de la Recherche) : campagne d'évaluation nationale pour la reconnaissance et la recherche d'images. Coordination scientifique
- GEORAMA (2008-2011, ANR) : Recherche d'images géographiques par la création et l'exploitation de ressources structurées à large échelle et de méthodes d'analyse et de recherche par le contenu. Coordinateur.
- MUSCLE (2004-2007, EU FP6 NoE) : *Multimedia Understanding through Semantics, Computation and Learning* [6]. Réseau d'excellence européen (42 partenaires) dirigé par l'INRIA. Responsable scientifique pour le CEA.

Pour le CEA LETI :

- PROSECCO (2015-2018, ANR) : Génération de code sécurisé avec des protections formellement prouvées. Caractérisation sécuritaire de modules cryptographiques et mécanismes d'authentification par analyse side-channel et injection de fautes.
- COFFI (2019-2023, ANR) : Intégrité du flot d'exécution – du logiciel à la microarchitecture. Responsable pour le CEA.
- ANR PICTURE (2021-2024, ANR) : Sécurité physique et intrinsèque des modèles de réseaux de neurones embarqués. Coordinateur.
- PULSE (2021-..., IRT Nanoelec) : Programme PULSE, Responsable de la thématique "*Sécurité des Implémentations de l'IA*".
- INSECTT (EU ECSEL, 2020-2023) : *Intelligent Secure Trustable Things*. Coordinateur scientifique du consortium Français : CEA, IDEMIA, STMicroelectronics, AKEOPLUS), responsable de la tâche d'évaluation et de vérification des systèmes d'IA.
- COMPROMIS (2024-2029, PEPR) : Sécurité des données multimédia, sécurité de l'IA.
- AI.MMUNITY (2024-2028, ANR) : *Security-by-design for robust federated learning systems*. Coordinateur.

1.4 Publications

Ma production scientifique se concentre sur mes deux périodes d'activités au sein du CEA.

1.4.1 CEA LIST (Laboratoire d'Intégration des Systèmes et des Technologies)

De 2004 à 2010, mes publications portent sur l'utilisation d'approches statistiques et plus particulièrement de *machine learning* pour la recherche d'information multimédia et multilingue.

4. https://www.mines-stetienne.fr/campus/campus-aix-marseille-provence/
5. 24/07/2023
6. http://muscle.ercim.eu/

Conférences internationales avec actes et comité de lecture :

- M. Joint, P.A. Moëllic, P. Hède, et al. HEKA : A general tool for multimedia indexing and research by content. In : 16th Annual Symposium Electronic Imaging, IS&T/SPIE, San Jose. 2004.
- P. Hède, P.A. Moëllic, J. Bourgeoys, et al. Automatic generation of natural language description for images. In : Computer-Assisted Information Retrieval - RIAO 2004, 7th International Conference, University of Avignon, France, April 26-28, 2004. p. 306-313.
- R. Besançon, P. Hède, P.A. Moëllic, et al. Cross-media feedback strategies : Merging text and image information to improve image retrieval. In : Multilingual Information Access for Text, Speech and Images : 5th Workshop of the Cross-Language Evaluation Forum, CLEF 2004, Bath, UK, September 15-17, 2004, Revised Selected Papers 5. Springer Berlin Heidelberg, 2005. p. 709-717.
- C. Millet, I. Bloch, P. Hède., P.A. Moëllic. Using relative spatial relationships to improve individual region recognition. In : The 2nd European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology, 2005. EWIMT 2005.(Ref. No. 2005/11099). IET, 2005. p. 119-126.
- P.A. Moëllic, P. Hède, G. Grefenstette, et al. Evaluating Content-Based Image Retrieval Techniques with the One Million Images CLIC TestBed. In : The Second World Enformatika Conference, WEC'05, February 25-27, 2005, Istanbul, Turkey, 2005. p. 171-174.
- S. Zinger, C. Millet, B. Mathieu, P.A. Moëllic, et al. Extracting an ontology of portrayable objects from WordNet. In : MUSCLE/ImageCLEF workshop on Image and Video retrieval evaluation. 2005. p. 17-23.
- C. Fluhr, P.A. Moëllic, P. Hède. Usage-oriented multimedia information retrieval technological evaluation. In : Proceedings of the 8th ACM international workshop on Multimedia information retrieval. 2006. p. 301-306.
- S. Zinger, C. Millet, B. Mathieu, P.A. Moëllic, et al. Clustering and semantically filtering web images to create a large-scale image ontology. In : Proceedings of Internet Imaging VII. SPIE, 2006. p. 89-97.
- A. Popescu, C. Millet, et P.A. Moëllic. Ontology driven content-based image retrieval. In : Proceedings of the 6th ACM international conference on Image and video retrieval. 2007. p. 387-394.
- A. Popescu, P.A. Moëllic. Olive : a conceptual web image search engine. In : Proceedings of the 15th ACM international conference on Multimedia. 2007. p. 147-148.
- A. Popescu, P.A. Moëllic, C. Millet. Semretriev : an ontology driven image retrieval system. In : Proceedings of the 6th ACM international conference on Image and video retrieval. 2007. p. 113-116.
- A. Popescu, P.A. Moëllic, I. Kanellos. ThemExplorer : finding and browsing geo-referenced images. In : Proceedings of the International Workshop on Content-Based Multimedia Indexing. IEEE, London, UK, 2008. p. 576-583.
- A. Popescu, G. Grefenstette, P.A. Moëllic. Gazetiki : automatic creation of a geographical gazetteer. In : Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries. 2008. p. 85-93.
- A. Popescu, H. Le Borgne, P.A. Moëllic. Conceptual image retrieval over a large scale database. In : Evaluating Systems for Multilingual and Multimodal Information Access : 9th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, Aarhus, Denmark, Revised Selected Papers 9. Springer Berlin Heidelberg, 2009. p. 771-778.
- P.A. Moëllic, J.E. Haugeard, G. Pitel. Image clustering based on a shared nearest neighbors approach for tagged collections. In : Proceedings of the 2008 International Conference on Content-based image and video retrieval. 2008. p. 269-278.
- A. Popescu, S. Souidi, P.A. Moëllic. See the world with themexplorer. In : Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries. 2008. p. 432-432.
- A. Popescu, P.A. Moëllic. MonuAnno : automatic annotation of georeferenced landmarks images. In : Proceedings of the ACM International Conference on Image and Video Retrieval. 2009. p. 1-8.
- A. Popescu, G. Grefenstette, P.A. Moëllic. Mining tourist information from user-supplied collections. In : Proceedings of the 18th ACM conference on Information and knowledge management. 2009. p. 1713-1716.
- A. Popescu, P.A. Moëllic, I. Kanellos, et al. Lightweight web image reranking. In : Proceedings of the 17th ACM International conference on multimedia. 2009. p. 657-660.
- D. Myoupo, A. Popescu, H. Le Borgne, P.A. Moëllic. Visual Reranking for Image Retrieval over the Wikipedia Corpus. In : Working Notes for CLEF 2009 Workshop co-located with

the 13th European Conference on Digital Libraries (ECDL 2009) , Corfù, Greece, September 30 - October 2, 2009. CEUR Workshop Proceedings 1175
— D. Myoupo, A. Popescu, H. Le Borgne, P.A. Moëllic. Multimodal image retrieval over a large database. In : Multilingual Information Access Evaluation II. Multimedia Experiments : 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009, Corfu, Greece, September 30-October 2, 2009, Revised Selected Papers 10. Springer Berlin Heidelberg, 2010. p. 177-184.

Journaux avec comité de lecture :
— C. Millet, I. Bloch, P. Hède, P.A. Moëllic. Automatic cleaning and segmentation of web images based on colors to build learning databases. Image and Vision Computing, 2010, vol. 28, no 3, p. 317-328.

Livre (chapitre) :
— A. Popescu, G. Grefenstette, P.A. Moëllic. Improving image retrieval using semantic resources. Advances in semantic media adaptation and personalization, 2008, Springer Berlin, Heidelberg, p. 75-96.

1.4.2 CEA LETI (Laboratoire d'Electronique et de Technologie de l'Information)

A partir de 2016, mes publications s'intéressent à la sécurité de l'IA embarquée.
Conférences internationales avec actes et comité de lecture :
— R. Bernhard, P.A. Moëllic, J.M. Dutertre. Impact of low-bitwidth quantization on the adversarial robustness for embedded neural networks. In : 2019 International Conference on Cyberworlds (CW). IEEE, 2019. p. 308-315. **(best paper)**
— B. Colombier, A. Menu, J.M. Dutertre, P.A. Moëllic. Laser-induced single-bit faults in flash memory : Instructions corruption on a 32-bit microcontroller. In : 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2019. p. 1-10.
— R. Bernhard, P.A. Moëllic, J.M. Dutertre, et al. Adversarial Robustness of Quantized Embedded Neural Networks. Computer & Electronics Security Applications Rendez-vous, 2019.
— A. Menu, J.M. Dutertre, Rigaud, Jean-Baptiste, et P.A. Moëllic. Single-bit laser fault model in NOR flash memories : analysis and exploitation. In : 2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). IEEE, 2020. p. 41-48.
— R. Bernhard, P.A. Moëllic, J.M. Dutertre. Luring Transferable Adversarial Perturbations for Deep Neural Networks. In : 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, 2021. p. 1-8.
— R. Cohendet, M. Solinas, R. Bernhard, P.A. Moëllic et al. Impact of reverberation through deep neural networks on adversarial perturbations. In : 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2021. p. 840-846.
— R. Bernhard, P.A. Moëllic, M. Mermillod, et al. Impact of spatial frequency based constraints on adversarial robustness. In : 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, 2021. p. 1-8.
— M. Dumont, P.A. Moëllic, R. Viera, et al. An overview of laser injection against embedded neural network models. In : 2021 IEEE 7th World Forum on Internet of Things (WF-IoT). IEEE, 2021. p. 616-621.
— R. Joud, P.A. Moëllic, R. Bernhard, et al. A Review of Confidentiality Threats Against Embedded Neural Network Models. In : 2021 IEEE 7th World Forum on Internet of Things (WF-IoT). IEEE, 2021. p. 610-615.
— R. Viera, J.M. Dutertre, M. Dumont, et P.A. Moëllic. Permanent laser fault injection into the flash memory of a microcontroller. In : 2021 19th IEEE International New Circuits and Systems Conference (NEWCAS). IEEE, 2021. p. 1-4.
— K. Hector, P.A. Moëllic, M. Dumont, et al. A closer look at evaluating the Bit-Flip Attack against deep neural networks. In : 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS). IEEE, 2022. p. 1-5.
— B. Nguyen, P.A. Moëllic, S. Blayac. Evaluation of Convolution Primitives for Embedded Neural Networks on 32-bit Microcontrollers. In : 22nd International Conference on Intelligent Systems Design and Applications (ISDA'22), 2022.
— R. Joud, P.A. Moëllic, S. Pontié, et al. A Practical Introduction to Side-Channel Extraction of Deep Neural Network Parameters. In : Smart Card Research and Advanced Applications :

21st International Conference, CARDIS 2022, Birmingham, UK, November 7–9, 2022, Revised Selected Papers. Cham : Springer International Publishing, 2023. p. 45-65.
— B. Nguyen, P.A. Moëllic, S. Blayac. Domain Generalization on Constrained Platforms : On the Compatibility with Pruning Techniques. In : Internet of Things : 5th The Global IoT Summit, GIoTS 2022, Dublin, Ireland, June 20–23, 2022, Revised Selected Papers. Cham : Springer International Publishing, 2023. p. 250-261.
— C. Gaine, P.A. Moëllic, J.M. Dutertre, O. Potin, Fault Injection on Embedded Neural Networks : Impact of a Single Instruction Skip 26th Euromicro Conference on Digital System Design (DSD, AHSA special sessions), 2023
— M. Dumont, K. Hector, P.A. Moëllic, J.M. Dutertre, S. Pontié. Evaluation of Parameter-based Attacks against Embedded Neural Networks with Laser Injection In International Conference on Computer Safety, Reliability, and Security. SafeComp 2023.
— K. Hector, P.A. Moëllic, J.M. Dutertre, M. Dumont. Fault Injection and Safe-Error Attack for Extraction of Embedded Neural Network Models In International Workshop on Security and Artificial Intelligence SECAI Workshop / ESORICS. 2023. **(best paper)**
— R. Joud, P.A. Moëllic, S. Pontié, J.B. Rigaud, Like an Open Book ? Read Neural Network Architecture with Simple Power Analysis on 32-bit Microcontrollers, In : Smart Card Research and Advanced Applications : 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14–16, 2023, Revised Selected Papers. P. 256-276

Conférences nationales avec actes et comité de lecture :
— P.A. Moëllic. The dark side of neural networks : an advocacy for security in machine learning. In Computer & Electronics Security Applications Rendez-vous 2018, CESAR, 2018.

Journaux avec comité de lecture :
— D. Paulin, R. Joud, C. Hennebert, P.A. Moëllic, et al. HistoTrust : tracing AI behavior with secure hardware and blockchain technology. Annals of Telecommunications, 2023

Livre (chapitre, *open access*) :
— M. Karner, J. Peltola, M. Jerne, L. Kulas, P. Priller. (2024). Intelligent secure trustable things. Chapitre : *Algorithmic and Implementation-Based Threats for the Security of Embedded Machine Learning Models*, Springer Nature, page 105-125.

Brevet :
— M-A. Solinas, M. Reyboz, M. Mermillod, S. Rousset, P-A. Moëllic, Method and device for controlling a system using an artificial neural network based on continual learning, Serial N° 18/061,470, Enregistrement : 04/12/2022.

1.5 Enseignements

Depuis l'année scolaire 2018-2019, je suis responsable de l'UP[7] *Machine Learning* au sein du GP[8] *Systèmes Informatiques* pour les étudiants en deuxième année du cursus ISMIN de l'école Mines Saint-Etienne. Pour l'année 2024-2025, correspondant à la période de rédaction de ce manuscrit, cette activité comprend 21 heures :

- 5 cours théoriques (15 heures) : Introduction et théorie du *Machine Learning*, *Deep Learning*. Les cours sont accessibles pour consultation[9].
- 2 travaux pratiques (6 heures) : apprentissage non supervisé et supervisé, *Deep Learning*.

J'assure l'intégralité des cours théoriques en promotion entière ainsi que les travaux pratiques avec trois intervenants (1/4 de promotion). Pour les étudiants poursuivant en troisième année avec la *majeure Systèmes Embarqués*, j'assure un cours de 6 heures sur la sécurité de l'IA embarquée. Le responsable des cursus mentionnés est M. Olivier POTIN (enseignant-chercheur, MSE).

1.6 Périmètre des travaux présentés dans ce manuscrit

7. *Unité Pédagogique* ou Matière dans le référentiel des Mines de Saint-Etienne
8. Groupe Pédagogique, *idem*.
9. `https://gitlab.emse.fr/pierre-alain.MOELLIC/teaching/-/tree/main/MinesSaintEtienne`

1.6.1 Périmètre professionnel

Les travaux présentés dans ce manuscrit concernent mes activités de recherche à partir de 2016 au sein de l'équipe commune de recherche SAS entre le CEA LETI et l'Ecole des Mines de Saint-Etienne. Ces travaux se sont structurés à travers plusieurs programmes et projets de recherche collaborative dont j'ai assuré la coordination scientifique. Principalement :

- ANR PICTURE : Sécurité physique et intrinsèque de l'inférence de modèles de réseaux de neurones embarqués.
- EU ECSEL INSECTT : Sécurité de l'IA embarquée pour l'IoT
- IRT NANOELEC, PULSE : Implémentation d'IA sécurisée

En plus des publications citées dans la section précédente, ces projets ont donné lieu à plusieurs communications, plus particulièrement :

- FIC'2021 (*Enjeux de sécurité lorsque l'IA s'insère dans les objets connectés*)
- CRYPTARCHI'2022[10] (*Laser Fault Injection Against Embedded Neural Network*)
- AMUSEC'2022[11] (*Sécurité de l'IA Embarquée : une surface d'attaque complexe*)
- JAIF'2023[12] (*Fault Injection and Embedded Neural Networks : Integrity & Confidentiality*)
- ESSAI'2024[13] (*Fault Injection and Safe-Error Attack for Extraction of Embedded Neural Network Models*)

Ces projets ont conduit à des thèses et post-doctorats dont j'ai initié les sujets et assuré l'encadrement pour le CEA LETI. Dans le cadre de l'équipe commune de recherche SAS, les thèses concernées ont été dirigées par des enseignants-chercheurs de MSE : Jean-Max Dutertre, Jean-Baptiste Rigaud et Sylvain Blayac.

- **Rémi BERNHARD** (PhD), 11-2018 / 11-2021. Direction de thèse : Jean-Max DUTERTRE (MSE). *Intégrité des réseaux de neurones : défenses contre les exemples adverses et leur transférabilité.*
- **Brice COLOMBIER** (PostDoc), 2017 / 2018, Collaboration : Jean-Max DUTERTRE (MSE), Alexandre MENU (MSE). *Injection de fautes laser bi-spots.*
- **Baptiste NGUYEN** (PhD), 02-2020 / 02-2023, Direction de thèse, Sylvain BLAYAC (MSE). *Machine learning pour l'exploitation de données temporelles : domain adaptation et optimisation d'inférence sur des microcontrôleurs low power.*
- **Raphaël JOUD** (PhD), 11-2021 / 11-2024, Co-encadrement : Simon PONTIE (CEA LETI), Direction de thèse : Jean-Baptiste RIGAUD (MSE). *Attaques side-channel contre la confidentialité des modèles de machine learning embarqués : attaques protections, évaluations.*
- **Kevin HECTOR** (PhD), 05-2022 / 05-2024, Direction de thèse : Jean-Max DUTERTRE (MSE). *Injection de fautes contre l'intégrité et la confidentialité des réseaux de neurones embarqués.*
- **Mathieu DUMONT** (PostDoc), 2020 - 2023, Collaboration : Jean-Max DUTERTRE, *Injection de fautes laser contre réseaux de neurones sur microcontrôleurs.*

Thèses en cours (liées au PEPR COMPROMIS et ANR AI.MMUNITY) :

- **Bastien VUILLOD** (PhD), 11-2023 / 11-2026. Direction de thèse : Jean-Max DUTERTRE. *Sécurité de l'apprentissage profond décentralisé : empoisonnement du federated learning.*
- **Benjamin HUBINET** (PhD), 12-2024 / 12-2027, Co-encadrement : Olivier SAVRY (CEA LETI), Direction de thèse : Jean-Baptiste RIGAUD (MSE). *Security-by-design pour les modèles de réseaux de neurones embarqués sur plateformes RISC-V.*
- **Assaad ALSANEH** (PhD), 11-2025 / 11-2028, Direction de thèse : Cédric GOUY-PAILLER (CEA LIST). *Intégrité, accessibilité et confidentialité de l'IA embarquée dans les étapes de post apprentissage.*

NB : Dans le manuscrit, les publications correspondant à ce périmètre sont indiquées en gras et orange : [X].

10. https://labh-curien.univ-st-etienne.fr/cryptarchi/workshop22/
11. https://amusec.i2m.univ-amu.fr/index.php/programme-2022/
12. https://jaif.io/2023/
13. https://essai-conference.eu/

1.6.2 Périmètre scientifique

Les travaux présentés dans ce manuscrit concernent tous des menaces à l'**inférence** qui exploitent plus particulièrement des failles dites physiques comme les **analyses *side-channel*** ou **l'injection de fautes** que nous définissons dans la section 3.4. A travers ces travaux, nous couvrons des menaces qui visent l'**intégrité**, la **disponibilité** et la **confidentialité** des modèles. Notons que nous ne nous intéressons pas aux menaces visant spécifiquement la confidentialité des données (d'apprentissage ou d'inférence). Pour chacune des menaces étudiées, plusieurs capacités adverses sont analysées en **boîte blanche** et **boîte noire**. Nous résumons ce périmètre d'étude dans le tableau suivant :

Table 1.1 – Périmètre des travaux présentés dans le manuscrit.

Menaces	**Inférence**			Apprentissage
	Intégrité	*Disponibilité*	*Confidentialité*	
Evasion & quantification	✓(3.4.2.4)			∅
Saut d'instruction		✓(3.6.2)		∅
Faute sur les paramètres		✓(4)	✓(5.3)	∅
Analyse *side-channel*			✓(5.2, 5.1)	∅

Chapitre 2

Notations, formalisme et jeux de données

Contents

2.1 Notations

Ensembles, données, valeurs.

— L'ensemble des entiers entre A et B (inclus) est noté $[\![A, B]\!]$.
— Le cardinal d'un ensemble $\mathcal{E}$ est noté $|\mathcal{E}|$ et représente le nombre d'éléments appartenant à $\mathcal{E}$.
— Un entier signé x peut être représentée sous forme binaire par $(b_0 b_1 ... b_{N-1})_2$. On appellera b_0 le bit le plus significatif ou selon l'abréviation anglais MSB. Dans la majorité des sections du manuscrit, $N = 8$ et la représentation est dite à "complément à deux" : quand le MSB est à 1 l'entier est négatif et à 0, l'entier est positif.
— Une valeur réelle flottante a sous 32 bits sous la norme IEEE-754 (`binary32`) possède un bit de signe (S_a) puis 8 bits d'exposant (E_a) et enfin 23 bits de mantisse (M_a) :

$$\begin{aligned} a &= (-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2 - 127} \times \left(1.b_{22}...b_0\right)_2 \\ &= (-1)^{S_a} \times 2^{E_a - 127} \times \left(1 + 2^{-23} \times M_a\right) \end{aligned} \tag{2.1}$$

Pour une valeur réelle x :
— $\lfloor x \rfloor$ est l'arrondi à l'entier inférieur.
— $\lceil x \rceil$ est l'arrondi à l'entier supérieur.
— $\lfloor x \rceil$ est l'arrondi à l'entier le plus proche.

Norme, produit scalaire, distance. La norme L_p de $v \in \mathbb{R}^d$ pour $0 < p < \infty$, aussi notée $\|v\|_p$, est :

$$\|v\|_p = \left(\sum_{i=1}^{d} |v_i|^p \right)^{\frac{1}{p}} \tag{2.2}$$

La norme L_∞, aussi notée $\|v\|_\infty$ est donc :

$$\|v\|_\infty = \max_{i=1}^{d} |v_i| \tag{2.3}$$

Et la norme L_0 notée $\|v\|_0$:

$$\|v\|_0 = \sum_{i=1}^{d} \mathbb{1}_{\{v_i \neq 0\}} \tag{2.4}$$

La fonction indicatrice $\mathbb{1}$ est définie dans l'équation 2.8.
Soient deux vecteurs u et v, le produit scalaire (*dot product*) entre u et v est noté $u \cdot v$. La distance cosinus entre u et v est définie par :

$$\text{cossim}(u, v) = \frac{u \cdot v}{\|u\|_2 \|v\|_2} \tag{2.5}$$

La distance euclidienne entre deux vecteur u et $v \in \mathbb{R}^d$ est :

$$d_2(u, v) = \|u - v\|_2 = \left(\sum_{i=1}^{d} |(u_i - v_i)|^2 \right)^{\frac{1}{2}} \tag{2.6}$$

La distance de Hamming entre deux représentations binaires a et b de même taille N est notée $d_H(a, b)$ et représente le nombre de bits qui ont des valeurs différentes entre a et b :

$$d_H(a, b) = \sum_{i=0}^{N-1} a_i \oplus b_i \tag{2.7}$$

Pour deux vecteurs, matrices ou tenseurs W_a et W_b de même dimension, la distance de Hamming, notée simplement $d_H(W_a, W_b)$ s'applique à chacune de leurs composantes.

Matrices, tenseurs, indices. Les matrices ou tenseurs seront simplement notés en lettres capitales. Les indices font référence aux dimensions de la matrice ou du tenseur et les exposants entre parenthèses à une instance particulière. Ainsi, pour un ensemble X de n données dans $\mathbb{R}^{n,m}$, le $k^{\text{ième}}$ échantillon (dans $\mathbb{R}^{n,m}$) sera noté $X^{(k)}$ et $X^{(k)}_{i,j}$ représentera la valeur réelle des $i^{\text{ième}}$ et $j^{\text{ième}}$ dimensions.

Pour un vecteur $u \in \mathbb{R}^d$, u_i est la composante de u à l'indice (dimension) $i \in [\![0, d-1]\!]$.

Fonctions caractéristiques.
Considérant une condition $\mathcal{D}$, la fonction *indicatrice* (ou *caractéristique*) de $\mathcal{D}$ est notée $\mathbb{1}_{\mathcal{D}}$:

$$\mathbb{1}_{\mathcal{D}} = \begin{cases} 1 \text{ if } \mathcal{D} \text{ is true} \\ 0 \text{ otherwise} \end{cases} \tag{2.8}$$

La fonction de signe $sign$ pour une valeur réelle a est :

$$\text{sign}(a) = \begin{cases} 1 \text{ if } a > 0 \\ -1 \text{ if } a < 0 \\ 0 \text{ if } a = 0 \end{cases} \tag{2.9}$$

Appliquée à un vecteur, la fonction $sign$ correspond au vecteur de même dimension composé du signe de chaque composante du vecteur.

La fonction $clip$ pour une valeur réelle a contraint cette valeur à une borne minimale u et maximale v :

$$\text{clip}(a, u, v) = \min(\max(a, u), v) \tag{2.10}$$

De la même façon que $sign$, la fonction $clip$ pour un vecteur s'applique à chacune de ses composantes.

Gradient. Soit une fonction f, telle que :

$$f : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_3}$$
$$x_1 \times x_2 \mapsto y$$

Le gradient de la fonction f au point (x_1, x_2) par rapport à la variable x_1 est notée $\nabla_{x_1} f(x_1, x_2)$. (ou $\frac{\partial f}{\partial x_1}(x_1, x_2)$).

Distributions. $\mathcal{N}(m, \sigma^2)$ représente la distribution normale de moyenne m et de déviation standard σ. La variance est σ^2. Une variable aléatoire X suivant $\mathcal{N}(m, \sigma^2)$ sera dite *normale* ou *gaussienne*. $\mathcal{U}(a, b)$ représente la distribution uniforme où la densité de probabilité est $p = 1/(b-a)$ dans $[a, b]$ (0 sinon).

Mesures de performance. Pour éviter toute confusion entre les termes anglais *Accuracy* et *Precision*, qui peuvent renvoyer à des mesures de performances différentes en fonction de la nature des systèmes, nous garderons le terme anglais "*accuracy*" (plutôt qu'*exactitude*). Pour un système de prédiction, l'*accuracy* mesure le taux de prédictions correctes jugées par rapport à une vérité-terrain.

2.2 *Machine Learning* supervisé et réseaux de neurones

2.2.1 Machine Learning

Nous utilisons la définition de Tom Mitchell [7] sur l'apprentissage :

*On dit d'un programme informatique qu'il apprend de l'**expérience** $\mathbb{E}$ par rapport à une classe de **tâches** $\mathbb{T}$ et d'une mesure de **performance** $\mathbb{P}$, si sa performance pour réaliser les tâches $\mathbb{T}$, telle que mesurée par $\mathbb{P}$, s'améliore avec l'expérience $\mathbb{E}$.*

L'expérience $\mathbb{E}$ régit les trois grands paradigmes d'apprentissage. Quand $\mathbb{E}$ correspond à un ensemble de données dit *d'entraînement*, qui sont autant d'échantillons représentatifs des entrées et - potentiellement - des sorties de la tâche $\mathbb{T}$ à accomplir, on parle d'*apprentissage non-supervisé* (*unsupervised learning*) ou *supervisé* (*supervised learning*). Dans le cas d'un apprentissage supervisé, les données correspondent à des entrées pour lesquelles sont associées une connaissance (i.e., une *supervision*) sur la sortie attendue par la tâche. L'exemple le plus caractéristique est une tâche de classification d'images ou pour chaque image d'apprentissage est associée un label (e.g., "chat"). Dans le cas d'un apprentissage non-supervisé, cette connaissance n'est pas apportée et les données d'entrée *brutes* sont accessibles par l'algorithme. Pour l'exemple précédent de classification d'images, sans labels associés, le programme doit distinguer les différentes les catégories d'images en estimant la distribution des données dans l'espace des images. Enfin, l'apprentissage par *renforcement* correspond à un paradigme ou l'expérience $\mathbb{E}$ est un ensemble d'interactions avec un environnement régit par des règles. L'exemple le plus populaire de ce paradigme est l'apprentissage d'un jeu de plateau (échecs, go...) ou du déplacement d'un robot dans un environnement contraint.

Ces trois paradigmes d'apprentissage ne sont pas hermétiques les uns par rapport aux autres. Il est tout à fait possible de considérer un apprentissage *semi-supervisé* dans le cas où l'ensemble d'apprentissage n'est que partiellement annoté ("*labélisé*"). De même, il est classique d'associer de l'apprentissage supervisé à du renforcement pour apporter des connaissances supplémentaires à la réalisation d'une tâche complexe. Notons aussi que les succès impressionnants de l'*IA Générative* s'expliquent en grande partie par l'essor de l'apprentissage *auto-supervisé* (*Self Supervised Learning*) [8], hybridation astucieuse entre l'apprentissage non-supervisé et supervisé pour pallier l'absence de bases de données annotées à très large échelle tout en profitant des masses considérables de données hétérogènes présentes sur Internet.

L'ensemble des travaux présentés dans ce manuscrit s'inscrit dans de l'apprentissage supervisé et plus particulièrement des tâches de classification qui représentent une large part de l'IA moderne.

Les réseaux de neurones profonds (DNN) sont des modèles paramétriques. Un modèle sera noté M_W et ses paramètres internes W. Dans un contexte d'apprentissage supervisé, un modèle paramétrique cherche à apprendre une fonction de transfert (*mapping*) entre un espace d'entrée multidimensionnel, noté $\mathcal{X} \subset \mathbb{R}^d$ et un espace de sortie, noté $\mathcal{Y}$. Les données traitées par la modèle pour réaliser sa tâche sont issues d'une distribution $\mathcal{D}$ sur $\mathcal{X}$. Une tâche classique en apprentissage supervisé est la classification pour laquelle l'espace de sortie est un ensemble fini de $|Y|$ classes, aussi appelés "labels". Par rapport à une entrée $x \in \mathcal{X}$, la sortie du modèle $M_W(x)$ est un ensemble de scores $M_W^{(0)}(x), ..., M_W^{(i)}(x), ..., M_W^{(|Y|)}(x)$. La *prédiction* du modèle M_W pour x sera alors $\hat{y} = \text{argmax}_i(M_W^{(i)}(x))$.

A l'apprentissage, les paramètres W du modèle M_W sont adaptés pour optimiser la performance $\mathbb{P}$ à la tâche de classification. L'apprentissage étant supervisé, le modèle dispose d'une base d'apprentissage notée (X^{train}, Y^{train}) avec laquelle il est possible de calculer une mesure de performance correspondant à une *erreur de prédiction*. Cette erreur $\mathcal{E}_{rr}$ est calculée par une fonction d'erreur ou de coût, appelée *loss* et notée $\mathcal{L}$:

$$\mathcal{E}_{rr} = \mathcal{L}\Big(\hat{y}, y\Big) = \mathcal{L}\Big(M_W(x), y\Big) \tag{2.11}$$

Les paramètres optimaux W^* sont ceux qui minimisent le *risque empirique* définit comme l'espérance de la fonction de *loss* :

$$W^* = \underset{W}{\text{argmin}} \underset{x,y\sim\mathcal{D}}{\mathbb{E}}\Big[\mathcal{L}(M_W(x), y)\Big] \tag{2.12}$$

Pratiquement, on utilise donc la base d'apprentissage (X^{train}, Y^{train}) comme un échantillonnage représentatif et l'équation précédente devient :

$$W^* = \underset{W}{\text{argmin}} \sum_{x,y\in(X^{train},Y^{train})} \mathcal{L}(M_W(x), y) \tag{2.13}$$

Pour des tâches de classification, une fonction de *loss* classique est la *categorical cross-entropy* :

$$\mathcal{L}(\hat{y}, y) = -\sum_{c}^{N} y^{(c)} \log\Big(\hat{y}^{(c)}\Big) \tag{2.14}$$

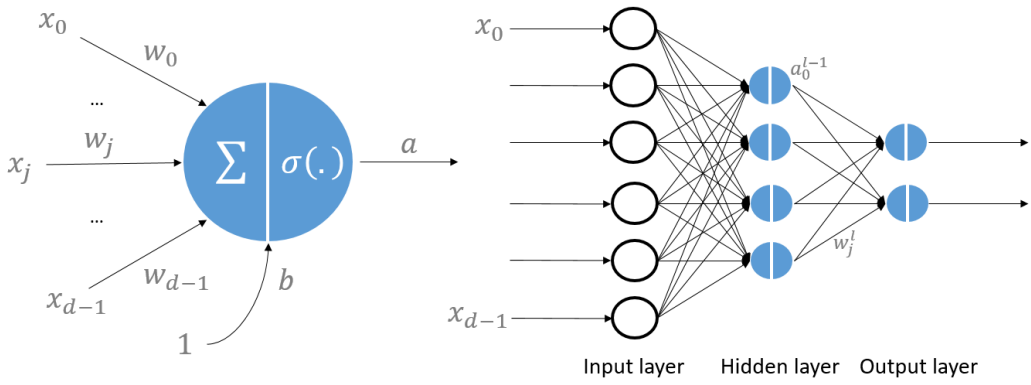


FIGURE 2.1 – (Gauche) Perceptron. (Droite) Multi-layer perceptron(MLP).

Avec $\hat{y}^{(c)}$ le score de prédiction pour la classe c et $y^{(c)}$ la vérité terrain issue de la base d'apprentissage.

Comme expliqué dans la section suivante (2.2.3), la résolution de l'équation 2.13 se fait par une technique de descente de gradient de la fonction de *loss* $\mathcal{L}$ par rapport aux paramètres $(\nabla_W \mathcal{L})$ [9].

2.2.2 Réseaux de neurones profonds

Perceptron. Le perceptron est la forme la plus simple d'un modèle paramétrique qui combine linéairement les données d'entrée $x \in \mathbb{R}^d$ avec les paramètres $w \in \mathbb{R}^d$ qui incluent un biais $b \in \mathbb{R}$. Le résultat de la somme pondérée est ensuite transmis à une fonction d'activation σ monotone, non linéaire, pour produire la sortie a du perceptron (Cf. Fig. 2.1) :

$$a(x) = \sigma\left(\sum_{j=0}^{d-1} w_j x_j + b\right) \tag{2.15}$$

***Multilayer Perceptrons* (MLP).** Il est possible de regrouper des perceptrons au sein d'une couche (*layer*), les perceptrons (aussi appelé *neurones* ou *unités*) recevront les mêmes entrées mais auront des paramètres propres. Plusieurs couches de neurones peuvent ainsi être combinées les unes après les autres permettant de constituer un réseau de neurones multicouches (MLP, Cf. Fig. 2.1). Dans un MLP, chaque neurone d'une couche est connecté à l'ensemble des sorties des neurones de la couche précédente. Les MLP sont aussi logiquement nommés *fully-connected neural networks* et les couches de perceptron sont aussi appelées "dense" ou "linéaire". L'équation 2.16 donne l'expression de la sortie d'un neurone j d'une couche l par rapport à la couche précédente $l-1$.

$$a_j^l(x) = \sigma\left(\sum_{i \in (l-1)} w_{i,j} a_i^{l-1} + b_j\right) \tag{2.16}$$

Avec $w_{i,j}$ le paramètre du neurone j de la couche l associé à la sortie du neurone i de la couche $(l-1)$. b_j est le biais du neurone j de la couche l. a_i^{l-1} et a_j^l sont, respectivement, les valeurs des sorties du neurone i de la couche $(l-1)$ et du neurone j de la couche l.

La dernière couche produit la sortie du modèle avec les scores de prédiction associés à chaque label de Y. Il est généralement utile de normaliser ces scores sous une forme de distribution de probabilité : $M_w^{(c)}(x) \in [0,1]$ et $\sum_c M_w^{(c)}(x) = 1$. Ceci est réalisé en utilisant la fonction Softmax :

$$\sigma_{Softmax}(a_i) = \frac{e^{a_i}}{\sum_{c=0}^{|Y|-1} e^{a_c}} \tag{2.17}$$

Fonction d'activation. Les fonctions d'activation assurent la non-linéarité des modèles pour une meilleure adaptation à des tâches (et des données) complexes. Les fonctions les plus utilisées sont la fonction Sigmoid, Tanh, Softmax (Cf. Eq. 2.17) et ReLU (*Rectifed Linear Unit*) qui est la

plus répandue. ReLU est linéaire par partie avec une partie nulle pour les entrées négatives, une non-linéarité à l'origine $x = 0$ et la fonction identité ($y = x$) pour les entrées positives. ReLU est traditionnellement définie par l'équation 2.18 :

$$\sigma_{ReLu}(x) = max(0, x) \tag{2.18}$$

***Convolutional Neural Network* (CNN).** Les réseaux de neurones convolutifs reposent sur des couches de convolutions qui appliquent des filtres (ou *kernels*), généralement de petite taille, sur un tenseur d'entrée. Le tenseur de sortie représente donc des caractéristiques (*features*) spécifiques à chaque filtre. La convolution entre un filtre et le tenseur est paramétrée par plusieurs facteurs. Le *stride*, s, détermine le pas de déplacement le long du tenseur. Le *padding*, p, permet de gérer la convolution aux bords du tenseur : le *padding "same"* ajoute des 0 en périphérie de telle sorte que la taille du tenseur de sortie soit identique à celle du tenseur d'entrée ; le *padding "valid"* va appliquer le filtre uniquement sur la zone du tenseur d'entrée où le filtre peut s'appliquer entièrement, ce qui conduira à un tenseur de sortie de taille plus petite. Généralement, nous manipulons des tenseurs et des filtres carrés. En posant $H_{in} \times W_{in}$ la taille du tenseur d'entrée (hauteur, largeur) et $H_{out} \times W_{out}$ celle du tenseur de sortie, la relation entre les deux tailles en fonction du *stride* et du *padding* est donnée par (l'équation est la même pour W_{out}) :

$$H_{out} = \left\lfloor \frac{H_{in} - k_h + 2p}{s} \right\rfloor + 1 \tag{2.19}$$

Formellement, une couche de convolution considère un tenseur X_{in} de dimension $[C_{in}, W_{in}, H_{in}]$ et applique un ensemble T_k composé de K filtres de dimension $[K, k_z, k_w, k_h]$ pour produire un tenseur de sortie X_{out} de dimension $[C_{out}, W_{out}, H_{out}]$. C, H et W sont respectivement le nombre de canaux (i.e., nombre de filtres), la largeur et hauteur du tenseur. Comme pour n'importe quelle couche neuronale, l'opération de convolution est suivie d'une activation non-linéaire σ. Pour un filtre $f (\in [\![0; K-1]\!])$ et une position $(i, j) \in [\![0; W_{out}]\!] \times [\![0; H_{out}]\!]$, sans formaliser le *stride* ni *padding*, les équations de la couche de convolution sont :

$$X_{out}(f, i, j) = b^{(f)} + \sum_{c=0}^{C_{in}-1} \sum_{m=0}^{k_w-1} \sum_{n=0}^{k_h-1} X_{in}(c, i+m, j+n) \cdot T_k(f, c, m, n) \tag{2.20}$$

$$a = \sigma(X_{out}) \tag{2.21}$$

Le nombre de paramètres à entraîner est donc $K \times (k_z \times k_w \times k_h + 1)$ avec $C_{out} = K$ et $C_{in} = k_z$.

Afin de réduire la taille des tenseurs de sortie et d'assurer un certain niveau d'invariance en translation, les CNNs associent les couches de convolution à du *pooling*. L'idée est de conserver localement des statistiques de premier ordre, typiquement la valeur maximale (*Max Pooling*) ou la moyenne (*Average Pooling*). Appliqué à des images, un CNN standard utilisera un *pooling* de taille (2×2), avec stride= 2 et *padding "same"*, qui divisera par deux la taille H_{out} et W_{out} (la profondeur C reste inchangée).

La succession de couches de convolution et de *pooling* permet à un CNN d'extraire des caractéristiques d'abord de bas niveau puis de plus haut niveau d'abstraction. Généralement, un CNN se termine par une série de couches *fully-connected* (comme pour un MLP) permettant de combiner les *features* pour construire la prédiction finale.

2.2.3 ML *pipeline* et apprentissage

Le *Machine Learning pipeline* représente l'ensemble des étapes permettant la création d'un système basé sur un ou plusieurs modèles de ML. Bien que le cycle de vie d'un tel système peut être particulièrement complexe, le *ML pipeline* est souvent simplifié à travers les processus de collecte de données, d'entraînement, de déploiement des modèles puis d'inférence :

- **Collecte et traitement des données.** Ces processus correspondent à l'acquisition des données à partir d'un ensemble de sources potentiellement très hétérogènes (e.g., sources ouvertes, capteurs) puis à plusieurs traitements permettant d'en améliorer la qualité ainsi que leur représentativité et de favoriser la convergence des modèles.
- La phase d'**entraînement** est l'étape d'optimisation du modèle à partir des données et des métriques de performance de la tâche à accomplir. Un ensemble de données d'entraînement est utilisé, noté X^{train}, pour l'optimisation des paramètres (cf. équation 2.22) ainsi qu'un ensemble de validation, noté X^{val} pour détecter d'éventuels problèmes de généralisation et plus particulièrement d'*overfitting*.

— **Déploiement.** Le modèle entraîné est ensuite déployé sur des *moteurs d'inférence* qui peuvent être de nature et de capacité très différentes, en allant du serveur de calcul aux cibles matérielles fortement contraintes comme des microcontrôleurs *low power*. Aussi, le déploiement peut faire intervenir une phase d'optimisation de modèle pour que celui-ci réponde aux différentes contraintes du moteur d'inférence (e.g., mémoire, latence).
— Enfin, la phase d'**inférence** correspond à la mise en place du modèle dans son environnement de production réel. Des processus de *monitoring* sont généralement associés pour surveiller les niveaux de performances.

Un *pipeline* plus complexe, mais plus réaliste, est proposé dans la figure 2.2. Il prend notamment en compte la notion de cycle de vie du système avec des processus dynamiques (plus particulièrement le *continuous training*) qui modélisent le fait que les données et le(s) modèle(s) ne sont pas statiques mais peuvent évoluer au cours du temps. Ce type de *pipeline* est au cœur des principaux *frameworks* de *MLOps* (Machine Learning Operations) qui appliquent les principes du *DevOps* (*Development Operations*) au ML.

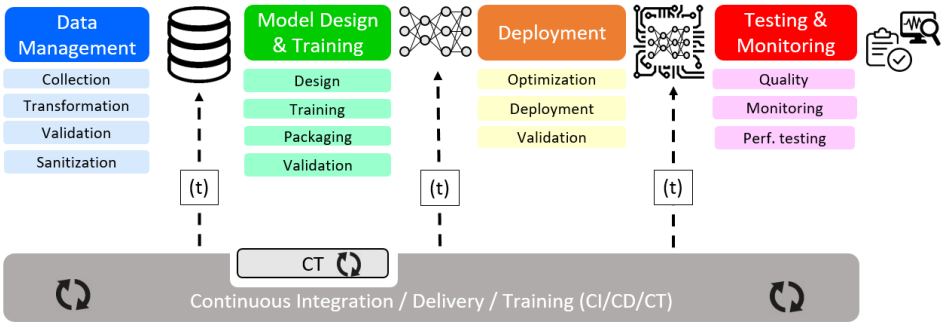


FIGURE 2.2 – Pipeline traditionnel du Machine Learning

L'étape d'apprentissage est évidemment une phase cruciale et repose, dans le cas d'un modèle de réseau de neurones profonds, sur un processus itératif, souvent coûteux en temps de calcul, permettant d'optimiser l'ensemble de ses paramètres. L'entraînement repose sur une technique de descente de gradient de l'erreur de prédiction, appelée *rétro-propagation du gradient* (*gradient backpropagation*). Comme le calcul de l'erreur de prédiction (*loss*) entre la prédiction $\hat{y}$ et la vérité-terrain y ne peut se faire qu'à la fin du modèle, les premiers paramètres optimisés, selon l'équation 2.22, sont ceux de la dernière couche. Puis, le gradient est propagé vers le début du modèle (*backward pass*) pour optimiser les couches antérieures. Les gradients sont calculés sur une petite portion de X^{train}, appelée *batch* et noté $\mathcal{B}$. Le paramètre est optimisé en lui retranchant une portion du gradient définie par le *learning rate* λ. Ce dernier influence la vitesse et la qualité de convergence de l'apprentissage. L'équation de mise à jour est la suivante (2.22) :

$$w^{t+1} = w^t - \lambda \sum_{x \in \mathcal{B}} \frac{\partial \mathcal{L}(M(x), y)}{\partial w}\bigg|_{w=w^t} \tag{2.22}$$

2.3 Jeux de données utilisés

Pour les travaux présentés dans ce manuscrit dans les chapitres 3, 4 et 5, nous utilisons majoritairement des jeux de données de référence en *Deep Learning* et en sécurité de l'IA qui correspondent à des tâches de classification d'images supervisée. Des échantillons représentatifs de ces jeux de données sont proposés dans la Figure 2.3.

MNIST. La base de données MNIST (*Modified National Institute of Standards and Technology*) [10] est composée d'images en niveaux de gris de taille 28×28 pixels et représentant des *digits* ("0" à "9") manuscrits. L'ensemble d'apprentissage est composé de 60,000 images (équilibré par rapport aux 10 labels possibles) et 10,000 images pour la validation et le test.

Fashion-MNIST. Cette base de données est composée de 70,000 images en niveaux de gris de taille 28×28 pixels (60,000 pour l'entraînement et 10,000 pour la validation et le test), réparties en 10 catégories de vêtement [11] (les images sont tirées de Zalando).

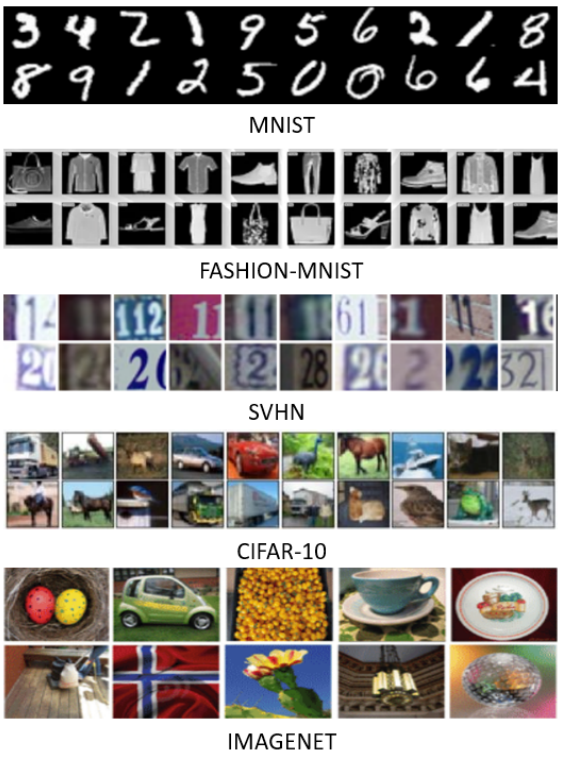


Figure 2.3 – Échantillons des bases de données utilisées.

SVHN. La base de données SVHN (*Street View House Numbers*) [12] est composée d'images naturelles de taille 32×32 en couleur de nombres, prises à partir d'images de Google Street View. Comme pour MNIST, chaque image correspond à un *digit* ("0" à "9") au centre de l'image. L'ensemble d'apprentissage correspond à 73,257 images et celui de test à 26,032 images.

CIFAR-10. Le *benchmark* CIFAR-10 (*Canadian Institute For Advanced Research*) [13] est composé d'images naturelles en couleur (RGB) provenant du Web correspondant à 10 labels (*airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, *truck*). Les images font 32×32 pixels. L'ensemble d'apprentissage regroupe 50,000 images et celui de test 10,000 images. Une version élargie correspondant à 100 classes est aussi proposée et connue sous le nom de CIFAR100.

ImageNet. ImageNet [14] provient du challenge ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) est une base d'images à large échelle composée d'image naturelle et correspondant à 1,000 labels. La version la plus utilisée propose des images de taille 224×224 pixels. L'ensemble d'apprentissage est composé de 1.2 million d'images et 50,000 pour le test et la validation.

Chapitre 3

Sécurité de l'Intelligence Artificielle

Contents

Introduction. La sécurité de l'Intelligence Artificielle est un sujet dont l'importance n'a cessé de croître, parallèlement au développement extrêmement rapide de modèles permettant aujourd'hui de traiter une multitude de tâches complexes. Les problèmes de sécurité et de sûreté de l'IA sont aujourd'hui au cœur des nombreuses actions de régulation et de standardisation, plus particulièrement en Europe avec l'*IA Act*. La fulgurante évolution de l'IA ne repose pas uniquement sur la complexité de modèles, nourris pas des quantités toujours plus importantes de données, mais aussi par leur déploiement à très large échelle. Ceci s'explique par le développement de plateformes matérielles variées et dédiées spécifiquement à l'IA permettant l'apprentissage et l'inférence *on-device* de modèles de ML, comme par exemple la généralisation de l'IA dans les *smartphones* ou l'utilisation de modèles de réseaux de neurones profonds de l'état de l'art dans des plateformes fortement contraintes comme des microcontrôleurs (par l'exemple dans l'IoT).

Ce déploiement a pour conséquence une augmentation significative de la surface d'attaque, puisqu'un attaquant – en plus des nombreux vecteurs d'attaques algorithmiques démontrés dans l'état de l'art – peut exploiter des failles liées aux implémentations logicielles et matérielles.

Organisation. Dans ce chapitre, nous apportons dans un premier temps quelques éléments de contexte permettant de situer les travaux présentés en fin de chapitre ainsi que dans les chapitres 4 et 5 du manuscrit. Dans un deuxième temps, nous définissons les éléments caractéristiques d'un *modèle de menace* propre à la sécurité de l'IA. Ensuite, dans la section 3.3.1, nous proposons un panorama des principales menaces algorithmiques démontrées par la communauté de l'*adversarial ML* et du *privacy-preserving ML*. Après avoir rappelé les caractéristiques propres à l'IA embarquée (3.4), nous décrivons l'état de l'art des menaces exploitant deux attaques physiques : les analyses *side-channel* (3.5) et l'injection de fautes (3.6) pour laquelle nous proposons des premiers résultats sur le saut d'instruction.

Projets, encadrements et publications associées. L'analyse de l'état de l'art sur la sécurité de l'IA a été principalement réalisée dans le cadre des projets (Cf. section 1.3) :

- ANR PICTURE (AAPG 2020, 2021-2024, coordinateur)
- IRT NANOELEC, Programme PULSE (2021-2025, responsable *Implémentation Sécurisée d'IA*)
- EU ECSEL INSECTT (2020-2023, coordinateur France, responsable *AI Validation & Verification*)
- ANR AI.MMUNITY (AAPG 2024, en cours, coordinateur)

Les travaux sur la quantification et les attaques par évasion (*adversarial examples*, section 3.4.2.4) ont été réalisés à travers l'encadrement de la thèse de **Rémi BERNHARD** avec la collaboration de **Jean-Max DUTERTRE** (MSE), et ceux portant sur l'injection de fautes (saut d'instruction, section 3.6.2) avec le post-doctorat de **Clément GAINE** (ANR PICTURE) en collaboration de **Jean-Max DUTERTRE** (MSE) et **Olivier POTIN** (MSE).

Une partie des résultats présentés dans ce chapitre ont été publiés dans les publications suivantes :

- [15] Bernhard, R., Moëllic P-A., & Dutertre, J-M. *Impact of low-bitwidth quantization on the adversarial robustness for embedded neural networks*. In 2019 International Conference on Cyberworlds (CW) (pp. 308-315). *Best Paper*.
- [16] Gaine, C., Moëllic, P. A., Potin, O., Dutertre, J. M., *Fault Injection on Embedded Neural Networks : Impact of a Single Instruction Skip*. In 26th Euromicro Conference on Digital System Design (DSD 2023) (pp. 317-324).

3.1 Contexte

3.1.1 Développement de l'IA moderne

Le terme "Intelligence Artificielle" (IA) correspond à un ensemble de disciplines parmi lesquels l'apprentissage machine ou *Machine Learning* (ML). Depuis la fin des années 60, le développement de l'IA a suivi un cheminement assez erratique avec plusieurs phases d'accélérations. Les plus récentes et notables concernent l'apprentissage profond (*Deep Learning* - DL).

Le *Deep Learning* est un ensemble de méthodes que l'on peut classer dans le *Representation Learning*, qui correspond à la capacité des modèles à apprendre à partir de données brutes par l'extraction de caractéristiques (ou *features)* de haut niveau pour réaliser une ou plusieurs tâches

spécifiques. Le *representation learning* représente une vraie rupture par rapport aux techniques de ML plus anciennes, comme les *Support Vector Machine* (SVM) par exemple, pour lesquelles il était nécessaire de traiter les données, en amont de l'apprentissage, pour extraire des caractéristiques pertinentes (par exemple, des histogrammes de couleurs ou de textures pour le traitement d'images). Les modèles associés au DL sont les réseaux de neurones profonds (*Deep Neural Network*). Leur structure "profonde", compilant des couches de neurones les unes après les autres, permet à un modèle d'extraire des caractéristiques hiérarchiques : les premières couches extraient des caractéristiques de bas niveaux (e.g., des contours, contrastes chromatiques pour de la classification d'images) et les dernières couches des caractéristiques de hauts niveaux proches du niveau sémantique de la sortie attendue pour la tâche à réaliser (e.g., des *patterns* ou formes). L'émergence du DL depuis le début des années 2010s a profondément transformé le ML, et l'IA d'une manière générale, avec un niveau de complexité des modèles qui n'a cessé de croître. La dernière "révolution" est l'IA Générative (*Generative AI*) qui a vu la taille, la complexité et les capacités des modèles exploser de façon significative en moins de 10 ans. Cette dernière évolution repose sur deux avancées fondamentales : (1) l'utilisation massive de données non supervisées à l'échelle du Web grâce au *Self-Supervised Learning* et (2) l'utilisation d'architectures unifiées, quelle que soit la nature des données (image, texte, vidéo, *time series*...), basées sur les mécanismes d'attention.

Au moment de la rédaction de ce manuscrit, la question de l'avenir à court et moyen termes de l'IA est un débat brûlant dans les communautés scientifiques associées à l'IA. Les très grands modèles de langues (LLM) et autres modèles dits de "fondation" atteignent un plateau de performance et ne disposent plus de données fraîches. L'explosion de la taille des modèles et des coûts (financiers, énergétiques) associés à leur apprentissage, le développement de modèles pouvant "raisonner" sur des problèmes logiques et/ou mathématiques sont autant de questions très importantes mais qui concernent malheureusement un cercle de plus en plus restreint d'acteurs privés de l'IA avec un risque de monopole technologique (données, plateformes de développement, modèles).

3.1.2 Déploiement à large échelle

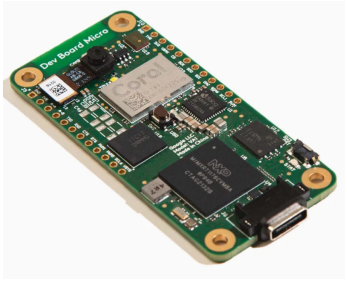
Si le développement de l'IA moderne repose principalement sur la croissance exceptionnelle de la complexité des modèles, le déploiement à très large échelle des modèles est une autre tendance très importante qui explique le succès fulgurant de l'IA en à peine une décennie. Ce déploiement concerne autant les domaines d'applications que la nature des plateformes matérielles sur lesquelles sont portés des modèles de DL pour leur entraînement et/ou leur inférence.

Diversification des applications. Le ML "pré-Deep Learning" était principalement restreint aux tâches de vision par ordinateur (*Computer Vision*, CV), le traitement automatique de la langue (*Natural Language Processing*, NLP) et de la parole (*Speech-to-Text*). Aujourd'hui, l'IA est utilisée dans une multitude de domaines d'applications pour répondre à des tâches complexes et variées (classification, régression, synthèse...) en manipulant des données fortement hétérogènes (images, textes, séries temporelles, données *tabulées*...).

Multiplication des plateformes HW. Le paradigme d'utilisation des modèles de DL a longtemps été restreint à un modèle très centralisé, de type "*ML-as-a-service*", où l'apprentissage est réalisé sur des serveurs de calculs (GPU) puis, les modèles sont utilisés à distance pour l'inférence, généralement sur des serveurs avec des architectures similaires, à travers une API utilisateur. Si ce paradigme est toujours d'actualité, la diffusion des modèles, pour l'inférence mais aussi pour l'apprentissage, a fortement diversifié les usages. La raison principale est la multiplication de plateformes matérielles spécifiquement développées pour supporter des modèles de DL et notamment des plateformes mobiles contraintes (mémoires, consommations, latence...), comme illustré dans la figure 3.1. Un bon exemple est la Coral Micro [1] qui intègre deux Cortex-M (M4 et M7) associés à un accélérateur de type TPU (*Tensor Processing Unit*, Google) avec 128 Mo de mémoire Flash et 46 Mo de RAM. Avec ce type de plateforme, il est tout à fait possible de faire tourner un modèle de reconnaissance d'objet et un "petit" LLM de 15 millions de paramètres adapté de LLAMA2 (Meta) [2]. Un autre exemple est le téléphone Google Pixel 9 dont l'architecture matérielle est significativement tournée pour les applications d'IA en local sur le *smartphone*. Ceci s'accompagne d'un état de l'art de plus en plus conséquent sur l'optimisation et la compression

1. https://coral.ai/products/dev-board-micro/
2. Cf. projet llama4micro de Max Braun - Google - https://github.com/maxbbraun/llama4micro

(a) STM32N6 avec NPU



(b) Coral Nano avec Cortex M4-M7 et TPU

(c) NVIDIA Orin Nano (GPU)

FIGURE 3.1 – Plateformes pour l'IA embarquée, avec accélérateur dédié. NPU : *Neural Processing Unit*, TPU : *Tensor Processing Unit*, GPU : *Graphics Processing Unit*.

des modèles (optimisation de paramètres, quantification, élagage de paramètres, architectures optimisées).

3.1.3 L'IA à l'heure de la régulation

Les sujets développés dans ce manuscrit s'inscrivent dans un contexte normatif et législatif particulier, à l'échelle Européenne et internationale. La question de la régulation de l'IA est aujourd'hui centrale et soulève des débats passionnés bien au-delà de la communauté scientifique. Progressivement, alors que les modèles devenaient de plus en plus complexes et puissants, la notion d'"IA de Confiance" (*Trustworthy AI*) a pris une place considérable dans le débat public et a été la pierre angulaire de la majorité des projets de régulation. Cette notion de confiance repose sur des grands principes :

— *l'explicabilité*,
— la sûreté (de fonctionnement) et
— la sécurité.

L'initiative européenne de l'*IA Act* a été l'une des premières au monde à poser un cadre de régulation quant à l'utilisation de l'IA pour des systèmes considérés comme critiques (Cf. Figure 3.2 pour les 4 niveaux de risques considérés). La pleine mise en œuvre de l'*IA Act* est prévue pour l'été 2026. Un lobbying particulièrement intense s'est opposé à ce texte et à d'autres projets de régulation comme le SB-1047 en Californie, finalement censuré (29/09/2024) par une décision du gouverneur de l'état. Ce SB-1047 californien a démontré combien la communauté scientifique était très divisée sur l'idée même de régulation de l'IA, s'opposant sur des questions, pourtant fondamentales, de la responsabilité des créateurs de modèles ou la confidentialité et la *privacy* des données d'apprentissage.

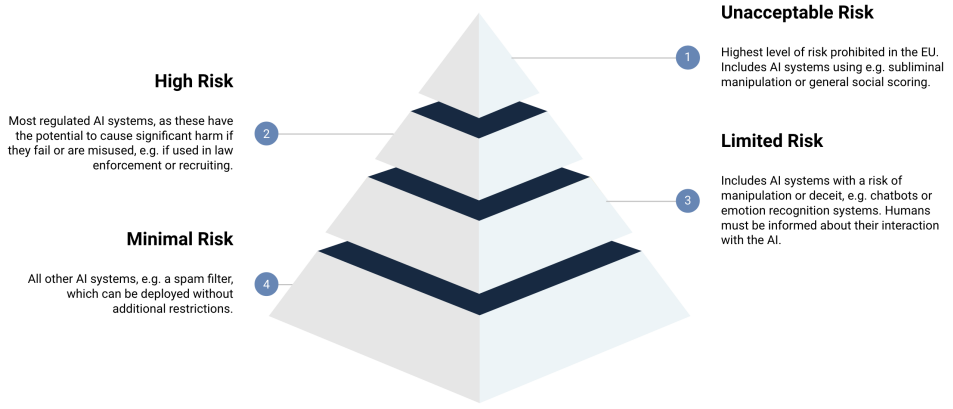


FIGURE 3.2 – Les quatre niveaux de risque définis dans l'*IA Act* (source trail-ml.com)

En parallèle des grandes actions de régulation de l'IA, une impressionnante activité normative et de standardisation est engagée depuis plusieurs années autour de l'IA et de son application dans de nombreux domaines critiques (e.g., l'aéronautique). L'*IA Act* Européen repose ainsi sur

des dizaines d'actions de standardisation, la majorité encore en cours de rédaction et de finalisation dans les instances internationales ou européennes, qui concernent autant les processus définissant le *pipeline* et le cycle de vie d'un système basé sur du ML que sa sûreté ou sa sécurité. Concernant la sécurité des systèmes basés sur de l'IA, le NIST a proposé un *AI Risk Management Framework*[3] pour sensibiliser les acteurs et utilisateurs d'IA aux nombreuses menaces, dont la majorité sont définies dans plusieurs taxonomies de références comme MITRE ATLAS[4].

A l'échelle européenne, les deux agences normatives (ETSI et CEN/CENELEC) ont multiplié les actions spécifiquement dédiées à la sécurité de l'IA, plus particulièrement, pour l'ETSI, le groupe *Security of AI industry specification* (ISG SAI[5]) et les actions sur *AI testing and certification* qui doivent supporter le déploiement de l'*IA Act* et, pour CEN/CENELEC, le Joint Technical Committee 21 (JTC-21, *Artificial Intelligence*) et 13 (JTC-13, *Cybersecurity and data protection*)[6].

3.2 Modèle de menace pour la sécurité de l'IA

La première étape de toute analyse de sécurité consiste à définir un *modèle de menace* c'est-à-dire à identifier les objectifs, les connaissances et les capacités des attaquants, ainsi que les vulnérabilités intrinsèques du système. Dans le domaine de la sécurité de l'IA, la définition de modèles de menace pertinents est devenue un besoin particulièrement critique [17], afin de mieux couvrir les surfaces d'attaque les plus importantes et d'anticiper les menaces émergentes. La définition d'un modèle de menace comporte les éléments suivants : la définition du système cible, les connaissances et capacités de l'adversaire, l'objectif de l'attaquant et les potentielles capacités du défenseur.

3.2.1 Le système cible

Les systèmes basés sur du ML couvrent une grande variété de contextes, allant de la nature des tâches à accomplir (e.g., classification, régression, prédiction), aux interactions avec l'utilisateur, en passant par les plateformes matérielles sur lesquelles les modèles sont éventuellement déployés, et les environnements logiciels. Du point de vue d'un *défenseur*, ces éléments doivent être parfaitement connus afin de définir correctement un ensemble de scénarios d'attaque réalistes.

Une distinction importante doit être faite entre les systèmes de ML basés sur des *capteurs* et ceux basés sur des APIs. Ces derniers représentent un cadre typique pour la majorité des travaux traitant des *adversarial examples* (voir Section 3.3.2). Dans un système basé sur une API, une entrée est directement fournie par un utilisateur : un attaquant potentiel a donc un contrôle total sur la manière d'alimenter le modèle, contrairement à un système avec des capteurs qui implique une couche physique supplémentaire.

Une autre caractéristique essentielle est liée au *pipeline* traditionnel de ML, défini en 2.2.3. Le système attaqué peut être uniquement axé sur l'inférence, c'est-à-dire que le processus d'entraînement a été réalisé sur un système différent avec des environnements matériels et logiciels distincts. Ensuite, le modèle entraîné est optimisé et déployé sur le système visé. À l'inverse, l'entraînement et l'inférence peuvent être regroupés sur un même système. Dans ce cas, un adversaire dispose d'une surface d'attaque plus large, car il peut également cibler le processus d'entraînement (par exemple avec des attaques par empoisonnement) pour mener des attaques avancées ciblant l'inférence.

3.2.2 Connaissances et capacités de l'adversaire

Pour atteindre son objectif, un adversaire peut disposer de capacités limitées ou, au contraire, étendues. Le premier critère important est sa capacité à effectuer une attaque au moment de l'entraînement et/ou de l'inférence :

— **Pendant l'entraînement** : l'attaquant cible directement la construction du modèle, par exemple en empoisonnant les données d'entraînement ou certains hyperparamètres. Ce cadre est très puissant car l'adversaire dispose d'un accès à l'algorithme d'apprentissage, à l'architecture du modèle et aux données d'entraînement.

3. https://www.nist.gov/itl/ai-risk-management-framework
4. https://atlas.mitre.org/matrices/ATLAS
5. https://www.etsi.org/technologies/securing-artificial-intelligence
6. https://www.cencenelec.eu/areas-of-work/cen-cenelec-topics/artificial-intelligence/

— **Lors de l'inférence** : l'attaquant corrompt généralement les données pour tromper le modèle ou exploite les paires entrée/sortie en interrogeant le modèle afin d'obtenir des informations. Le champ d'action de l'attaquant peut être limité par des restrictions appliquées aux utilisateurs (par exemple, authentification ou limitation du nombre de requêtes).

Un autre point essentiel pour définir les capacités de l'adversaire est le niveau de connaissance qu'il possède sur le modèle cible. Ce point est crucial pour établir tout scénario réaliste d'attaque.

— **Paradigme "boîte blanche" :** l'adversaire a une connaissance parfaite des éléments internes du modèle cible, c'est-à-dire son architecture et ses paramètres. De plus, dans un scénario de boîte blanche pur, l'adversaire peut également avoir accès aux données d'entraînement.

— **Paradigme "boîte noire" :** à l'inverse, dans un contexte d'attaque en boîte noire, l'adversaire ne dispose d'aucune information (ou seulement partielle) sur le modèle cible. Les capacités de l'attaquant reposent uniquement sur les possibilités d'interroger le modèle. Un adversaire peut avoir un accès illimité ou limité au modèle. Par exemple, pour une API de ML basée sur le cloud, un utilisateur (défini par une adresse IP et un identifiant) peut être limité à 50 requêtes par jour. Cette restriction est uniquement pertinente dans un contexte boîte noire, car un adversaire disposant d'une connaissance parfaite du modèle cible peut entraîner localement un modèle substitut avec un accès illimité.

3.2.3 Objectif d'une attaque

L'objectif d'un adversaire est défini selon la triade classique utilisée pour tous les systèmes d'information : **Confidentialité–Intégrité–Disponibilité**.

Dans la suite, nous considérons un modèle de réseau de neurone M_W, avec W ses paramètres, qui a été entraîné pour une tache supervisée de classification avec un ensemble d'apprentissage (X^{train}, Y^{train}, $|Y|$ est le nombre de labels). On parle de *feature*, la fonction $f_M(.)$ associée au modèle M qui a une donnée d'entrée x renvoie la sortie de l'avant-dernière couche.

3.2.3.1 Intégrité

Cibler l'*intégrité* signifie que l'attaquant cherche à dévier le comportement nominal de M_W en altérant les sorties attendues. Plusieurs modalités d'altération peuvent être définies en fonction de la précision et de l'objectif de l'attaque, comme les attaques par évasions ciblées ou non ciblées.

Les attaques les plus étudiées sur l'intégrité sont les attaques par évasion, aussi appelées *adversarial examples* (illustrées dans la Figure 3.3), qui sont détaillés dans la section 3.3.2. Une autre attaque pertinente contre l'intégrité d'un modèle est *l'empoisonnement*, où (X^{train}, Y^{train}) est modifié pendant l'entraînement pour compromettre les inférences futures.

Figure 3.3 – Illustration issue de [18] d'un *adversarial example* (attaque Projected Gradient Descent - PGD) à partir d'une image d'ImageNet. Une perturbation imperceptible est ajoutée à une image (correctement classée par un modèle) conduisant à une classification erronée.

Les attaques par empoisonnement permettent de cibler l'intégrité ou la disponibilité en fonction du niveau des altérations. Si l'adversaire cherche à corrompre l'intégrité du modèle cible, il concentrera les erreurs de classification sur un sous-ensemble très précis des inférences de manière à ce que le système garde une bonne performance moyenne. En d'autres termes, l'objectif est généralement de rester "*sous le radar*".

3.2.3.2 Disponibilité.

Menacer la disponibilité signifie que l'adversaire cible le système global ou l'environnement hébergeant les algorithmes de ML de manière à le rendre indisponible, ou détériore si gravement

les performances ou le comportement du système qu'il devient inutilisable. Plus précisément, les attaques liées à la disponibilité cherchent à altérer :

— l'accès au système,
— la qualité du système (par exemple, la confiance d'une prédiction),
— les performances du système (contraintes de mémoire, vitesse d'inférence, etc.).

Il est évident que l'intégrité et la disponibilité sont deux concepts étroitement liés. En effet, une attaque sur l'intégrité qui altère significativement les performances d'un modèle peut rendre ce dernier inutilisable. Par conséquent, les attaques basées sur l'intégrité constituent des vecteurs réalistes pour cibler la disponibilité, c'est notamment le cas pour les menaces par empoisonnement.

3.2.3.3 Confidentialité

Menacer la confidentialité concerne à la fois les données et le modèle. En effet, les données d'apprentissage et le modèle peuvent représenter tous deux des informations confidentielles et précieuses. Extraire des données capturées ou mémorisées dans un modèle peut être critique dans plusieurs domaines comme les applications médicales. Par exemple, Carlini et al. [19] montrent que des informations secrètes (notamment des numéros de sécurité sociale et de cartes de crédit) peuvent être extraites d'un modèle (par exemple un LSTM pour du NLP) entraîné avec des grandes collections d'emails. Une autre menace est connue sous le nom de *membership inference attacks*, MIA), où un attaquant tente de savoir si une entrée cible appartient à X^{train}, ce qui, dans certains cas, peut entraîner de graves atteintes à la confidentialité.

De la même manière, un attaquant peut extraire d'un modèle protégé (boîte noire) des informations sur sa structure ou ses paramètres. Par exemple, dans [20], une attaque d'*extraction de modèle* est réalisée contre des plateformes de ML en ligne (BigML et Amazon ML) en interrogeant simplement le modèle cible pour approximativement recréer un modèle substitut. L'état de l'art montre qu'en sélectionnant soigneusement les requêtes et même avec des sorties minimales (seulement le label avec le plus score de prédiction), des informations partielles sur le modèle peuvent être extraites. Ces menaces sont détaillées dans la section 3.3.3.

3.2.4 Scénario du "pire", attaquant adaptatif

Un piège classique consiste à sous-estimer les capacités et les connaissances d'attaquants "avancés". Un exemple typique concerne les systèmes embarqués. Avec la capacité d'accéder physiquement à un dispositif, un attaquant peut utiliser un clone ou construire une version substitutive pour profiler toutes les informations susceptibles de faciliter une attaque. Par conséquent, un concept important est celui du *scénario du pire* (*worst-case scenario*) qui est étroitement lié aux *attaques adaptatives* (*adaptive attacks*) pour le développement des protections. L'idée de base est que l'évaluation de l'efficacité d'une défense *doit* prendre en compte des attaquants potentiels qui ont connaissance de cette défense (ou, au moins, de son principe) et qui peuvent adapter leurs attaques pour contourner la protection. Plusieurs travaux, comme [21] pour les exemples adverses, montrent que de simples adaptations peuvent considérablement améliorer l'impact d'une attaque pour contourner une protection mal évaluée.

3.2.5 Capacité du défenseur

Comme mentionné dans [22], les capacités du défenseur sont rarement détaillées dans la littérature alors qu'elles représentent une connaissance importante pour définir correctement les modèles de menace et, surtout, pour concevoir des schémas de défense robustes. Serban *et al.* listent les étapes suivantes où un défenseur peut intervenir pour élaborer sa stratégie :

— *Prétraitements des entrées* : le défenseur cherche à surveiller les entrées qui alimentent les modèles, par exemple en excluant les valeurs aberrantes s'il a une bonne connaissance de la distribution des données.
— *Techniques de dissimulation* : un adversaire peut tirer profit des informations fournies par le système qui ne sont pas réellement nécessaires pour accomplir correctement une tâche. Par exemple, pour certains systèmes, fournir les scores pour un modèle de classification n'est généralement pas nécessaire et seul le label prédit (sans score de confiance) suffit.
— *Renforcement du modèle* (pendant l'entraînement) : de bonnes pratiques pendant la phase d'entraînement peuvent rendre le modèle plus robuste face aux attaques. Cela comprend une sélection appropriée et une construction des ensembles de données d'entraînement et

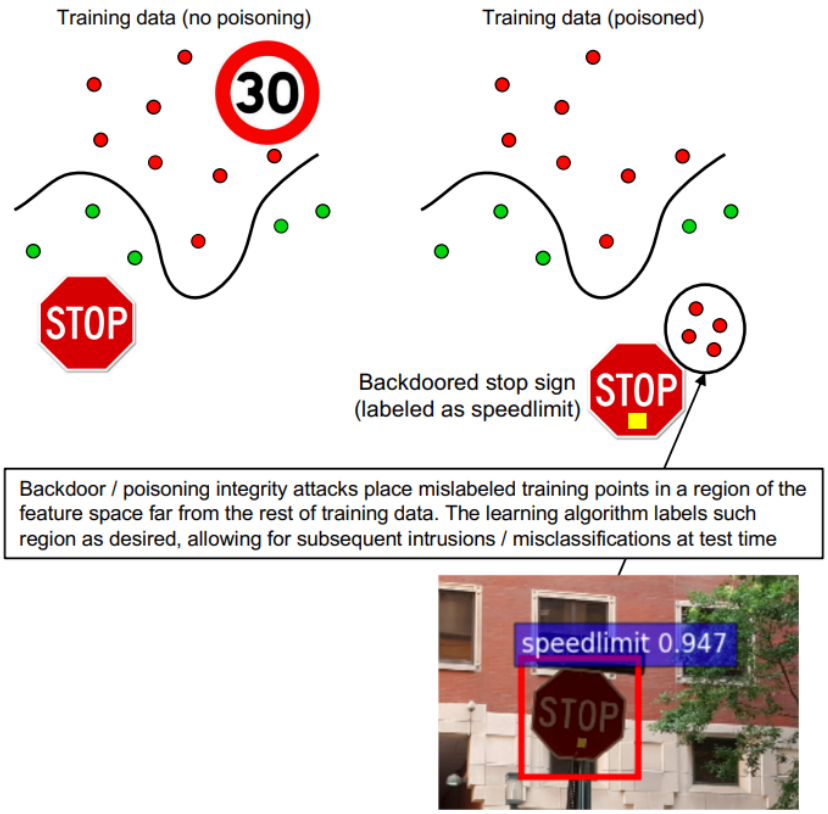


FIGURE 3.4 – Exemple d'empoisonnement de données (*backdoor attack)*. Source [24, 25]

.

de validation pour éviter les ensembles déséquilibrés et les biais. De plus, si cela est possible en fonction des caractéristiques et des exigences du système, l'utilisation de plusieurs modèles (méthode par ensemble) apparaît comme une bonne stratégie pour atténuer les menaces potentielles.

3.3 Panorama des menaces (algorithmiques)

Dans cette section, nous détaillons les principales menaces contre un modèle de ML et plus spécifiquement des modèles de réseaux de neurones profonds. Sauf exception, nous nous restreignons à des scénarios supervisés qui correspondent très majoritairement aux études de l'état de l'art. Les menaces décrites ci-dessous sont des menaces dites *algorithmiques* (aussi parfois nommées *API-based*) dans le sens où elles ne reposent pas sur des spécificités liées à l'implémentation (logicielle ou matérielle) des modèles (inférence ou apprentissage) ou des vecteurs d'attaques physiques comme c'est le cas dans les travaux présentés dans les chapitres 4 et 5 et dont l'état de l'art est présenté dans la section 3.4 suivante.

Par soucis de concision et pour rester dans le périmètre des travaux présentés dans ce manuscrit, les (nombreuses) menaces portant spécifiquement sur l'IA générative [23] et plus particulièrement les LLM (e.g., *prompt injection*) ne sont pas présentés. L'état de l'art sur ces nouvelles questions est conséquent et de nombreuses actions essaient de construire des benchmarks d'évaluation comme *JailBreakBench*[7].

3.3.1 Attaquer l'intégrité à l'apprentissage

A l'apprentissage, la principale catégorie d'attaques contre l'intégrité d'un modèle, concerne les menaces par empoisonnement (*poisoning attacks*). L'idée principale est d'altérer l'apprentissage d'un modèle pour qu'il apprenne un comportement malveillant. L'altération concerne généralement les données d'apprentissage [25,26] et le comportement malveillant cible le plus souvent une instance particulière, comme l'illustre la figure 3.4. Néanmoins, presque toutes les configu-

7. https://jailbreakbench.github.io/

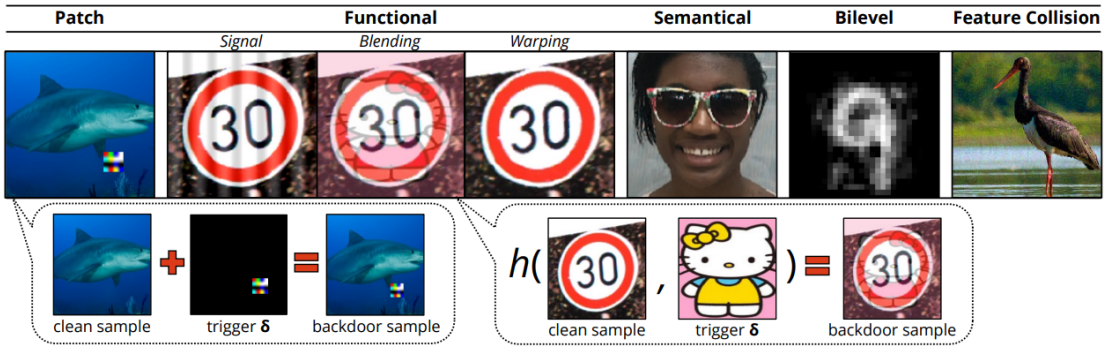


FIGURE 3.5 – Exemples de plusieurs types d'empoisonnement de données d'apprentissage. Source [29].

rations d'attaques ont été présentées dans la littérature, le sujet étant toujours très actif même pour les grands modèles de fondations [8] [27, 28].

Le premier type de *poisoning attacks* correspond aux menaces dites ***non-ciblées*** [9] qui peuvent aussi être catégorisées comme des menaces contre la disponibilité des modèles [29]. L'objectif est d'empoisonner les données d'apprentissage pour que le modèle se trompe indépendamment de l'entrée. On vise donc une décroissance significative de la performance moyenne du modèle à l'inférence pour le rendre *inutilisable*.

Plusieurs stratégies ont été proposées dans un contexte non-ciblé :

— **Label poisoning**. L'empoisonnement revient à permuter certains labels de l'ensemble d'apprentissage [30]. Une stratégie de permutation aléatoire est généralement sous optimale surtout pour les bases de données à large échelle comportant beaucoup de labels (e.g., ImageNet) puisque l'attaquant doit pratiquer un trop grand nombre de modifications.

— **Data & label poisoning**. L'attaquant peut altérer la donnée avec une perturbation adverse *et* le label associé.

— **Clean-label poisoning**. L'attaquant manipule une part généralement non négligeable des données mais sans modifier les labels. Cette stratégie concerne les attaques qui agissent dans le domaine des *features*. La proportion des données à empoissonner peut représenter un réel défi pour l'attaquant.

Les attaques ***ciblées*** cherchent à modifier les performances du modèle uniquement quand celui-ci est confronté à des données bien spécifiques. Il y a donc une contrainte supplémentaire par rapport aux attaques non-ciblées puisque le modèle doit conserver son niveau de performance sur toutes les autres données. Une référence est `PoisonFrog` [31] qui a montré (sur CIFAR-10) que l'empoisonnement de 50 données d'apprentissage permettait de corrompre la classification d'une image spécifique avec un taux de succès de 70%. L'idée principale est le principe de *feature collision* : l'attaquant cherche à modifier une donnée d'apprentissage x de telle façon que la *feature* $f(.)$ correspondante soit proche de celle d'une autre donnée cible t (qui n'appartient pas à la même classe que x).

Parmi les attaques ciblées, les plus étudiées sont les attaques dites *backdoor* qui reposent sur l'association d'un *trigger* sur les données (par exemple, un petit *patch* dans un coin d'une image), à l'apprentissage et à l'inférence. C'est le *trigger* qui inocule le poison à l'apprentissage et qui permettra à un attaquant de modifier la prédiction d'un modèle pour toutes les données présentant ce *trigger*. Ce principe est illustré dans la figure 3.5 avec trois grands types de *trigger* :

— **Patch** : l'attaquant ajoute une perturbation saillante fixe. Les attaques *backdoor* par *patch* sont puissantes mais peuvent être plus facilement détectées.

— **Fonctionnel**. Contrairement au *patch*, un *trigger* fonctionnel cherche à être le plus imperceptible possible en le cachant grâce à des transformations adaptées à la nature des données et la tâche.

— **Sémantique**. Le *trigger* est un élément des données déjà présent dans la base d'apprentissage. L'objectif est d'utiliser un *trigger* qui paraît *naturel* et *probable* (e.g., les lunettes dans la figure 3.5).

8. On parle de modèles de "fondation" pour les modèles de très grande taille, entraînés sur des volumes considérables de données, qui peuvent être appliqués à un large éventail de tâches.

9. Dans certains état de l'art de référence, comme [29], on parle aussi d'*indiscriminate attacks*

Les deux derniers exemples de la figure 3.5 (*Bilevel* et *Feature Collision*) ne concernent pas des attaques ciblés de type *backdoor* mais des attaques par empoisonnement où le *poison* est optimisé pour éviter d'utiliser un *trigger* explicite et atteindre l'objectif de l'attaquant, qui peut être ciblé ou non-ciblé (altération globale de la performance du modèle).

Des travaux récents ont démontré des résultats d'empoisonnement sur des applications modernes, comme dans [32] qui utilisent des modèles entraînés sur des corpus à l'échelle du Web. Les auteurs montrent une attaque en empoisonnant uniquement 0.01% de LAION-400M[10] ou COYO-700M[11] en achetant des sites et domaines et en modifiant des ressources publiques (e.g., Wikipedia). Khaddaj *et al.* ont récemment montré que si les attaques *backdoor* pouvaient parfaitement reposer sur des *features indistinguables* par rapport à celles *normalement* apprises (comme dans le cas des *triggers* sémantiques), la *backdoor* reposait nécessairement sur la *feature* la plus forte. Ces nouveaux éléments théoriques permettent de définir des nouvelles défenses réactives par détection.

3.3.2 Attaquer l'intégrité à l'inférence

La menace la plus représentative des attaques contre l'intégrité à l'inférence concerne les **attaques par évasion** aussi connues sous le nom d'***adversarial examples*** [3, 4, 24] (cf. illustration figure 3.3). L'objectif est de tromper la prédiction d'un modèle en ajoutant une perturbation optimisée ϵ à la donnée d'entrée x, comme formalisé dans l'équation 3.1.

$$\arg\min_{\epsilon}\Big(M_W(x+\epsilon) = l\Big) \quad \text{avec } l \neq M_W(x) \text{ et } x_{adv} = x+\epsilon \in \mathcal{X} \tag{3.1}$$

L'objectif principal de l'attaquant est de tromper le modèle avec une donnée perturbée la plus proche possible de la donnée saine. Le cadre formel le plus utilisé est de contraindre la perturbation ϵ à travers une borne imposée sur la norme L_p. Les attaques les plus étudiées utilisent les normes L_2 et L_∞. Les attaques par évasion peuvent être non-ciblées ou ciblées, c'est-à-dire que l'adversaire peut spécifiquement chercher à tromper une prédiction vers un label cible. Les deux versions sont illustrées dans la figure 3.6.

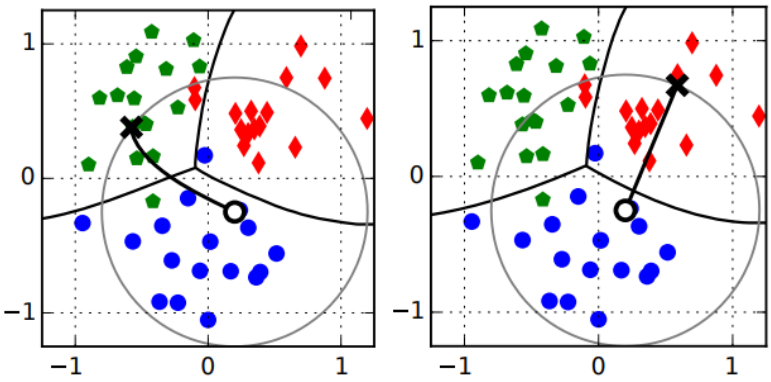


FIGURE 3.6 – Exemple d'une attaque par évasion *ciblée* (gauche) et *non-ciblée* (droite). L'instance attaquée est le rond bleu. Dans le cas de l'attaque ciblée, l'adversaire cherche à ce que la prédiction soit le label vert, plus "éloigné" que le label rouge qui sera choisi pour l'attaque non-ciblée. Le cercle indique la perturbation maximale autorisée selon la norme L_2 (budget de l'attaquant).

La littérature sur les attaques par évasion est conséquente avec de nombreuses attaques répondant à plusieurs types de modèles de menace, en boîte blanche ou boîte noire. Les attaques "boîte blanche" reposent sur l'optimisation de la perturbation en utilisant le gradient de la *loss* par rapport à l'entrée x, comme l'attaque FGSM qui propose de générer très simplement x_{adv} à travers l'équation 3.2 :

$$x_{adv} = x + \epsilon sign\Big(\nabla_x \mathcal{L}\big(M_W(x); y\big)\Big) \tag{3.2}$$

La référence pour les attaques boîte blanche est une version itérative de FGSM nommée *Projected Gradient Descent* (PGD) et illustrée dans la figure 3.7[12]. PGD repose d'une part sur une application itérative de FGSM en ne considérant qu'une petite portion du gradient à chaque

10. https://laion.ai/blog/laion-400-open-dataset/
11. https://github.com/kakaobrain/coyo-dataset
12. Illustration tirée de https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3

itération et, d'autre part, en réalisant plusieurs fois l'attaque avec un état initial échantillonné aléatoirement autour de x.

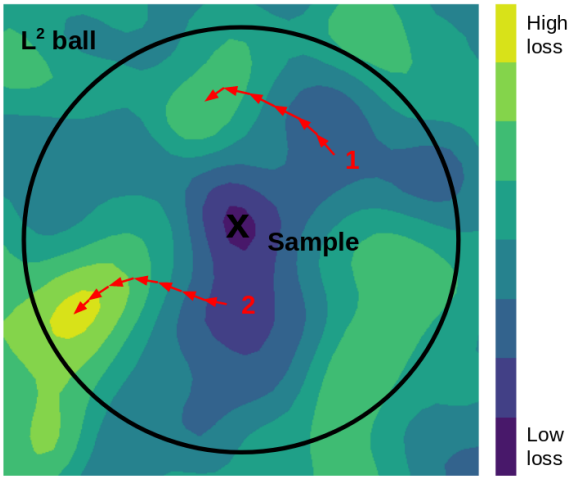


FIGURE 3.7 – Attaque PGD avec deux "redémarrages" aléatoires. Le second conduit à une erreur de prédiction plus importante.

Dans un contexte boîte noire, l'attaquant n'a pas une connaissance suffisante du modèle pour calculer directement le gradient $\nabla_x \mathcal{L}$. Une première solution repose sur la propriété de transférabilité [33] des *adversarial examples* : un attaquant va utiliser son propre modèle (boîte blanche) qui réalise la même tâche que le modèle victime pour construire son attaque (x_{adv}) puis va utiliser les x_{adv} sur le modèle victime. Une seconde stratégie est d'estimer les gradients en exploitant des paires entrée/sortie issues du modèle victime. Les sorties peuvent être les vecteurs de prédictions normalisées (i.e., après SoftMax) ou non normalisées (*logits*) (*score-based attacks*) ou uniquement le label prédit (*decision-based attacks*) [34, 35].

L'évaluation des *adversarial examples* est encore sujette à beaucoup d'études malgré plus d'une décennie de proposition d'attaques (et de défenses associées). L'état de l'art actuel correspond à l'utilisation d'AutoAttack [36] qui est un ensemble d'attaques boîte blanche *et* boîte noire (dans des versions ciblées et non-ciblées) connues comme étant les plus puissantes et adaptées pour comporter le moins de paramètres possibles. AutoAttack est utilisée comme attaque pour le benchmark de référence RobustBench [13].

3.3.3 Confidentialité

Les menaces conte la confidentialité concernent, d'une part, les attaques qui visent les données (d'apprentissage mais aussi d'inférence) et, d'autre part, le modèle en lui-même. Certains domaines (e.g., santé, défense) sont particulièrement sensibles quant à la confidentialité des données d'apprentissage qui peuvent être privées ou classifiées par exemple. Quant aux modèles, leur *design*, développement et apprentissage peuvent être le résultat d'une expertise qui représente un avantage concurrentiel majeur et la confidentialité de leurs paramètres et hyper-paramètres doit être préservée.

3.3.3.1 Confidentialité des données

Extraction des données d'entraînement. Les modèles DNN ont tendance à encapsuler (i.e., mémoriser) des informations sur les données d'entraînement qui peuvent fuir par l'exploitation de leurs paramètres ou de leurs prédictions. Carlini *et al.* [19] soulignent des *mémorisations non intentionnelles* critiques pour les modèles de langage entraînés sur de grands ensembles de données textuelles. Ce problème peut être (en partie) résolu par la *differential privacy* (DP) [37, 38], une technique qui ajoute du bruit pendant l'apprentissage (i.e., directement dans l'équation 2.22 d'optimisation des paramètres) et permet de limiter la quantité d'information des données qu'il est possible d'extraire du modèle.

Inversion de modèle. Dans le cas d'une attaque par *inversion de modèle* un adversaire va reconstruire les données d'entraînement ou inférer des informations sensibles contenues dans ces don-

13. https://robustbench.github.io/

nées. Les attaques par inversion de modèle prennent généralement tout leur sens dans le contexte d'applications traitant des données avec une partie *non sensibles* (accessibles par l'attaquant) et une partie *sensible* (inaccessibles). Ce type d'attaque est assez vaste et peut être classiquement catégorisé en deux types : les attaques par *inférence*, et celles par *reconstruction* [39]. Dans le premier cas, l'attaque consiste à déduire des attributs sensibles de manière exacte ou approximative, ou à estimer des propriétés liées aux échantillons de données d'entraînement. Ainsi, en fonction de l'objectif de l'attaque, nous avons des attaques par *inférence d'attributs*, par *inférence approximative d'attributs* et par *inférence de propriété*. Dans ce dernier cas, l'objectif est d'inférer des propriétés qui ne sont pas explicitement définies comme des attributs dans l'ensemble d'entraînement (e.g., savoir si une personne porte des lunettes dans une image). Si l'objectif est d'inférer exactement l'intégralité d'une donnée d'entraînement, on tombe alors dans la menace d'*extraction de données* discutée ci-dessus. La seconde catégorie de *reconstruction* a été majoritairement démontrée pour des modèles de *computer vision*. L'objectif est d'exploiter les sorties d'un modèle pour reconstruire une image ou une image représentative.

Membership Inference. L'objectif d'un adversaire réalisant une *Membership Inference Attack* (MIA) Attaque d'Inférence d'Appartenance (notée MIA ci-après) contre un modèle M est de savoir si un exemple x a été ou non utilisé lors de l'entraînement [40]. La MIA est l'attaque de référence pour évaluer la robustesse d'un modèle par rapport à des menaces sur la confidentialité des données d'apprentissage. Une référence est LiRA (pour *Likelihood Ratio Attack*) proposée par Carlini *et al.* [41] qui critiquent la façon dont les MIA sont évaluées à travers une *accuracy* moyenne. Ils proposent une évaluation à partir du taux de vrais-positifs calculé pour un faible taux de faux-positifs (i.e., de baser l'évaluation sur une courbe ROC, traçant le taux de vrais positifs TVP par rapport au taux de faux positifs). L'idée de LiRA est de considérer une MIA sur une donnée (x, y) (i.e., est-ce que cette donnée était dans (X^{train}, Y^{train})?) comme un test statistique sur deux distributions de modèles : ceux qui ont été entraînés sur la donnée (x, y) et ceux qui n'ont pas utilisé la donnée. Comme il n'est pas possible de calculer parfaitement de telles distributions, ils proposent –à partir du modèle cible M donné – d'entraîner plusieurs modèles avec (ensemble "IN") et sans (ensemble "OUT") la donnée (x, y) puis de modéliser les scores de confiance de chaque ensemble "IN" et "OUT" avec deux gaussiennes. LiRA revient donc à un simple test du rapport de vraisemblance (équation 3.3) du score de confiance réellement observé par M ($conf_{obs}$) par rapport aux deux gaussiennes :

$$\frac{p\Big(conf_{obs}|\mathcal{N}(\mu_{in}, \sigma_{in})\Big)}{p\Big(conf_{obs}|\mathcal{N}(\mu_{out}, \sigma_{out})\Big)} \tag{3.3}$$

3.3.3.2 Extraction de modèle

Dans [42], Jagielski *et al.* apportent une première taxonomie des objectifs d'une attaque par extraction de modèle. Un premier objectif est de cloner le modèle cible dans un objectif de vol de propriété intellectuelle. Ce scénario d'attaque est dénommé "*fidélité*". L'attaquant va chercher à construire un modèle substitut le plus proche possible du modèle cible pour copier parfaitement ses performances (i.e., autant les bonnes prédictions que les mauvaises). Un second objectif est de construire un modèle qui va – à moindre coût – copier les capacités du modèle cible pour, par la suite, dépasser ses performances via un apprentissage supplémentaire ou du *fine-tuning*. Ce scénario est appelé "*performance*" et diffère du précédent par le fait que l'extraction de l'architecture et des valeurs des paramètres du modèle cible n'est pas une nécessité pour l'attaquant qui ne cherche qu'à voler ses performances (et par là, l'expertise et les données du concepteur du modèle cible). Enfin, un dernier objectif correspond à la création d'un modèle substitut permettant à l'adversaire de construire des attaques en boîte blanche plus performantes qu'il pourra par la suite transférer sur le modèle cible (e.g., des *adversarial examples*).

Par la suite, nous nous concentrons exclusivement sur le scénario de fidélité qui sera au cœur du chapitre 5. Pour cela, nous reprenons notre formalisme avec un modèle cible de réseau de neurones supervisé M_W, avec W ses paramètres. Le domaine d'entrée est $\mathcal{X}$ et M est entraîné grâce à un jeu de données d'entraînement (X^{train}, Y^{train}), où $Y^{train} \subset \mathbb{R}^K$ K étant le nombre de labels. Pour une entrée $x \in \mathcal{X}$, la prédiction de sortie est $M_W(x) \in \mathbb{R}^K$, et l'étiquette prédite est $\hat{y} = \text{argmax}(M_W(x))$. Dans un scénario de *fidélité*, l'objectif d'un adversaire est de concevoir un modèle M'_Θ qui imite M_W aussi parfaitement que possible, en fonction de ses connaissances et de ses capacités. Notez que M' vise à fournir les mêmes prédictions que M, y compris les éventuelles erreurs de M_W. Évidemment, cet objectif dépend de la façon dont on mesure la fi-

délité de M' par rapport à M. Une approche typique [42] consiste à mesurer l'accord entre les deux modèles au niveau des étiquettes, c'est-à-dire qu'un objectif classique est de s'assurer que $\text{argmax}(M'_\Theta(x)) = \text{argmax}(M_W(x))$ pour chaque x échantillonné à partir d'une distribution cible sur $\mathcal{X}$. Une extraction plus complexe et optimale, appelée *Extraction Fonctionnellement Équivalente*, vise à atteindre $M'(x) = M(x), \forall x \in \mathcal{X}$. Notez que l'attaque la plus puissante possible, aboutissant à un modèle substitut avec exactement la même architecture et les mêmes paramètres ($W = \Theta$), est irréalisable en exploitant uniquement des paires entrée/sortie issues du modèle victime [42].

La majorité des approches proposées dans la littérature considère le problème d'extraction uniquement à travers celle des valeurs des paramètres, ce qui signifie que l'attaquant possède déjà les informations relatives à l'architecture de M_W où qu'il a utilisé d'autres attaques pour les avoir. Nous montrons dans le chapitre 5 (section 5.1) que les analyses *side-channel* peuvent s'avérer extrêmement puissantes pour l'extraction d'architecture d'un réseau de neurones embarqué.

La première famille de méthodes repose sur des approches d'*active learning*, c'est-à-dire de l'entraînement de M'_Θ à partir d'une très grande quantité de paires entrées/sorties, collectées à partir du modèle victime, pour construire un jeu de données d'entraînement substitut efficace. On se situe donc dans un contexte d'attaque (réaliste) où l'adversaire à un accès minimal (sinon nul) aux données d'apprentissage du modèle cible. Dans ce cas, ces approches nécessitent énormément de requêtes et sont confrontées à des difficultés d'apprentissage (e.g., *overfitting*) qui conduisent à la convergence d'un modèle substitut très peu *fidèle* au modèle cible.

Une deuxième approche repose sur une récupération mathématique, là aussi en exploitant des paires entrée/sortie choisies associées à des propriétés basées sur les gradients du modèle qui exploitent les particularités des fonctions d'activation, plus particulièrement ReLU (dont la dérivée seconde est nulle partout sauf au point critique $x = 0$) comme dans [43]. Les résultats expérimentaux de [43] montrent une extraction complète des paramètres d'un MLP à 100,000 paramètres (avec une seule couche cachée) en $2^{21.5}$ requêtes, avec une erreur d'extraction dans le pire cas de 2^{-25}. Les limitations de ces approches résident dans leur complexité pour des modèles plus profonds et sa stricte dépendance à ReLU. Néanmoins, des améliorations significatives ont été proposées dans [44] et [45].

3.3.4 Disponibilité

Sur le concept de disponibilité. Les attaques liées à la disponibilité visent à altérer : l'*accès* au système, la *qualité* du système (par exemple, la confiance d'une prédiction), la *performance* du système (contraintes mémoires, vitesse d'inférence, etc.). De toute évidence, l'intégrité et la disponibilité sont deux concepts étroitement liés. En effet, une attaque sur l'intégrité qui altère gravement la performance moyenne d'un modèle peut rendre ce dernier inutilisable. Par conséquent, les menaces basées sur l'intégrité sont des vecteurs réalistes pour attaquer la disponibilité dès lors que la cible est la qualité du système.

Les menaces liées à la disponibilité incluent aussi toutes les attaques classiques qui visent à affaiblir une infrastructure informatique ou de communication, comme les attaques par déni de service (Denial-of-Service – DoS) ou par déni de service distribué (Distributed DoS – DDoS). Par exemple, une architecture de ML peut ne pas être dimensionnée pour traiter simultanément un grand nombre d'inférences. Ces menaces ne sont pas directement liées aux modèles de ML et, par conséquent, sortent du cadre de ce manuscrit.

Sponge examples. Une des premières attaques visant directement la disponibilité a été la génération de *sponge examples*, démontrée dans [46]. L'idée principale vient de l'observation que, pour la majorité des modèles de réseaux de neurones profonds, l'inférence ne fait réellement intervenir qu'une faible quantité de connexions, plus particulièrement pour les modèles basés sur la fonction d'activation ReLU qui *éteint* certains neurones. Aussi, Shumailov *et al*. affirment que cette *parcimonie* a une influence importante sur le temps d'inférence et que, si toutes les connexions étaient "activées", cela entraînerait une augmentation significative du nombre de multiplications-accumulations et donc du temps d'inférence. Un *sponge example* est donc une entrée x^s à laquelle on ajoute une perturbation optimisée pour maximiser toutes les activations d'un modèle :

$$x^s = \underset{x}{\text{argmin}} - \sum_{a_i \in \mathcal{A}} \|a_l\|_2$$

où $\mathcal{A}$ désigne l'ensemble de toutes les valeurs d'activation et a_i représente l'activation de la couche l. Notons que, dans un scénario *boîte noire*, les activations a_l n'étant pas connues de l'atta-

quant, les auteurs proposent d'utiliser un algorithme génétique pour construire de façon itérative un *sponge example* à partir d'échantillons pris aléatoirement dans l'espace d'entrée $\mathcal{X}$. L'efficacité des *sponge examples* dépend fortement de la nature des modèles et des tâches. Les résultats sont ainsi concluants sur des benchmarks de NLP, avec un facteur d'au moins 3× sur la vitesse d'inférence, mais sont plus discutables pour des modèles classiques de classification d'images. Une limitation de l'attaque est que l'optimisation de la norme L_2 des activations n'entraîne pas nécessairement une augmentation du nombre de neurones qui vont effectivement être activés (mais seulement la valeur d'activations déjà présentes).

Le principe des *sponge examples* a été généralisé à l'apprentissage, dans un contexte d'empoisonnement non-ciblé dans [47]. L'objectif est d'empoisonner un modèle à travers les données d'entraînement pour accroître la "consommation énergétique" de l'inférence tout en préservant sa performance moyenne. Le terme "énergie" ne correspond pas ici à la norme L_2 des activations comme dans les *sponge examples* mais au nombre de neurones activés. L'attaque permet une augmentation de l'énergie supérieure à 100% dans la majorité des benchmarks utilisés (CIFAR-10, GTSRB, CelebA).

Perturber le processus d'apprentissage. Certains travaux liés aux attaques par empoisonnement démontrent qu'un attaquant est capable d'altérer fortement le processus d'apprentissage en modifiant certaines étapes particulières, en particulier l'initialisation des poids [48] et le processus d'échantillonnage des *batchs* (*batching attacks*) [49]. Avec de telles attaques, le temps d'apprentissage peut être significativement augmenté et le modèle final converge vers un faible niveau de performance. Par exemple, une simple permutation de la valeur initiale des paramètres permet de faire converger un modèle DNN sur Fashion-MNIST à 50% d'*accuracy* alors qu'il atteint 90% sans attaque [48]. De même, les attaques de type *batching* peuvent considérablement altérer l'apprentissage même si elles sont appliquées sur une seule *epoch* [49].

3.4 Spécificités de l'IA embarquée

Les menaces présentées dans la section précédente ne concernent qu'une surface d'attaque dite "algorithmique" et qui représente la majorité de l'état de l'art actuel en sécurité de l'IA. Cependant, depuis quelques années, la communauté de la sécurité matérielle s'est emparée du sujet, poussée par le fort développement de plateformes dédiées à l'IA embarquée. Dans cette section, nous rappelons quelques éléments caractéristiques de l'IA embarquée (plus particulièrement sur microcontrôleurs) avec les techniques standards pour l'optimisation et la compression de modèles. Pour la quantification, nous présentons une étude sur son impact sur les *adversarial examples*. Puis, dans les sections 3.5 et 3.6, nous discutons de l'état de l'art sur l'exploitation des deux principales attaques physiques, l'analyse *side-channel* et l'injection de fautes. Pour cette dernière, nous décrivons des premières expériences sur l'influence d'un saut d'instruction dans l'inférence d'un CNN sur microcontrôleur (Cortex-M4).

3.4.1 Complexité des modèles et inférence sur MCU

Les modèles de réseaux de neurones profonds de l'état de l'art sont généralement entraînés sur des plateformes à base de GPU, capables de paralléliser de multiples calculs en précision `float32` (ou 64). Les principales opérations sont les suivantes :

- Opérations de multiplication-accumulation (*MAC*) : l'opération linéaire de base entre les entrées x et les paramètres w (et biais) d'une couche ($acc \leftarrow acc + w \times x$). Le nombre de MAC est un indicateur traditionnel de la complexité des modèles.
- Application d'une activation non-linéaire : des fonctions basiques comme *ReLU* ($ReLU(x) = max(0, x)$) ou plus complexes comme *Softmax* (exponentielle, divisions).
- Autres opérations non linéaires : plus particulièrement les statistiques de base – mais coûteuses – comme la *moyenne* et la *variance* au niveau d'un *batch* pour la couche de normalisation (*batch normalization*) et les opérations de *pooling* (généralement via les opérateurs *mean* et *max*).

Exemple sur microcontrôleur. Considérons un STM32H7 (Cortex M7), un microcontrôleur 32 bits typique adapté à l'IA embarquée. Cette plateforme dispose de 2 Mo de mémoire Flash et de 512 Ko de mémoire SRAM. L'utilisation de la mémoire pour l'inférence d'un DNN se répartit comme suit :

— Mémoire Flash : programme d'inférence, architecture du modèle et paramètres (poids, filtres, biais).
— Mémoire SRAM : entrées fournies par l'utilisateur, calculs intermédiaires comme les sorties des fonctions d'activation.

Un *benchmark* classique de l'état de l'art est ImageNet (Cf. section 2.3) avec des images disponibles en plusieurs résolutions et pour lequel les modèles CNN de types MobileNet offrent un bon compromis entre la taille (i.e., nombre de paramètres), latence et précision. Ces modèles possèdent un facteur de compression (α) qui influence leur profondeur (nombre de couches) [50,51]. Capotondi *et al.*[14] illustrent la compatibilité de différents modèles MobileNet par rapport aux contraintes mémoires d'un STM32H7[15]. La figure 3.8 montre ainsi qu'avec des images de résolution de 160×160 pixels et un MobileNet avec un paramètre de compression $\alpha = 0.25$ (470K paramètres), l'inférence du modèle (représentant 21 millions de MAC) est possible avec une quantification des paramètres sous 8 bits et atteint une *accuracy* supérieure à 0.4 (*accuracy* top-1 et 0.68 en top-5). Typiquement, la majorité des architectures MobileNet ne respectent pas les contraintes de mémoire du STM32H7 avec des paramètres en `float32`. Avec un MobileNet-160-0.25, l'appareil doit disposer d'au moins $0.47 * 4 = 1.88$ Mo de mémoire Flash uniquement pour stocker les paramètres du modèle. Ce problème soulève la question de la nécessité des paramètres en pleine précision pour les modèles DNN. Par exemple, la Figure 3.8 (droite) montre qu'avec une représentation des paramètres en entier 8 bits, sept architectures MobileNet respectent les contraintes de mémoire (représentées par le rectangle rouge).

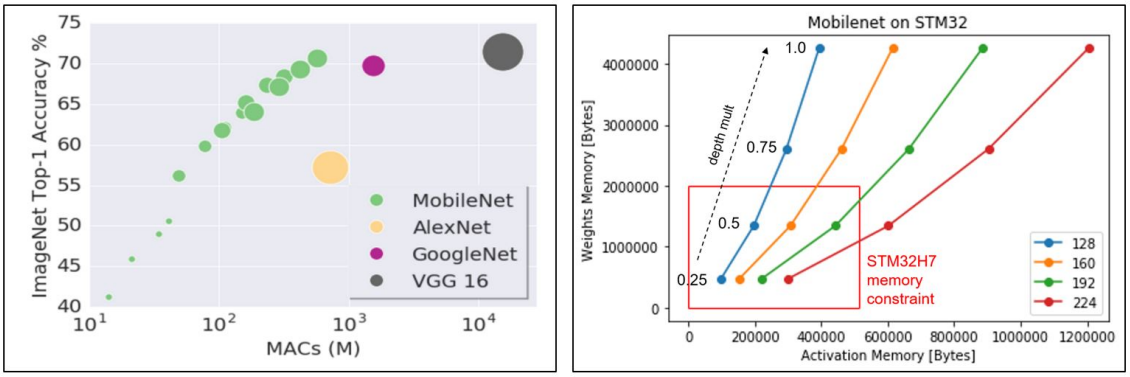


Figure 3.8 – (Gauche) Complexité MAC de plusieurs modèles d'état de l'art. (Droite) Consommation mémoire Flash et SRAM pour différents modèles MobileNet (v1). Les besoins en mémoire pour un STM32H7 sont représentés par le rectangle rouge (paramètres pour la Flash et calculs d'activation pour la SRAM). Source : Capotondi *et al.* [14].

Une part de plus en plus importante de la littérature sur le *Deep Learning* est actuellement consacrée à la *compression des modèles*. La tendance allant à des tailles de modèle de plus en plus importantes (Cf. les architectures Transformer), l'état de l'art montre que dans la majorité des cas, il est possible d'atteindre des niveaux similaires de performances en adaptant la précisions de représentation des paramètres (par exemple le format FP8 pour les modèles de langue [52]) mais aussi des calculs intermédiaires (activations) ou en supprimant certains paramètres considérés comme inutiles. Nous détaillons les principales techniques ci-dessous.

3.4.2 Optimisation et compression de modèles

Pour répondre aux contraintes parfois fortes de certaines plateformes matérielles (mémoire, énergie, latence), comme les microcontrôleurs, de nombreuses stratégies ont été proposées pour permettre et faciliter le déploiement de modèles de l'état de l'art sur une grande variété de systèmes. Nous détaillons ci-dessous les principales techniques.

3.4.2.1 Architectures optimisées et distillation

Les principales architectures de DNN ont majoritairement été définies pour répondre à des questions de performance par rapport à des tâches spécifiques (e.g., classification d'images, modèle de langue). Un exemple classique est l'architecture ResNet [53] permettant d'accroître la

14. https://github.com/EEESlab/mobilenet_v1_stm32_cmsis_nn
15. https://www.st.com/en/evaluation-tools/nucleo-h743zi.html

profondeur des modèles CNN. Néanmoins, certaines architectures comme MobileNet [50] ou EfficientNet [54] ont spécifiquement été créées pour optimiser la complexité des modèles (i.e., réduire le nombre de paramètre) et améliorer la vitesse d'inférence. L'optimisation peut aussi être réalisée à l'échelle des couches comme les couches de convolution. Dans [55], nous analysons les performances de plusieurs primitives de convolution (*Grouped convolution* [56, 57], *Depthwise separable convolution* [58], *Shift convolution* [59] et *Add convolution* [60]). Une autre approche appelée NAS (pour *Neural Architecture Search*) propose de chercher la meilleure architecture à partir d'un ensemble de types de couche (e.g., convolution, dense) et d'hyper-paramètres (e.g., nombre de neurones, nombres et taille de filtres) [61]. Un travail de référence est la série des MCUNet du MIT [16] [62–64] qui combine deux optimisations conjointes : (1) l'architecture du modèle avec une approche de type NAS et (2) la gestion de la mémoire pour accélérer l'inférence.

Enfin, les techniques par *distillation* ont aussi été proposées pour le déploiement sur des cibles contraintes. L'idée principale est d'entraîner un modèle plus petit (répondant aux contraintes de la cible matérielle) pour transférer les connaissances du modèle cible. Dans un contexte de *distillation*, le modèle cible est appelé le "professeur" (*teacher*) et le modèle déployé est l'"élève" (*student*) [65, 66]. Le modèle élève acquiert les connaissances du modèle professeur en essayant de reproduire les représentations intermédiaires (*features map*) pendant un processus d'entraînement proche du *transfer learning*.

3.4.2.2 Pruning

Le *pruning* consiste à supprimer les paramètres d'un modèle qui n'ont pas d'importance pour la création de la prédiction. Le *pruning non structuré* [67, 68] supprime les paramètres indépendamment les uns des autres sans prendre en compte de leur appartenance à une structure neuronale (e.g., supprimer un seul paramètre d'un filtre de convolution). Ces approches conduisent à des représentations creuses qui nécessitent des librairies de calculs spécifiques (e.g., cuSPARSE pour les GPUs NVIDIA). Le *pruning* structuré [69] sélectionne et supprime les paramètres à l'échelle des structures neuronales, typiquement tout un filtre pour les CNNs et un neurone pour les couches denses. La réduction du nombre de paramètres peut être significative tout comme le temps d'inférence. Notons que le *pruning* peut aussi être appliqué dans des scénarios de *transfert learning* comme l'adaptation de domaines comme nous l'avons montré dans [70].

3.4.2.3 Quantification

Les techniques de quantification peuvent être réalisées pendant (*training-aware)* ou après (*post-training*) l'entraînement en transposant directement les paramètres à une précision inférieure, par exemple en passant de valeur en précision flottante 32 bits à des entiers signés sous 8 bits.

Bien que très simple à implémenter, les approches post-entraînement souffrent de baisses de performance pour des quantifications inférieures à 8 bits. En revanche, la quantification pendant l'apprentissage permet de conserver un niveau de performance même pour des fortes compressions (e.g., 1, 2 ou 4 bits). Courbariaux *et al.* [71] ont proposé une méthode d'entraînement pour des DNN quantifiés jusqu'à 1 bit (réseaux binarisés). Pendant la *forward pass*, $W_b = sign(W)$ remplace les poids W, et pendant la *backward-pass* de la rétropropagation des gradients, pour contrer les problèmes de dérivation liés à l'opérateur $sign(.)$, ils proposent d'approximer le gradient avec l'estimateur *Straight-Through Estimator* (STE) (équation 3.4.2.3, partie grisée) :

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial w_b}\frac{\partial w_b}{\partial w} \approx \frac{\partial \mathcal{L}}{\partial w}\Big|_{w=w_b} \mathbb{1}_{|w|\leq 1}$$

À la suite de ce travail de référence, plusieurs efforts [72, 73] ont étendu la quantification aux paramètres *et* aux activations mais aussi en proposant des précisions mixtes [74] qui permettent de s'adapter à des architectures plus complexes (voir par exemple le *leaderboard* du challenge "*Quantization on ImageNet*" [17].

La quantification à l'apprentissage permet de conserver un niveau de performance remarquable surtout quand la quantification ne concerne que les paramètres. Pour illustrer l'intérêt de cette approche, dans [15], nous prenons un CNN avec la même architecture que dans Courbariaux *et al.* [72] avec environ 10 millions de paramètres (3 blocs convolutionnels avec respectivement 128, 256 et 512 filtres du MaxPooling et de la batch normalisation suivis par trois couches

16. https://github.com/mit-han-lab/mcunet
17. https://paperswithcode.com/sota/quantization-on-imagenet

denses avec 1024, 1024 et 10 neurones). Nous obtenons les résultats présentés dans le tableau 3.1. On remarque que, quand seuls les paramètres sont quantifiés, la performance des modèles quantifiés sont stables par rapport au modèle en précision `float32`, même pour des niveaux élevés de quantification (1 et 2 bits). Ce résultat s'explique aussi par le fait que le modèle est sensiblement surdimensionné pour les tâches à accomplir : à l'apprentissage, la contrainte supplémentaire apportée par la quantification est largement compensée par la capacité du modèle.

TABLE 3.1 – *Accuracy* (sur les ensembles de test) avec une quantification pendant l'apprentissage.

	CIFAR10 (0.89)				SVHN (0.96)			
Quantification (bits)	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Paramètres + activations	0.79	0.87	0.88	0.88	0.89	0.95	0.95	0.95
Paramètres uniquement	0.88	0.88	0.88	0.88	0.96	0.95	0.96	0.95

Dans la littérature, la quantification est souvent simulée, comme pour l'étude des attaques sur les paramètres (Cf. Chapitre 4). Plus particulièrement, [75] et [6] utilisent une quantification post-apprentissage des paramètres sur 8 bits en utilisant l'équation 3.4 suivante :

$$w^{(q)} = \left\lfloor \frac{HT\Big(w, -max(|W_l|), max(|W_l|)\Big)}{p_l^{(q)}} \right\rceil \qquad p_l^{(q)} = \frac{max\Big(|W_l|\Big)}{2^7 - 1} \tag{3.4}$$

p_l^q est le pas de quantification associé à la couche l, w est un paramètre de W_l, le tenseur des paramètres de la couche l, et la fonction HardTanh (notée HT) définie ci-dessous (Equation 3.5).

$$\mathrm{HT}(x, min\_v, max\_v) = \begin{cases} \text{max\_v} & \text{si } x > \text{max\_v} \\ \text{min\_v} & \text{si } x < \text{min\_vl} \\ x & \text{sinon} \end{cases} \tag{3.5}$$

Un autre schéma classique de quantification sous 8 bits, utilisé par exemple dans les librairies de déploiement sur MCU CMSIS-NN (ARM) [76] ou NNoM [77], utilise un pas en puissance de deux qui permet de n'avoir aucune division mais uniquement des additions d'entiers et des décalages de bits :

$$d = \left\lceil log_2\Big(max(|W_l|)\Big) \right\rceil \qquad w^{(q)} = \Big\lfloor w \cdot 2^{7-d} \Big\rfloor \tag{3.6}$$

Notons que la quantification est aujourd'hui une technique extrêmement utilisée pour l'IA embarquée mais aussi pour les très grands modèles de fondations, dont les LLMs. En effet, la taille de ces gigantesques modèles pose de sérieux défis quant à leur déploiement même pour des plateformes que l'on n'a pas l'habitude de considérer comme "contraintes" (e.g., ordinateur personnel, *smartphone* haut de gamme). Le sujet de la quantification des modèles est donc aujourd'hui particulièrement actif. PyTorch et NVIDIA recommandent un passage au format FP16 (en réalité certains calculs intermédiaires restent en FP32) pour les modèles sous PyTorch afin de gagner (au moins) un facteur 2 à l'apprentissage[18] et l'utilisation des formats FP8 a permis des progrès importants pour l'optimisation des LLM [52] (e.g., DeepSeek-V3 [78]).

Néanmoins, l'influence de la quantification (à l'inférence et à l'apprentissage) sur la sécurité des modèles soulève de nombreuses questions dont certaines encore ouvertes notamment pour les modèles modernes (LLM, *foundation models*, *diffusion models*). Dans la section suivante, nous illustrons l'influence de la quantification sur la sécurité en analysant la robustesse des modèles aux attaques par évasion (*adversarial examples*).

3.4.2.4 Quantification et robustesse des modèles

Dans [15], nous montrons que la quantification peut (à tort) être considérée comme une défense contre les *adversarial examples*. En effet, dans [79], Galloway *et al.* affirment que les modèles entraînés avec des paramètres et des activations sur 1 bit (quantification binaire $\{-1, 1\}$)

18. https://pytorch.org/blog/accelerating-training-on-nvidia-gpus-with-pytorch-automatic-mixed-precision/

présentent une robustesse intéressante face aux attaques par évasion. Néanmoins, leurs études portent uniquement sur MNIST et sur une seule attaque boîte blanche (FGSM) qui n'est pas l'attaque la plus puissante. De plus, les gradients obtenus via le STE peuvent ne pas être représentatifs des gradients réels (comme démontré dans [80]), ce qui soulève des interrogations quant à la réelle pertinence des attaques basées sur les gradients contre les modèles quantifiés. En utilisant plusieurs attaques de type boîte blanche (FGSM, BIM[19], CWl_2) ainsi que des attaques boîte noire (*gradient-free*, ZOO, SPSA), nous montrons que la quantification amplifie le phénomène de *gradient masking*, un phénomène connu pour perturber les attaques basées sur les gradients mais sans apporter une réelle robustesse aux modèles. Nous constatons que ce phénomène est surtout présent pour les modèles dont les activations sont aussi quantifiées. Les modèles sujets aux *gradient masking* restent extrêmement sensibles à des attaques s'appuyant sur des approximations de gradient, ou des attaques boîte noire ne reposant pas sur les gradients, comme SPSA. Ainsi, l'attaque SPSA est plus performante que l'attaque FGSM (cf. équation 3.2) ou BIM (i.e., FGSM itérative) pour des modèles entièrement binarisés (notés w_1a_1 dans les tableaux de la figure 3.9) pour lesquels l'estimation des gradients est complexe : nous obtenons en effet dans [15] un taux de succès des *adversarial examples* de 84% pour SPSA et seulement 44% pour BIM. Évaluer de tels modèles uniquement avec des attaques boîte blanche de type FGSM apporte un faux sentiment de sécurité car la robustesse apparente des modèles ne repose en réalité que sur une faiblesse de l'attaque.

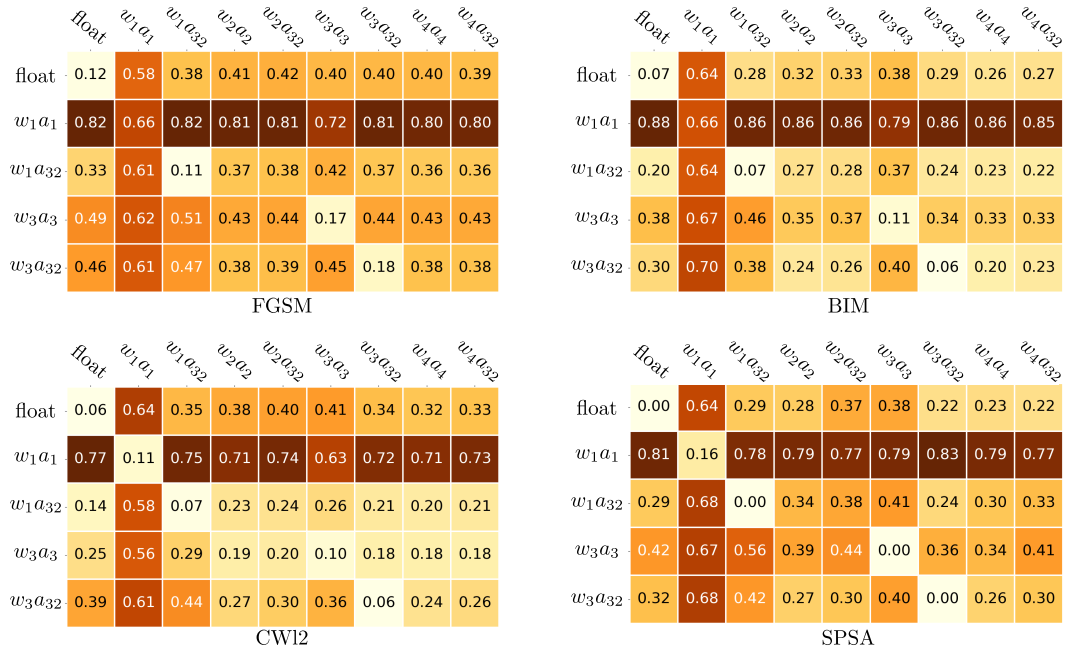


Figure 3.9 – Résultats de transférabilité adverse pour CIFAR-10. Les lignes correspondent aux modèles sources et les colonnes aux modèles cibles. Les valeurs correspondent à l'*adversarial accuracy*, plus la valeur est basse, plus la transférabilité est élevée. w_ia_j désigne un modèle avec une quantification des poids sur i bits et une quantification des activations sur j bits.

Enfin, nous mettons en évidence de faibles capacités de transférabilité entre des modèles classiques et des modèles quantifiés, ainsi qu'entre des modèles quantifiés avec différents niveaux de précision. Le résultat des expériences de transférabilité (pour CIFAR-10) est présenté dans les tableaux de la figure 3.9. Ce résultat est intéressant pour la conception de protections contre les attaques en boîte noire qui fonctionneraient en utilisant un ensemble de modèles avec des quantifications différentes. Une première explication de ce phénomène est l'effet direct de la quantification entre l'inférence d'une donnée et sa version *adverse*. Des valeurs d'activations peuvent se retrouver dans le même intervalle de quantification ou, au contraire, projetées dans des valeurs quantifiées différentes. La quantification peut donc annuler ou amplifier la propagation de la perturbation dans le modèle. Une autre explication est le *désalignement* des gradients induit par la quantification qui peut être particulièrement prononcé comme le montre le tableau de la figure 3.10 qui présente la similarité cosinus entre les gradients de deux modèles (0 signifiant des gradients orthogonaux). En effet, nous observons que les valeurs de similarité cosinus pour les

19. BIM est une version itérative de FGSM, comparable à PGD

gradients entre les modèles originaux (32 bits) et les modèles quantifiés, ainsi qu'entre des modèles quantifiés avec différents niveaux de précision, sont relativement proches de 0. Cela indique des directions de gradients presque orthogonales, en particulier entre les modèles entièrement binarisés et les autres modèles. Cela concorde avec les résultats présentés dans la Fig. 3.9, où les capacités de transférabilité pour les attaques considérées sont les plus faibles lorsque des modèles entièrement binarisés sont impliqués.

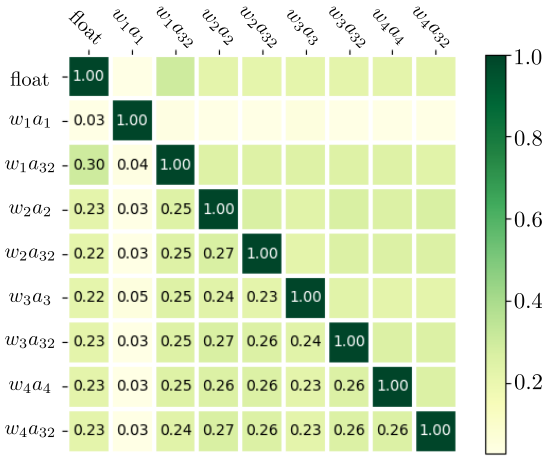


FIGURE 3.10 – Valeurs de similarité cosinus entre le gradient de la *loss* par rapport à l'entrée, pour les modèles en pleine précision et les modèles quantifiés. $w_i a_j$ désigne un modèle avec une quantification des paramètres sur i bits et une quantification des activations sur j bits.

Un résultat important de ces expériences est que les caractéristiques des modèles embarqués, particulièrement celles induites par les approches de quantification (des poids ou des activations), doivent être convenablement prises en compte afin de concevoir des schémas de protection adaptés et efficaces.

3.4.3 Prise en compte des attaques physiques

Une surface d'attaque complexe. Lorsque des modèles de réseaux neuronaux embarqués sont physiquement accessibles par un attaquant, la surface d'attaque inclut d'une part les menaces algorithmiques et théoriques, décrites dans les sections précédentes, *et* tous les vecteurs d'attaque qui reposent sur l'exploitation des failles logicielles et/ou matérielles de l'inférence embarquée. Les premières menaces considèrent le modèle comme une abstraction algorithmique avec un attaquant exploitant des requêtes auprès d'une API [20]. Le second type d'attaques exploite des observations physiques (comme les émissions électromagnétiques ou le temps d'exécution) ou des altérations par des vecteurs physiques (par exemple, l'injection de fautes par un laser directement dans la mémoire d'un microcontrôleur). La complémentarité de ces deux "mondes" d'attaque augmente significativement la complexité et la criticité des menaces contre l'IA embarquée.

Attaques physiques. Il faut distinguer les attaques physiques par **observation** de celles par **perturbation**. Les attaques par observation, plus particulièrement les analyses par canaux auxiliaires ou *side-channel* (SCA, pour *Side-Channel Analysis*) concernent les analyses qui n'interagissent pas physiquement avec le matériel mais exploitent des signaux et observations qui proviennent de la cible matérielle pendant son fonctionnement. L'objectif est de relier ces observations avec des connaissances du système (logicielles ou matérielles) afin de déduire des informations secrètes du système. Parmi les *signaux* les plus utilisées, on trouve les émanations électromagnétiques (EM, *EM Analysis*), captées par une sonde EM dédiée, les mesures de consommation (*Power Analysis*) et les mesures de temps de traitement (*Timing Analysis*). Une littérature très abondante concerne l'utilisation du SCA contre des modules cryptographiques (e.g., AES) [81].

Pour les attaques physiques par perturbation, notons tout d'abord le *probing* (ou *microprobing* qui consiste à *sonder* par des aiguilles le circuit de câblage interne d'un circuit. Cette technique est coûteuse et demande une forte expertise. Elle n'est pas traitée dans ce manuscrit. Le second type d'attaques est l'injection de faute par l'intermédiaire d'un moyen d'injection bien particulier qui va entraîner différents types de faute en fonction de la région visée dans le circuit (mémoire,

20. Dans la littérature, toutes ces attaques sont parfois nommées "*API-based attacks*".

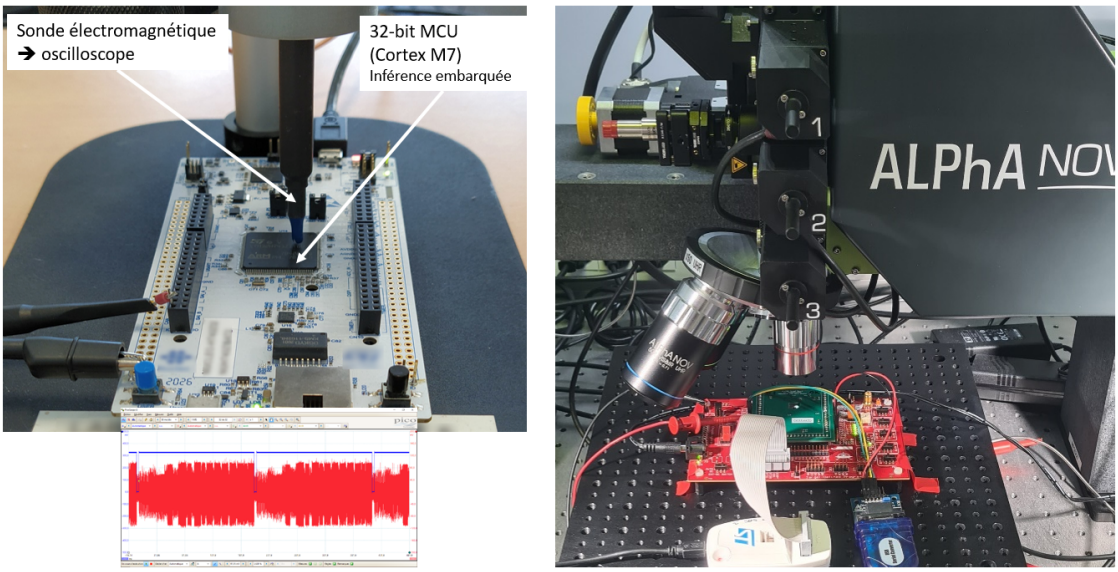


FIGURE 3.11 – (Gauche) Analyse *side-channel* avec une sonde électromagnétique capturant la *trace* de deux inférences sur un microcontrôleur (Cortex M7). (Droite) Injection de fautes laser sur un microcontrôleur (Cortex M3).

logique, bus...). Parmi les types d'injection les plus utilisés dans la littérature et les laboratoires de caractérisation sécuritaire, notons l'injection laser, l'injection électromagnétique, le *glitch* de tension ou d'horloge. Ces moyens d'injection possèdent des caractéristiques très différentes qui sont résumés dans le tableau 3.2 issu d'un état de l' art de référence dans le domaine de la cryptographie [82]. Les attaques par injection de fautes nécessitent généralement un accès physique à la cible [83], mais certaines variantes peuvent être réalisées à distance, comme Rowhammer [84], qui cible la DRAM. En sollicitant fortement certaines lignes mémoire, une attaque Rowhammer perturbe les lignes adjacentes (dans l'espace d'adressage de la victime), ce qui peut engendrer des fautes de type *bit-flip*. D'autres approches à distance exploitent des failles logicielles dans les modules d'ajustement dynamique de la tension et de la fréquence (DVFS, *Dynamic Voltage and Frequency Scaling*) [85].

TABLE 3.2 – Résumé de techniques d'injection de fautes. Source [82].

Technique	Précision spatiale	Précision temporelle	Expertise technique	Coûts	Connaissance de l'implémentation	Dégâts possibles de la plateforme
Underfeeding	Haute	Aucune	Basic	Faible	Non	Non
Clock glitch	Faible	Haute	Modérée	Faible	Oui	Non
EM pulse	Faible	Modérée	Modérée	Faible	Non	Possible
Heat	Faible	Aucune	Faible	Faible	Oui	Possible
Power supply glitch	Faible	Modérée	Modérée	Faible	Partielle	Non
Light radiation	Faible	Faible	Modérée	Faible	Non	Oui
Light Pulse	Modérée	Modérée	Modérée	Modérée	Oui	Possible
Laser beam	Haute	Haute	Forte	Haut	Oui	Possible
Focused Ion Beam	Complète	Complète	Très forte	Très haut	Oui	Oui

Les moyens d'injection de fautes regroupent des techniques à faible coût, telles que les brusques modifications (*glitchs*) de tension ou d'horloge qui vont perturber l'intégralité du dispositif cible, et des méthodes à coût modéré/élevé, telles que les impulsions électromagnétiques ou les injections laser. L'injection de fautes laser (*Laser Fault Injection*, LFI) est particulièrement utilisée dans les laboratoires de tests de sécurité, par exemple à des fins de certification, car elle permet une analyse puissante avec une précision temporelle et spatiale élevée [86], en offrant notamment une injection de faute à l'échelle du bit. On parle typiquement de modèle de faute de type *bit-set*, *bit-reset* ou *bit-flip*. Lorsqu'une faute de type *bit-set* est injectée, si la valeur initiale du bit est 0, le bit fauté passe alors à 1 ($0 \rightarrow 1$) mais si la valeur initiale du bit est déjà 1, elle reste à 1 ($1 \rightarrow 1$), c'est-à-dire que le bit ciblé est *protégé* de toute erreur. Ce type de modèles de faute est dit *dépendant des données*, car l'injection d'une faute dépend de l'état initial du bit. Un modèle de faute de *bit-reset* correspond à un comportement symétrique : $0 \rightarrow 0$, $1 \rightarrow 0$. Généralement, un modèle de faute indépendant des données de type *bit-flip* est utilisé et conduit à une inversion de la valeur du bit

quelle que soit sa valeur d'origine : $0 \rightarrow 1$, $1 \rightarrow 0$. Néanmoins, avec l'injection laser, le modèle de faute est généralement soit une mise à 1 (*bit-set*), soit une remise à 0 (*bit-reset*).

L'injection de faute par laser repose sur un phénomène *photoélectrique* induit par l'interaction entre les photons du laser et le silicium de la cible, plus particulièrement au niveau des jonctions PN qui possèdent localement un champ électrique important. La création de paires d'électrons-trous crée alors un *photocourant* transitoire qui peut se propager et induire une faute. Une étude physique plus approfondie est dans [87, 88].

L'utilisation d'analyse *side-channel* et par injection de fautes sont décrites plus précisément dans les sections suivantes 3.5 et 3.6.

3.5 Analyses *side-channel* et confidentialité des modèles

Dans cette section, nous détaillons l'état de l'art de l'exploitation d'analyses *side-channel* dans des scénarios d'extraction de modèle, que cela soit pour l'extraction de l'architecture ou la valeur des paramètres. Notons qu'une étude complète et critique des attaques et des défenses a été proposé à USENIX 2024 dans [89] couvrant l'extraction de modèles embarqués ("on-device") à travers plusieurs mécanismes d'attaques, dont les analyses *side-channel*.

3.5.1 Extraire l'architecture

Dans un scénario d'extraction de modèle de type "fidélité", la connaissance de l'architecture du modèle est fondamentale pour l'attaquant. L'hypothèse de base pour les attaques se concentrant uniquement sur les valeurs des paramètres est de considérer que l'architecture est connue [20, 42, 43, 90], notamment parce que le modèle cible serait issu d'une architecture standard de l'état de l'art et largement présente dans les *model zoos*, voire directement depuis les plateformes de développement de DL [21]. Néanmoins, dans de nombreux cas d'usage, l'architecture du modèle cible peut être issue de l'adaptation d'une famille de modèles connue (e.g., VGG, ResNet) ou avoir été entièrement et spécifiquement conçue pour la tâche à effectuer. Pour tous ces scénarios, il est nécessaire à l'attaquant d'extraire l'architecture en utilisant d'une part une expertise en DL et d'autre part des connaissances liées à la tâche et la nature des données du modèle cible [91].

Un premier vecteur d'attaque est d'exploiter le temps d'inférence comme dans [92] (*Timing Attack*) où l'analyse du temps d'inférence par régression permet d'inférer la profondeur du réseau, c'est-à-dire le nombre de couches. Pour trouver ensuite la bonne combinaison de couches, une technique par apprentissage par renforcement et *Neural Architecture Search* (NAS) permet de réduire efficacement l'espace de recherche et reconstruire une architecture de substitution optimale proche du modèle cible (en comparant les performances des dernières architecture candidates avec la cible sur un jeu de test de l'attaquant). Une limitation importante est que l'approche a été testée uniquement sur des architectures VGG sur CIFAR-10.

Certaines approches reposent sur une hypothèse forte des connaissances de l'attaquant qui ne doit pas extraire l'architecture mais plutôt *choisir* une architecture parmi un ensemble d'architecture possibles. C'est le cas dans [93] qui classifient des traces *side-channel* avec un modèle de type SVM (*Support Vector Machine*) parmi 16 architectures (4 variantes de 4 types de CNNs). De simples modifications de certains hyper-paramètres de ces modèles (nombre et tailles des filtres de convolution par exemple) peut sensiblement réduire la performance de l'attaque.

Les attaques en cache sont particulièrement utilisées mais concernent des cibles plus complexes que les microcontrôleurs utilisés dans nos travaux (processeurs [94–96], GPU [97, 98] ou FPGA [99]). De la même façon, l'analyse des accès mémoire est un vecteur d'attaque privilégié dans [100–102] pour plateformes GPU ou FPGA. Pour GPU, l'analyse du trafic par les ports PCIe permet aussi d'extraire des informations sur l'architecture des modèles [103, 104].

Très peu de travaux se concentrent sur les microcontrôleurs. Dans CSI-NN de Batina *et al.* [105], les auteurs profitent d'une technique d'extraction des valeurs des paramètres pour induire des informations sur les couches du modèle et plus particulièrement sur la frontière entre les couches. Néanmoins, cette approche repose sur une extraction qui est particulièrement difficile et qui possède encore beaucoup de questions ouvertes comme expliqué dans le chapitre 5. Dans le chapitre 5, nous présentons une méthodologie d'extraction de l'architecture **[106]**, uniquement basée sur une *Simple Power Analysis* pour des microcontrôleurs de type ARM Cortex-M (démontrée sur un Cortex-M7) pour lesquels la librairie de déploiement de référence est CMSIS-NN [76].

21. Plusieurs architectures sont directement implémentées et accessibles avec Tensorflow ou Pytorch (Cf. `TorchVision.models` et `TorchText.models`)

En parallèle de ces démonstrations d'attaques qui essaient de couvrir la grande variété de plateformes matérielles qui peuvent supporter l'inférence d'un DNN, des mécanismes de défenses ont aussi été proposés pour rendre les implémentations plus robustes à ce type de menaces. Une méthode privilégiée est d'ajouter de l'aléa, par exemple dans l'espace d'adressage des mémoires pour empêcher les analyses basées sur les *Memory Access Patterns* comme dans [107] où des accès mémoire fictifs sont aussi ajoutés. Une autre approche est de modifier l'architecture des modèles, en ajoutant des couches ou en modifiant certains paramètres des couches de convolutions pour les CNNs, voire en combinant des couches successives (comme convolution+pooling) [108–110]. Pour les modèles CNNs, Luo *et al.* [111] proposent d'adapter les paramètres des couches de convolution pour réaliser une obfuscation spécifique des émanations électromagnétiques sur FPGA. Ceci est réalisé en étudiant finement les rapports entre les traces EM et des couches de convolutions possédant des paramètres différents (taille des tenseurs d'entrée et de sortie, nombre et taille des filtres).

3.5.2 Extraire les paramètres

Dans un scénario d'attaque d'extraction des valeurs des paramètres par analyses *side-channel*, nous faisons l'hypothèse que l'attaquant a une connaissance parfaite de l'architecture.

Comme pour l'extraction de l'architecture, l'exploitation du temps d'inférence permet aussi d'obtenir des informations sur les paramètres, comme dans [112] sur une plateforme CPU (x86) pour lesquelles les multiplications ou additions entre valeurs flottantes 32-bit sous la norme IEEE-754 sont significativement plus longues dès lors qu'elles concernent des valeurs spéciales (∞, *NotANumber*, *denormalized*[22]). En exploitant ce phénomène et des entrées choisies, les auteurs parviennent à récupérer les paramètres d'un MLP à 4 couches. Néanmoins, la contremesure est assez simple, puisqu'il suffit d'activer le mode *flush-to-zero* qui transforme ces valeurs spéciales en 0. Une *timing attack* est aussi au cœur de la méthode proposée dans [113] en exploitant la fonction d'activation ReLU pour des modèles avec des paramètres encapsulées dans des formats différents sur un ATmega-328P, un ARM Cortex-M0+ et une plateforme RISC-V RV32IM. Les auteurs proposent un mécanisme de défense reposant d'une part sur une représentation particulière des valeurs des paramètres en virgule flottante et, d'autre part, sur une nouvelle implémentation de ReLU assurant un temps de traitement constant.

Néanmoins, l'une des principales études concerne l'utilisation d'une approche par CPA [105] (CSI-NN), ou plutôt une CEMA puisqu'elle utilise les émanations électromagnétiques de la cible (microcontrôleurs ATMega328P et ARM Cortex M3, sans FPU) avec une sonde adaptée. CSI-NN est principalement axé sur une extraction "basse fidélité" des valeurs des paramètres qui sont stockées en mémoire en précision flottante 32-bit (IEEE754) mais propose aussi d'inférer le nombre de couches (Cf. section précédente) et les fonctions d'activation (ReLU, Sigmoid, Tanh ou Softmax) par une analyse temporelle. L'extraction des paramètres par CPA ne concerne pas les biais (ce qui a une grande importance, comme nous le détaillons dans le chapitre 5) et cherche à retrouver une valeur approximative des valeurs des paramètres (i.e., 10^{-2}) car les auteurs considèrent que ce niveau de précision est suffisant pour un attaquant même dans un scénario de type "fidélité". Pour simplifier le problème, ils considèrent aussi que les paramètres (qui ont généralement des valeurs assez faibles après l'apprentissage) sont tous bornés ($|w| < N$). L'extraction va s'attacher d'abord aux 8 premiers bits contenant le bit de signe et la partie exposant (7 bits) puis les deux octets de la mantisse. L'implémentation fait intervenir plusieurs opérations qui ne sont pas en temps constant et pour contrer l'effet de la désynchronisation, les auteurs réalisent une opération d'alignement des traces à l'échelle de chaque neurone. L'évaluation est réalisée sur un MLP (entraîné sur MNIST) et à un CNN quantifié sur 8 bits (CIFAR-10) mais la particularité du CNN (les poids sont partagés) sur la stratégie d'exploitation des fuites *side-channel* n'est pas détaillée. Si les résultats apparaissent prometteurs, l'absence de mise à disposition des implémentations et les nombreuses questions méthodologiques ouvertes poussent à réaliser des études plus précises comme celle proposée dans le chapitre 5 (5.2).

3.6 Injection de fautes à l'inférence

3.6.1 Fauter les paramètres

Les principales attaques [6,75,114] visant directement les paramètres d'un réseau de neurone profond ont été démontrées à l'inférence, contre l'intégrité d'un modèle. Dans le cas de modèles

22. Partie exposant à 0 mais pas la mantisse.

pour de la classification d'images, il s'agit pour l'attaquant de réaliser le moins de perturbations possibles sur les valeurs des paramètres pour diminuer la métrique de performance (*accuracy*) jusqu'à atteindre un objectif défini par l'adversaire. La nature de cet objectif permet de définir deux grandes catégories d'attaques – ciblée ou non ciblée – que l'on retrouve classiquement pour d'autres attaques visant l'intégrité d'un modèle, comme les *poisoning attacks* décrites en section 3.3.1.

Comme démontré dans [115], attaquer les paramètres de modèles DNNs quantifiés sur 8 bits en réalisant de façon aléatoire des bit-flips est inefficace. Pour un modèle ResNet-20 entraîné sur CIFAR-10, une centaine de bit-flips aléatoires baisse l'*accuracy* moyenne du modèle de moins de 1%. De très nombreuses études sur le *pruning* de modèle [116] ont montré qu'une minorité de paramètres possède une forte influence sur la prédiction d'un modèle. Aussi, une condition indispensable pour une attaque efficace est d'identifier les paramètres les plus *sensibles*.

3.6.1.1 Attaques non ciblées

Formalisme. En utilisant un ensemble d'évaluation (X^{adv}, Y^{adv}) suffisamment représentatif de la tâche à effectuer par le modèle cible M_W, un attaquant va chercher à corrompre de façon optimale la pertinence moyenne des prédictions, qui est mesurée par la *loss* $\mathcal{L}$. Dans ce scénario non-ciblé, c'est le seul objectif de l'attaquant : il n'a pas la volonté de *spécifiquement* tromper le modèle pour une catégorie d'entrées bien précise.

Formellement, en notant $\tilde{W}$ les paramètres après l'attaque, l'objectif non-ciblé est donc le contraire de la minimisation du risque empirique de l'équation 2.13, qui définit l'objectif de l'apprentissage supervisé de M_W :

$$\max_{\tilde{W}} \sum_{(x_i,y_i)\in(X^{adv},Y^{adv})} \mathcal{L}(M(x_i;\tilde{W}),y_i) \tag{3.7}$$

Néanmoins, comme le vecteur d'attaque est l'injection de fautes dans les paramètres, cet objectif est contraint par un budget S représentant le nombre maximum de fautes qu'il est possible de réaliser. Cette contrainte est généralement représentée en utilisant la distance de Hamming d_H (équation 2.7) entre les paramètres W du modèle cible et les paramètres $\tilde{W}$ après l'attaque. L'équation 3.7 devient donc :

$$\max_{\tilde{W}} \sum_{(x_i,y_i)\in(X^{adv},Y^{adv})} \mathcal{L}(M(x_i;\tilde{W}),y_i) \quad \text{s.t.} \quad d_H(\tilde{W},W) \leq S \tag{3.8}$$

Attaque de référence. La Bit-Flip Attack (BFA), proposée par Rakin *et al.* est l'attaque de référence visant les paramètres d'un DNN [6]. Cette attaque boîte blanche non ciblée repose sur une succession de *bit-flips* effectués jusqu'à ce que la performance du modèle atteigne un niveau proche de l'aléatoire (*random-guess level*). La BFA s'applique à des modèles quantifiés sur 8 bits (ou moins) ce qui correspond à l'état de l'art des bonnes pratiques pour le déploiement des modèles sur des moteurs d'inférence. Les paramètres sont stockés en mémoire sous forme d'entier signé (`int8`). La BFA est une attaque *gradient-based* (et donc boîte blanche), proche du principe de fonctionnement de l'attaque FGSM, pour les *adversarial examples*. Comme pour FGSM, qui cherche à altérer une entrée x pour accroître l'erreur de prédiction en suivant la direction du gradient de la *loss* $\nabla_x\mathcal{L}$, la BFA cherche à altérer les valeurs des paramètres en suivant la direction du gradient de la *loss* $\nabla_w\mathcal{L}$. Néanmoins, contrairement à l'attaque FGSM qui va modifier toutes les dimensions de x (dans le cas d'une attaque contrainte par la norme l_2 ou l_∞), la BFA doit viser un minimum de paramètres (Cf. équation 3.8). Pour ce faire, la BFA trie – par couche – les paramètres qui possèdent les gradients $\nabla_w\mathcal{L}$ les plus forts. Ces paramètres étant ceux qui ont le plus d'influence sur l'erreur de prédiction, les auteurs font l'hypothèse que la modification de leur valeur (par injection de faute) va fortement influencer la performance du modèle. Comme l'altération des paramètres se fait à l'échelle du bit, selon un modèle de faute de type *bit-flip*, les gradients de la *loss* sont aussi calculés pour chaque bit b_i d'un paramètre w. Comme dans [6], on simplifie la notation en posant $\nabla_b\mathcal{L} = [\frac{\partial\mathcal{L}}{\partial b_7}, ..., \frac{\partial\mathcal{L}}{\partial b_0}]$ pour des paramètres quantifiés sur 8 bits.

La sélection des bits est un processus itératif. Lors d'une itération, l'attaquant va analyser chaque couche du modèle les unes après les autres. Pour une couche donnée, il liste les bits les plus sensibles en triant les gradients $\nabla_b\mathcal{L}$ et va simuler pour chacun l'influence sur la *loss* après attaque (bit-flip). L'évaluation des gradients est réalisée à partir d'un ensemble d'attaque restreint de données $(X^{adv}, Y^{adv}) \subset (X^{train}, Y^{train})$. Des alternatives [117, 118] ont été proposées pour construire des données X^{adv} pour des scénarios où l'attaquant a un accès limité à X^{train}. Pour cette

étape de sélection, la BFA n'est pas réellement appliquée (mais simulée) et chaque bit retrouve sa valeur nominale. Quand toutes les couches ont été analysées, l'attaquant sélectionne le bit (à l'échelle du modèle) dont l'attaque conduit à la plus grande augmentation de la *loss* et réalise l'attaque. Puis, le processus recommence jusqu'à ce que le modèle atteigne une performance proche du *random-guess level*.

Pour formaliser l'application de la faute, on utilise un masque binaire m qui va suivre le signe du gradient ascendant de la *loss* (équation 3.9, $\tilde{b}$ sont les bits fautés et $sign(\nabla_b \mathcal{L}) \in \{-1, +1\}^8$) :

$$\begin{aligned} m &= b \oplus \Big(sign(\nabla_b \mathcal{L})/2 + 0.5\Big) \\ \tilde{b} &= b \oplus m \end{aligned} \tag{3.9}$$

Comme noté précédemment, l'équation 3.9 est similaire à l'équation de l'attaque FGSM (équation 3.2). L'utilisation du masque m permet de définir la table de vérité de la BFA, proposée dans le tableau 3.3, qui dépend de la valeur nominale du bit b_i et du signe du gradient.

Table 3.3 – Table de vérité de la BFA.

b_i	sign($\nabla_{b_i} \mathcal{L}$)	$\tilde{b_i}$	m
0	1 (+)	1	1
0	0 (-)	0	0
1	1 (+)	1	0
1	0 (-)	0	1

Dans [6], la BFA est simulée sur des modèles ResNet entraînés sur CIFAR-10 et des modèles AlexNet et ResNet sur ImageNet. Pour CIFAR-10, l'objectif de l'attaquant est d'atteindre une *accuracy* moyenne sur la base de test inférieure ou égale à 11% et 0.2% pour ImageNet. Sur 5 attaques, la valeur médiane du nombre de bit-flips à réaliser pour les modèles ResNet-20,32,44,56 sur CIFAR-10 est, respectivement, 10, 12, 11 et 18 fautes. Les auteurs reportent des résultats similaires pour les modèles entraînés sur ImageNet.

Alternative de Stutz *et al.* Une alternative à la BFA a été présentée dans [114] en partant du principe qu'altérer les paramètres dans le sens du gradient ascendant $\nabla_b \mathcal{L}$ est le processus contraire de la mise à jour des paramètres avec la *backward pass* pendant l'apprentissage (Cf. équation 2.22). Les auteurs de [114] proposent donc de s'affranchir du processus itératif de la BFA en faisant une passe de "contre-apprentissage" et en ajoutant deux contraintes permettant de répondre au vecteur d'attaque par injection de fautes. Formellement, nous avons donc :

$$\begin{aligned} &\tilde{W} = W + \lambda \nabla_W \mathcal{L}(M(x), y)) \\ &d_H(\tilde{W}, W) \leq S \;\text{ et }\; d_H(\tilde{w}, w) \leq 1 \quad \forall w \in W \end{aligned} \tag{3.10}$$

Le processus est répété tant que le budget S n'est pas atteint. λ est similaire au *learning rate* de l'équation 2.22 et permet de régler la vitesse de convergence vers l'objectif de l'attaquant. La seconde contrainte permet de réaliser une seule faute par paramètre. Si plusieurs *bit-flips* concernent un paramètre w, l'attaquant ne conserve que celui d'indice le plus proche du MSB. Stutz *et al.* rapportent des performances supérieures à la BFA et soulignent le fait que la BFA a tendance à faire converger toutes les prédictions vers un seul label. Si leur attaque est aussi plus rapide que la BFA (le processus itératif de la BFA peut effectivement être problématique), elle nécessite plus d'attention sur ses paramètres intrinsèques (notamment le *learning rate* de l'attaque λ, l'utilisation de *momemtum* et de normalisation pour les gradients...) pour être réellement efficace.

Exploitation pratique de la BFA. La démonstration initiale de la BFA dans [6] est purement théorique, à partir d'un ensemble de simulations sur des modèles CNNs de l'état de l'art. Dans `DeepHammer` [115], la BFA est mise en pratique en utilisant Rowhammer [84] comme méthode d'injection de faute contre des plateformes disposant de processeurs Intel (i7) et de mémoire DRAM (DDR3). Comme dans [6], les modèles attaqués sont des CNNs quantifiés sur 8 bits et entraînés sur plusieurs benchmarks (Fashion-MNIST, Google Speech Command, CIFAR-10, ImageNet). L'application de la BFA via Rowhammer dans [115] répond à deux grands problèmes. Le premier est d'adapter la recherche des bits sensibles *et* altérables en prenant en compte les contraintes du système (ici, CPU avec DRAM) et en minimisant simultanément le nombre de bit-flips. En effet, l'altération d'une certaine combinaison de bits peut ne pas être possible si le profil

DRAM des emplacements compatibles avec un bit-flip ne le permet pas. De plus, même si plusieurs bit-flips sont réalisables, le succès de l'attaque tient aussi dans l'altération du moins de bits possible. La deuxième difficulté est directement liée à Rowhammer avec un attaquant qui cherche à altérer plusieurs bits dans un délai d'exploitation raisonnable (la faute par Rowhammer étant transitoire, les bits fautés ne persistent pas après une réinitialisation de la mémoire ou un reboot du système [119]). La principale différence entre `DeepHammer` et la démonstration théorique initiale de la BFA [6] tient donc dans l'addition – après le tri des paramètres par rapport au gradient de la *loss* – d'une phase de filtrage des paramètres qui sont réellement altérables par un bit-flip selon l'organisation de la mémoire DRAM (qui doit être initialement *profilée* comme dans [120]).

Pour Fashion-MNIST, les auteurs rapportent un minimum de 3 *bit-flips* sur le modèle "LeNet" pour atteindre le *random-guess level* (10%). Pour Google Speech Command, seulement 7 *bit-flips* sur un VGG-13 permettent d'atteindre une *accuracy* de 3.25% alors que le modèle nominal avait une performance moyenne de 96.38%. Enfin, pour un modèle ResNet-50 sur ImageNet, 23 *bit-flips* suffisent pour atteindre 0.17% d'*accuracy* (random-guess level : 0.1%).

Récemment, une nouvelle attaque contre les mémoires DRAM, RowPress [121], a été utilisée pour la BFA [122] sur des mémoires DRAM DDR4 (Samsung). A la différence de Rowhammer qui "pilonne" les lignes mémoires, RowPress joue sur la durée d'activation. L'utilisation de RowPress s'avère beaucoup plus efficace que Rowhammer, d'une part pour le nombre de fautes à effectuer et, d'autre part, vis-à-vis de la complexité et la durée nécessaire pour réaliser les fautes. Ainsi, les auteurs rapportent qu'il est possible de réaliser 20 fois plus de fautes avec RowPress que Rowhammer pour une même durée d'attaque [122].

3.6.1.2 Attaques ciblées

Objectifs. Comme pour certaines *poisoning attacks*, les attaques non-ciblées contre les paramètres présentées dans la section précédente, plus particulièrement la BFA, sont autant des attaques contre l'intégrité que contre la disponibilité dans le sens où elles rendent le modèle attaqué inutilisable avec des performances proches d'un niveau de prédiction aléatoire. Suite à la présentation de la BFA, d'autres attaques ont été proposées pour répondre à un objectif *ciblé* de l'attaquant. L'idée principale est d'introduire un comportement malveillant pour certaines données d'entrée choisie par l'adversaire. Les auteurs de la BFA ont donc adapté leur attaque avec la TBFA (pour *Targeted Bit-Flip Attack*) [123] pour répondre à deux grands scénarios ciblés : *1-to-1* et *N-to-1*.

Le premier correspond à corrompre la prédiction d'entrée associée à un label source pour que le modèle prédise toujours le même label cible. Par exemple, pour CIFAR-10, toutes les images d'*avions* (label source) seront classées comme des *chevaux* (label cible). Le choix de l'attaquant est donc double puisqu'il concerne autant le type de données d'entrée que la nature de la sortie.

Dans le second scénario (*N-to-1*), le comportement malveillant ne concerne que le label de prédiction et s'applique à toutes les données d'entrée. Toujours avec CIFAR-10, cela signifie que toutes les images présentées au modèle (et pas uniquement les images d'avions) seront prédites comme des *chevaux*, le label cible. Notons que dans [124], ce scénario est légèrement adapté en dirigeant avec une probabilité uniforme les prédictions des données ciblées sur les $N-1$ autres classes. Contrairement à la BFA, l'attaque dans [124] se concentre sur la couche possédant la matrice de paramètres de rang le plus élevé [23].

Néanmoins, il existe une nuance dans le scénario *1-to-1* par rapport au comportement du modèle pour les données qui ne sont pas ciblées par l'attaquant (dans notre exemple, toutes les images autres que *avion*). L'attaque TBFA *1-to-1* standard n'impose aucun critère pour les données autres que celles visées spécifiquement par l'attaquant : le modèle peut donc répondre correctement ou se tromper pour ces données. L'attaque *1-to-1 Stealthy* permet d'ajouter un critère pour que les perturbations engendrées par les bits-flip (e.g. "avion" → "cheval") n'altèrent pas la qualité de prédiction pour les autres données. Le terme *stealthy* employé par les auteurs de [123] renvoie à un potentiel objectif d'*imperceptibilité* de l'attaquant (classique pour les attaques ciblées) et qui cherchera donc à construire une attaque plus difficilement détectable si le défenseur à la possibilité de valider le modèle sur un ensemble d'évaluation. D'autres attaques ciblées reposent sur des scénarios proches de la *1-to-1 stealthy* comme dans [126] ou [127] et plus particulièrement [128] où le comportement malveillant est appliqué uniquement aux données d'entrée qui possèdent un marqueur construit de la même façon que les perturbations pour les *adversarial examples*.

23. Cf. méthode HRank [125] utilisée pour le pruning de modèle

Formalisme. Contrairement à l'équation 3.8 des attaques non ciblées et la maximisation d'une erreur moyenne de prédictions, l'objectif d'une attaque ciblée comme la TBFA est de minimiser une *loss* $\mathcal{L}_{target}$ spécifique à l'objectif de l'attaquant :

$$\begin{aligned} \min \mathcal{L}_{target} = \min_{\tilde{W}} \mathbb{E}_X \Big[& \alpha_1 \, \mathcal{L}(M(x, \tilde{W}); y_t) + && \textit{(N-to-1)} \\ & \alpha_2 \, \mathcal{L}(M(x, \tilde{W}); y_t) \cdot \mathbf{1}_{x \in X_p} + && \textit{(1-to-1)} \\ & \alpha_3 \, \mathcal{L}(M(x, \tilde{W}); y) \cdot \mathbf{1}_{x \in X_{j \neq p}} \Big] && \textit{(1-to-1) Stealthy} \end{aligned} \tag{3.11}$$

Où p est l'indice associé au label des données ciblées (e.g., "avion"), t l'indice du label de prédiction ciblé (e.g., "cheval") et y le label de la vérité terrain associé à une donnée d'entrée x. $\alpha_1 = 1$ pour une attaque *N-to-1* et 0 sinon. $\alpha_2 = 1$ pour une attaque *1-to-1* (standard ou *stealthy*), 0 sinon et $\alpha_3 = 1$ uniquement pour l'attaque *1-to-1 Stealthy*.

Résultats de la TBFA. Comme pour la BFA non-ciblée, la TBFA va trier les gradients en utilisant la loss $\mathcal{L}_{target}$ (équation 3.11) correspondante. L'attaque a été évalué sur plusieurs CNNs entraînés sur CIFAR-10 et ImageNet. Les auteurs remarquent que l'attaque standard *1-to-1* a une influence très forte sur les données qui ne sont pas visées par l'attaquant : par exemple, sur un MobileNet-V2 entraîné sur ImageNet, l'*accuracy* de test – calculée sans prendre en compte des données ciblées – chute à 1.19% alors que le modèle a une *accuracy* nominale de 72.01 %. L'attaque *stealthy* permet effectivement de retrouver un niveau de performance plus important pour ces données mais la chute de performance reste malgré tout notable (pour le même modèle, l'*accuracy* remonte à 33.99 %). Logiquement, l'attaque *1-to-1* est celle qui nécessite le moins de *bit-flips* car elle possède l'objectif le moins complexe (Cf. équation 3.11 avec $\alpha_1 = \alpha_3 = 0$) surtout avec des modèles possédant une forte capacité (i.e., un très grand nombre de paramètres).

3.6.1.3 Défenses

Les principales protections contre les attaques visant les paramètres reposent sur des mécanismes de détection des fautes pendant l'inférence ou sur des techniques de renforcement des modèles, c'est-à-dire, que l'on cherche, pendant l'apprentissage, à rendre les modèles intrinsèquement plus robustes contre des perturbations comme celles de la BFA.

Vérification de l'intégrité des paramètres. La majorité des approches qui cherchent à détecter les attaques de type BFA reposent sur la création et la comparaison de signatures d'intégrité. Le contrôle d'intégrité est réalisé à l'échelle de chaque couche comme dans *ModelShield* [129] avec une signature de type (*xxHash*)[24] et une méthode de vérification optimisée (parallélisme par arbre de hashage) mais uniquement pour GPU. La vérification peut se concentrer uniquement sur les couches les plus sensibles comme proposé par Javaheripi *et al.* [130] voire certains bits de groupes de paramètres comme dans [131] ou [132]. Ces méthodes sont généralement évaluées sur des CNNs uniquement et leur empreinte mémoire ainsi que l'impact sur le temps d'inférence peut être problématique en fonction des plateformes et des applications.

Une alternative intéressante aux méthodes précédentes reprend le principe des *honeypots* en cyber-sécurité. Dans *Neuropots* [133] des neurones *appâts* sont ajoutés au modèle de telle façon qu'une attaque comme la BFA cible majoritairement ces paramètres. L'idée principale est de réduire le nombre de paramètres dont il faut vérifier l'intégrité. Les *appâts* sont ajoutés aléatoirement puis entraînés par *fine-tuning* en utilisant une *loss* spéciale (calculée au niveau des activations des neurones appâts) qui assure notamment que l'ajout de ces neurones ne perturbent pas la performance nominale du modèle. La détection des fautes se concentre sur une simple comparaison d'un *checksum* des neurones appâts et les paramètres sont réinitialisés si une faute est détectée.

Renforcement des modèles. Le principe de renforcement de modèle a été particulièrement étudiée contre les attaques par évasion avec l'*adversarial training*. L'idée principale est d'injecter à l'apprentissage des perturbations adverses afin que le modèle apprenne à réaliser correctement la tâche en présence de perturbations malveillantes. La transposition aux attaques sur les paramètres paraît donc logique. Néanmoins, les premières approches ne cherchent pas à modifier les paramètres pendant l'apprentissage mais à modifier seulement le niveau de quantification. Les auteurs de la BFA signalaient déjà dans [6] qu'un modèle non quantifié avec des paramètres en

24. https://code.google.com/archive/p/xxhash/

valeurs flottantes 32 bit pouvait être attaqué en modifiant le bit de poids fort de la partie exposant d'un seul paramètre. La première des défenses est donc de réduire les valeurs minimales et maximales possibles par de la quantification. A l'extrême, la *binarisation* des paramètres [75] offre la meilleure robustesse face à la BFA mais détériore trop fortement les performances des modèles. Pour pallier ce problème, ils proposent un *clustering* adaptatif permettant de quantifier chaque couche autour de deux valeurs qui peuvent être différentes de $\{-1;+1\}$. Leurs résultats montrent que sur un modèle VGG-11 avec des paramètres sur 8, 6, 4 et 1 bit, la BFA à besoin respectivement de 16, 21, 49 et 7874 *bit-flips* pour atteindre leur objectif de prédictions aléatoires.

Le *clipping* est une autre méthode (très simple) qui permet de réguler la dynamique de distribution des valeurs des paramètres pendant l'apprentissage et offre des performances très intéressante contre la BFA, comme démontré dans [114]. Le *clipping* borne simplement les paramètres dans un intervalle $[-\alpha, \alpha]$ (adapté pour chaque couche) et est réalisé après chaque itération d'apprentissage :

$$W_l^{clipping} = \max\Big(\min(W_l, |\alpha|), -|\alpha|\Big) \tag{3.12}$$

Comme la quantification, le *clipping* contraint fortement la dispersion des poids et donc l'effet potentiel d'une faute. Néanmoins, son impact est renforcé par son application pendant l'apprentissage car il agit comme une régularisation sur les paramètres, forçant le modèle à utiliser plus de paramètres et à induire une redondance naturelle des paramètres. Il en résulte une distribution plus uniforme de la sensibilité des paramètres vis-à-vis de la tâche et donc moins de paramètres concentrant une forte importance dans la création des *features* les plus utiles à la prédiction.

Notons que le principe du *clipping* peut être adapté à l'inférence en l'appliquant plus finement à l'échelle de groupes de paramètres (par exemple, un filtre de convolution) et combiné à de la quantification, comme dans [134], pour atténuer l'effet d'une faute. Si les résultats de [134] montrent une certaine robustesse contre la BFA pour des modèles CNNs de l'état de l'art, la méthode repose sur plusieurs paramètres dont l'influence peut réduire la performance nominale du modèle.

Enfin, plusieurs études ont essayé d'appliquer l'***adversarial training*** [135] pour les attaques contre les paramètres [75, 114]. La première tentative d'*adversarial weight training* contre la BFA [75] n'est pas concluante car la méthode se heurte à la limitation principale de la BFA, qui est une méthode itérative cumulant les fautes les unes après les autres. L'objectif de leur apprentissage renforcé est défini par l'équation 3.13. En injectant des *bit-flips* pendant l'apprentissage, leur méthode ne fait qu'améliorer le modèle contre un ensemble restreint de fautes sans apporter une robustesse générique. Aussi, la BFA appliquée après leur *adversarial training* va simplement trouver d'autres paramètres qui seront devenus sensibles au cours de cet entraînement.

$$\min_W \mathcal{L}_{adv} = \min_W \mathcal{L}\Big(M(x;W), y\Big) + \alpha \mathcal{L}\Big(M(x;\tilde{W}), y\Big) \tag{3.13}$$

Face à ce problème, Stutz *et al.* proposent deux autres stratégies [114] illustrées dans la figure 3.12 : RandBET injecte des perturbations de type *bit-flip* aléatoire et AdvBET injecte des *bit-flips* optimisés. Dans les deux cas, Stutz *et al.* font une moyenne entre les gradients du modèle fauté et ceux du modèle non fauté pour stabiliser l'entraînement.

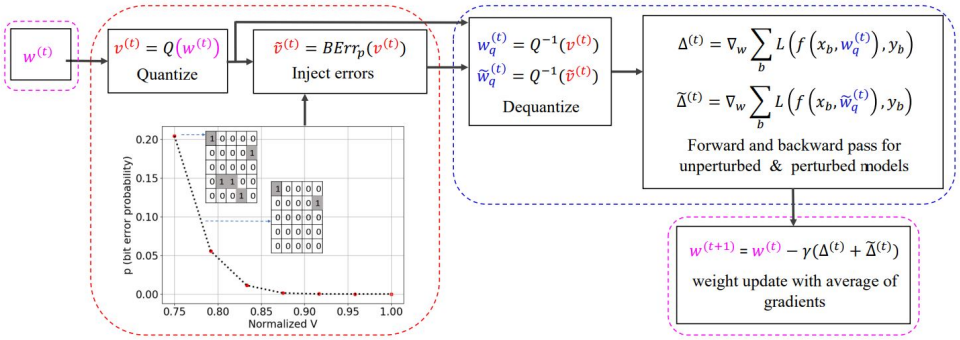


FIGURE 3.12 – Fonctionnement du Random Bit Error Training (RandBET) (source : [114])

Comme pour la première tentative de [75], le bénéfice de AdvBET est faible et l'entraînement est particulièrement complexe à réaliser (lenteur de convergence, problème d'*overfitting*). Au contraire, les résultats de RandBET sont très prometteurs. Bien que les perturbations injectées soient aléatoires, le modèle gagne une réelle robustesse face à la BFA. Néanmoins, RandBET

accentue la dispersion des paramètres, aussi son impact est significativement renforcé quand il est associé à du *clipping*. Ainsi, pour un modèle ResNet-20 entraîné sur CIFAR-10, 30 *bit-flips* suffisent pour converger vers le *random-guess level* (*accuracy de 10%*) alors que le même modèle avec RandBET associé à du *clipping* garde une *accuracy* autour de 70% (valeur nominale : 85%).

Protections des mémoires DRAM. La BFA a principalement été démontrée avec des attaques Rowhammer (ou plus récemment RowPress [122]) visant des mémoires DRAM. Outre les très nombreuses protections qui ont été proposées contre les attaques Rowhammer, et qui visent principalement à surveiller et contrôler la fréquence des activations des rangées de mémoire sur une période spécifiée, certains travaux se sont penchés spécifiquement sur l'impact de schémas de défense contre la BFA. Plus particulièrement, des mécanismes de permutation de rangées de mémoire sont étudiées dans [136] ou [137] pour accroître la complexité des attaques Rowhammer voire bloquer les attaques en profitant de mécanisme de verrouillage. Notons que l'utilisation de l'attaque RowPress pour la BFA est très récente (2024, [122]) et l'état de l'art encore très parcellaire quant aux protections adéquates pour ce vecteur d'attaque.

3.6.2 Fauter les instructions

3.6.2.1 État de l'art

Les premières mentions de saut d'instruction par injection de fautes contre une inférence de réseaux de neurones concernent les fonctions d'activation dans [138, 139]. Néanmoins, seul les codes `C` des fonctions Sigmoid, Tanh et ReLU sont portés dans un `ATMega328P` 8-bit puis attaqués par de l'injection laser. L'impact des fautes observées est analysé en simulation pour des modèles MLP. L'implémentation complète d'un modèle MLP avec 3 couches, toujours sur un `ATMega328P`, est finalement proposée dans [140] ou la fonction d'activation de la seconde couche est fautée par un laser. Les résultats montrent que fauter les fonctions d'activation n'est pas une stratégie intéressante pour un attaquant car il est nécessaire de fauter un trop grand nombre de neurones pour faire chuter la performance moyenne du modèle (au minimum 75% des neurones [139]).

L'attaque de la fonction d'activation Softmax est aussi réalisée par *clock glitch* dans [141] où la fonction est portée sur ATXMega128. La perturbation est réalisée par un générateur de glitch (FPGA DE0-CV) associé à un icWaves (Riscure) et une ChipWhisperer Lite qui permettent de détecter un motif de consommation à partir d'une trace *side-channel* déclenchant l'attaque (le motif de référence étant défini *off-line* par l'attaquant). Le modèle est un CNN, entraîné sur MNIST, avec 2 couches de convolution et une couche dense. Seule la dernière couche est déployée sur le microcontrôleur pour des questions de limitation mémoire [25]. L'attaque permet de sauter une partie de la boucle principale de Softmax, qui calcule les probabilités des prédictions, afin de ne produire qu'un seul score (les autres restant à 0). L'attaque permet de faire chuter l'*accuracy* du modèle à 11.2% avec 98.4% des images du test classées dans le label 0.

Le *clock glitching* est aussi au cœur de [142] mais contre le signal d'horloge du DSP d'un FPGA (Xilinx ZCU102) pour cibler les calculs intermédiaires de convolution d'un CNN. Les expériences portent sur 9 modèles CNNs. Sur 8 des 9 modèles évalués, le taux de mauvaise classification atteint 98%.

3.6.2.2 Expériences sur microcontrôleur

Dans cette section, nous présentons plusieurs expériences, présentées dans **[16]** sur le saut d'instruction contre l'inférence d'un modèle CNN (entraîné sur Fashion-MNIST) embarqué sur un microcontrôleur (ARM Cortex-M) en utilisant de l'injection laser *et* électromagnétique (*pulse* EM). L'objectif est de démontrer l'impact d'une seule faute perturbant le flot d'instructions sur les performances du CNN déployé. Contrairement aux travaux cités ci-dessus, nous considérons un programme d'inférence complet, embarqué avec des outils de déploiement adaptés aux microcontrôleurs et analysons différents chemins d'attaque. Ces expériences ont été les premières à démontrer à la fois des injections électromagnétiques et des injections laser pour un programme d'inférence d'un CNN sur une plateforme ARM Cortex-M4.

Plateforme matérielle, modèle et jeu de données. Notre cible est un microcontrôleur ARM Cortex-M4 capable de fonctionner à une fréquence allant jusqu'à 100 MHz et doté de 512KB de mémoire Flash et 128KB de RAM. Le modèle CNN entraîné sur Fashion-MNIST possède deux couches de convolution (32 et 48 filtres de taille 3×3), chacune étant suivie d'un *MaxPooling*. En

25. Le début du modèle est pré-calculé en Python sur un PC annexe.

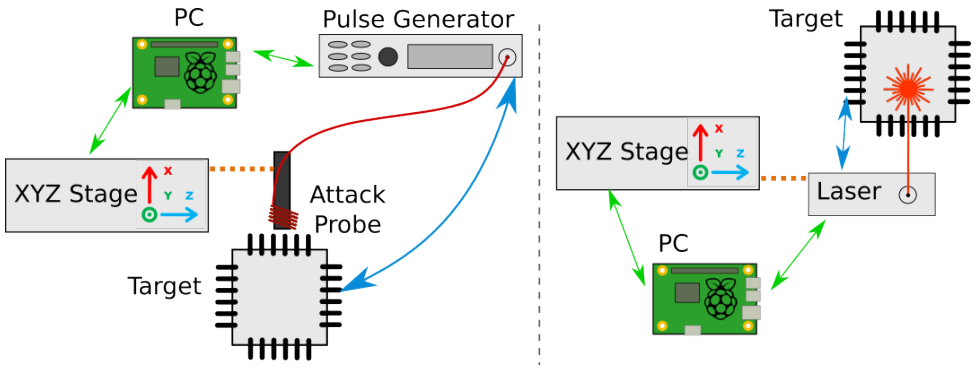


FIGURE 3.13 – Schéma du setup pour l'injection de faute par pulse EM (gauche) et par laser (droite).

fin de modèle, deux couches denses avec 24 et 10 neurones permettent la création des prédictions. La fonction d'activation est ReLU pour tout le modèle à l'exception de Softmax pour la dernière couche dense. Le modèle possède 70 914 paramètres et, après apprentissage sur Tensorflow, atteint une précision de 80% sur la base de test.

Le modèle est ensuite déployé sur la cible avec la bibliothèque NNoM [77]. Les paramètres, activations ainsi que les scores de prédiction sont quantifiés sur 8 bits (Cf. section 3.4.2.3, équation 3.6). NNoM permet d'avoir un accès complet en boîte blanche au code d'inférence. La précision du modèle quantifié en 8 bits est évaluée directement sur la cible avec un échantillon aléatoire de 100 images avec lequel nous obtenons une *accuracy* moyenne de 77%.

Configurations expérimentales. La figure 3.13 propose deux schémas de nos *set-up* expérimentaux pour l'injection de fautes. Pour l'injection de fautes par champ électromagnétique (EMFI), nous utilisons un générateur d'impulsions de tension capable de délivrer des impulsions de 200 V, connecté à une sonde d'injection. La tension de contrôle du générateur d'impulsions est de 200 V avec un temps de montée de 2 ns pour une largeur d'impulsion de 10 ns.

Pour l'injection de fautes par laser (LFI), nous utilisons une source infrarouge de 1 064 nm avec une énergie d'impulsion de 0.1 W, une durée d'impulsion de 50 ns et une taille de spot de 5 µm. Les injections sont réalisées sur la face arrière du silicium, nécessitant une ouverture du composant (mécanique ou chimique [26]). Pour nos expériences, la fréquence de fonctionnement de la carte est réduite à 50 MHz.

Le déclenchement des injections laser et électromagnétiques est synchronisé avec un signal généré par la cible (*trigger*) ce qui nous permet de contrôler l'attaque par rapport à une instruction bien spécifique.

Code de test. La première étape est de réaliser la cartographie des zones sensibles de la cible, c'est-à-dire de localiser les régions où des fautes sont exploitables du point de vue d'un attaquant (dans notre cas, un saut d'instruction). Cette analyse repose sur un code assembleur de test (code 3.1) effectuant des manipulations de registres très simples. En comparant les valeurs retournées des registres après l'exécution du code de test, avec et sans injection de faute, nous cherchons à identifier une zone où une instruction *sub* n'est pas exécutée, c'est-à-dire une faute de type saut d'instruction (*instruction skip*).

Avec EMFI, nous obtenons une zone sensible de 200 µm sur 100 µm (NB : l'aire totale de la cible est 4 mm × 4 mm). Pour obtenir ce résultat, nous avons utilisé une sonde électromagnétique très précise, décrite dans [143]. Dans la figure 3.14, les positions des injections réussies sont représentées en bleu et celles pour LFI en rouge. Ces résultats sont cohérents par rapport à d'autres travaux de référence, notamment [144].

Après la caractérisation des zones sensibles, nous pouvons attaquer notre inférence selon deux chemins d'attaque critiques pour un modèle CNN :

1. la première couche de convolution, qui extrait les *features* de bas niveau à partir d'une image d'entrée,
2. l'ajout des biais dans la première couche dense.

26. L'ouverture est réalisée par la plateforme Micro-Packs : `https://www.pf-micropacks.org/fr/micro-packs/la-plate-forme/`

```
1  movs r3, #1
2  movs r4, #55
3  movs r5, #55
4  nop
5  ...
6  nop
7  sub R4, R4, R3
8  ...
9  sub R4, R4, R3
10 sub R5, R5, R3
11 ...
12 sub R5, R5, R3
13 //lecture des registres ensuite
```

Listing 3.1 – Code de test manipulant les registres.

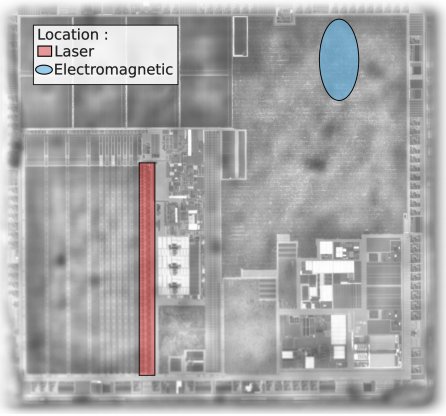


Figure 3.14 – Zones sensibles pour EMFI (bleu) et LFI (rouge, bordure de mémoire Flash).

Saut d'instruction sur la première couche de convolution. L'opération de convolution repose sur des boucles imbriquées dont la première porte le nombre de filtres, comme décrit dans l'algorithme 1. Le code 3.2 est le code assembleur de la boucle. En sautant une instruction (branchement `blt`, ligne 11), notre objectif est d'interrompre prématurément la boucle sur les filtres, arrêtant ainsi l'exécution des convolutions. La conséquence du saut d'instruction dépend fortement de la façon dont cette opération est implémentée dans la librairie utilisée. Dans notre cas (NNoM), une telle faute interrompt complètement le processus de convolution avec une sortie de boucle. Ainsi, si une faute est injectée au moment où le filtre j est appliqué, alors les convolutions postérieures (avec les filtres $[j+1;K]$) ne sont pas réalisées et les calculs continuent pour la suite de l'inférence (i.e., MaxPooling puis seconde couche de convolution, etc.).

Algorithm 1 Couche de convolution (K filtres)

Input : Tenseur X de taille $H \times H \times C$, tenseur des paramètres Θ de taille $Z \times Z \times C \times K$, tenseur des biais de dimension K
Output : Tenseur Y de taille $H \times H \times K$

for k in $[1,K]$ **do**
  for x in $[1,H]$ **do**
    for y in $[1,H]$ **do**
      $Y_{x,y,c} = B_k$
      for m in $[1,Z]$ **do**
        for n in $[1,Z]$ **do**
          for c in $[1,C]$ **do**
            $Y_{i,j,c} += \theta_{m,n,k,c} \cdot X_{x+m,y+n,k}$
return Y

Par simulation, nous observons un chemin d'attaque valide au niveau de l'instruction de branchement (ligne 11 dans le code 3.2). La Figure 3.15 (courbe orange) présente l'impact sur la performance du modèle (sur 100 images) d'injections de fautes simulées en fonction de l'index du dernier filtre exécuté (les filtres restants sont donc ignorés). Logiquement, la performance (*ac-*

```
1  ldr   r3, [r7, #100]
2  adds  r3, #1
3  str   r3, [r7, #100]
4  ldr   r2, [r7, #72]
5  ldr   r3, [r7, #68]
6  add   r3, r2
7  str   r3, [r7, #72]
8  ldrh.w   r3, [r7, #132]
9  ldr   r2, [r7, #100]
10 cmp   r2, r3
11 blt.w   80037fa //cible de la faute
```

Listing 3.2 – Code assembleur de la boucle sur les filtres de convolution. L'instruction ciblée est l'instruction de branchement ligne 11.

curacy) du modèle diminue d'autant plus que la boucle est interrompue au niveau des premiers filtres. Néanmoins, il est très intéressant d'observer que les derniers filtres n'ont pas un impact significatif sur la qualité de la prédiction, puisque sortir de la boucle après le 17^{me} filtre ne perturbe que légèrement la performance du modèle (de 75% à 82%). Une hypothèse permettant d'expliquer ce phénomène est que le modèle est vraisemblablement en surcapacité et qu'une partie de ces paramètres n'a pas de réelle influence sur la prédiction. La majorité des caractéristiques utiles sont *capturées* par les premiers filtres de la couche. Nous avons reproduit des résultats similaires avec d'autres jeux de test de 100 images. Ce résultat est particulièrement intéressant car il permet à l'attaquant d'obtenir des informations sur le modèle (et pas uniquement d'attaquer son intégrité). En effet, le saut d'instruction sur la boucle des filtres de convolution apporte des informations importantes sur l'importance des filtres dans la prédiction : à partir du 17^{me} filtres, les filtres ont peu d'importance sur la prédiction. Dans le cas d'un scénario d'extraction de modèle, une telle connaissance peut aider un attaquant à concentrer ses efforts uniquement sur la récupération d'une petite partie des paramètres et réduire la complexité de son attaque.

Pour démontrer cette attaque en pratique, nous réalisons une EMFI sur les 100 images de notre ensemble de test original. Les résultats expérimentaux et simulés, présentés dans la Figure 3.15 (courbes orange et bleu), sont presque identiques. Les différences notables s'expliquent par la difficulté de répétabilité de l'injection électromagnétique et des potentiels échecs dans l'injection. Il est aussi possible que l'injection provoque un arrêt de la cible, ce qui se produit aux filtres 14 et 19 (chutes significatives de précision à ces deux indices).

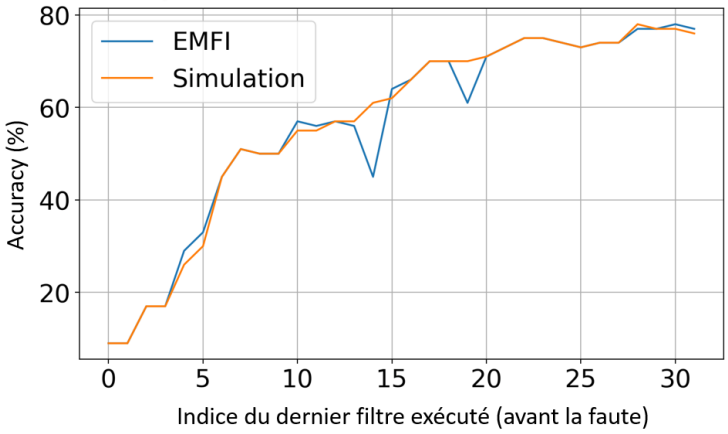


Figure 3.15 – Performance du modèle (*accuracy*) avec un saut d'instruction sur la première couche de convolution (simulation et EMFI).

Pour l'injection laser, nous n'avons réussi qu'une seule injection de faute sur le premier filtre, qui donne une performance à 10% comme les simulations et EMFI (nous obtenons logiquement un niveau de prédiction aléatoire puisque l'intégralité de la *forward pass* de l'inférence est perturbée et aucune caractéristique n'est extraite). Néanmoins, la complexité pour le déclenchement du tir (position et délai d'injection), pour l'injection sur les filtres suivants, a été très importante et nécessite des travaux complémentaires ultérieurs.

Avec l'implémentation de NNoM, nous observons qu'une sortie prématurée de la boucle (dès les premiers filtres) entraîne une prédiction qui est identique à la précédente inférence, exécutée

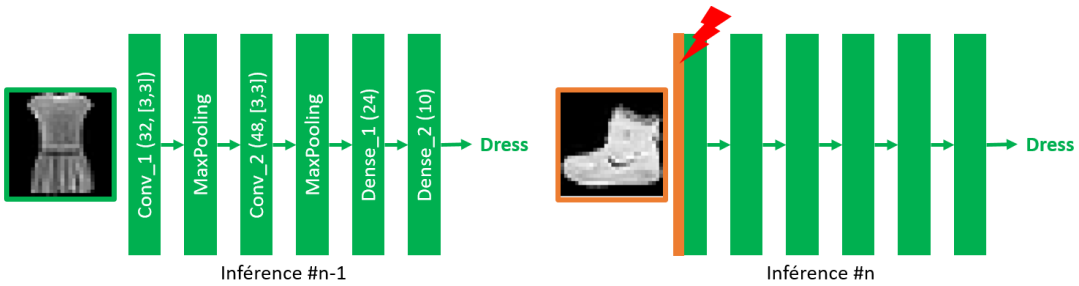


FIGURE 3.16 – Illustration de l'*effet mémoire* lors du saut de la première couche de convolution.

correctement. Cet *effet mémoire*, illustré dans la figure 3.16, peut-être exploité par un attaquant en fonction de la tâche et du cas d'usage du système (e.g., authentification de personne). L'attaque est possible car le résultat des calculs de la couche de convolution est stocké en mémoire RAM mais sans aucune réinitialisation.

Saut d'instruction sur l'initialisation des biais. Notre deuxième chemin d'attaque est d'altérer les biais des dernières couches, plus particulièrement en visant l'instruction `store` qui initialise la valeur d'un biais. L'attaque cherche à corrompre significativement les valeurs de biais, ce qui (les biais étant en fin de modèle) peut conduire à des erreurs de prédiction. En simulation, nous écrivons une valeur d'adresse dans le registre à la place de la valeur du biais de la première couche dense ce qui permet d'attribuer le biais à une grande valeur. Nous exploitons aussi ce chemin d'attaque avec des EMFI et LFI. Les résultats sont présentés dans le tableau 3.4 en fautant chacun des 24 biais (moyenne et écart type). Si l'on analyse plus finement les résultats pour chaque neurone, on remarque que LFI est très proche de la simulation. Pour EMFI, même si les valeurs de biais sont différentes (par rapport à la simulation), les fautes altèrent sensiblement les prédictions.

TABLE 3.4 – *Accuracy* moyenne (et écart type) avec injection de faute à l'initialisation des biais sur la première couche dense

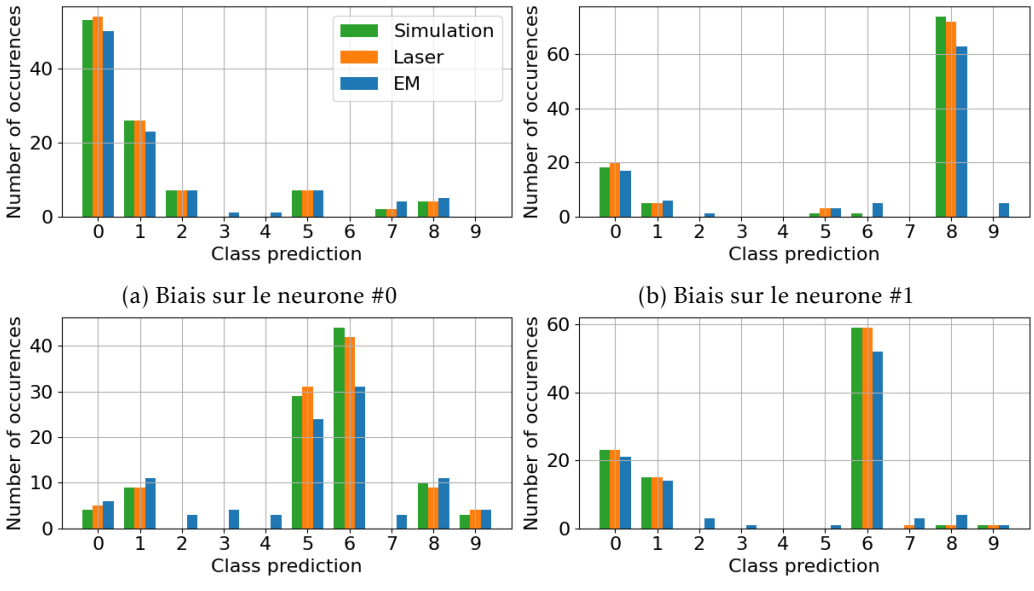
	Accuracy Simulation	*Accuracy* Laser	*Accuracy* EM
Avec faute	33% (6.1)	33% (6.2)	42% (8.2)
Sans faute	77%	77%	77%

La figure 3.17 représente la distribution des prédictions après une faute sur les quatre premiers neurones. Par exemple, la modification du biais du premier neurone (indice #0) a entraîné des prédictions majoritairement sur le label 0 (*T-shirt*) dans environ 50% des inférences. En fonction du biais visé, d'autres labels peuvent être ciblés, comme l'avant-dernier label (*Bag*) pour le biais du second neurone (indice #1).

Une protection possible est de faire du *clipping* des valeurs des biais en limitant leurs valeurs minimale et maximale. Nos premières expériences sur les quatre premiers neurones montrent qu'un *clipping* par rapport à la valeur 2048 permet d'annuler l'effet de l'attaque et de revenir à un niveau de performance nominale.

Commentaires et expériences supplémentaires. Nous avons aussi expérimenté une attaque de la fonction d'activation ReLU au niveau de la première couche dense. Il est possible en effet de forcer (ReLU=0) ou d'ignorer la remise à zéro (ReLU=Identité) pendant l'application de ReLU. Nos expériences en simulation, EMFI et LFI ont montré qu'un seul saut d'instruction ne permettait pas de faire sensiblement chuter l'*accuracy* moyenne du modèle (Cf. [16]). Cette observation est cohérente avec les résultats de [138] dans lequel il était nécessaire d'effectuer un grand nombre de fautes sur ReLU pour altérer la précision globale d'un modèle MLP à 5 couches (les fautes étaient injectées sur l'avant-dernière couche).

Les expériences présentées ci-dessus sont des premières tentatives qui démontrent néanmoins la forte sensibilité d'une inférence d'un DNN. Pour ces expériences, nous nous sommes placés en boîte blanche pour faciliter notre caractérisation. Plus particulièrement, nous avons synchronisé les injections directement avec un *trigger* dans le code. Ceci est notamment possible car nous utilisons une librairie de déploiement (NNoM) *open source*. Néanmoins, il nous paraît envisageable



(a) Biais sur le neurone #0

(b) Biais sur le neurone #1

(c) Biais sur le neurone #2

(d) Biais sur le neurone #3

FIGURE 3.17 – Répartition des prédictions après la modification du biais des 4 premiers neurones.

de déclencher les tirs laser ou les impulsions EM à partir de motifs *side-channel* car le code d'inférence fait intervenir de nombreuses boucles dont les fuites électromagnétiques sont parfaitement repérables, comme nous le décrivons dans le Chapitre 5. Une perspective intéressante serait donc de coupler injection de faute et analyse *side channel* pour automatiser l'attaque et d'évaluer l'attaque sur d'autres librairies de déploiement, comme la plateforme STMCubeMX.AI.

Réaliser des expériences avec d'autres librairies de déploiement est aussi intéressant pour l'exploitation des chemins d'attaques que nous avons présentés. Plus particulièrement, notre étude sur le saut d'instruction au niveau de la boucle des convolutions est fortement dépendante de l'implémentation de la librairie NNoM. Nous avons simulé une attaque dans laquelle nous ne sortons pas de la boucle après la faute mais ignorons seulement le filtre visé. Dans ce cas, nous observons des chutes significatives de la performance du modèle, jusqu'à 20% en visant un seul filtre. Ceci est parfaitement cohérent avec l'attaque BFA que nous avons décrite dans les sections précédentes et que nous analysons dans le chapitre 4.

Une librairie standard pour les microcontrôleurs de la famille Cortex-M est la librairie de ARM, CMSIS-NN [76], qu'il est possible de sélectionner comme *backend* dans NNoM. La principale différence est pour l'opération de convolution, car CMSIS-NN utilise l'algorithme `im2col` [76] qui transforme l'image d'entrée et l'ensemble des filtres en une nouvelle représentation matricielle, de sorte que la convolution soit effectuée comme une multiplication matricielle. Nous avons effectué un premier ensemble de tests de simulation sur une implémentation basée sur CMSIS-NN qui a montré qu'un seul saut d'instruction dans la boucle sur la dimension du tenseur de sortie (i.e., le nombre de filtres) de la première couche de convolution provoque une prédiction forcée sur le label 0 (*T-shirt*) dans 98% des cas. D'autres expériences sont nécessaires pour analyser d'autres vulnérabilités potentielles de `im2col`.

Enfin, concernant les protections, nous avons mentionné quelques conseils évidents pour réduire l'impact des fautes (réinitialisation de la mémoire, *clipping*) mais le principal défi réside dans la taille et la complexité du code d'inférence. Des protections basées sur des vérifications d'intégrité ou de redondance peuvent entraîner des coûts supplémentaires prohibitifs et réduire considérablement les performances du système. Des schémas de défense prometteurs incluent les protections basées sur le CFI (Intégrité du Flot de Contrôle) qui visent à vérifier le flux d'exécution d'un programme et à détecter toute altération potentielle [145, 146].

Chapitre 4

Évaluer les attaques contre les paramètres en mémoire

Contents

Introduction. Les attaques contre l'intégrité d'un modèle de réseau de neurones profond, qu'elles soient à l'inférence ou l'apprentissage, sont principalement des menaces reposant sur l'exploitation de paires entrée/prédiction et qui ne cherchent pas à perturber intrinsèquement le modèle (son architecture ou ses paramètres). C'est le principe de base des *adversarial examples* ou des menaces par empoisonnement de données que nous avons décrit dans le chapitre précédent. La prise en compte d'une surface d'attaque plus grande a permis de définir des menaces qui ciblent directement les éléments les plus sensibles d'un DNN, c'est-à-dire leurs paramètres. Ces attaques, nommées "*Parameter-based Adversarial Attacks*" ont d'abord été démontrées théoriquement, puis ont été connectées aux techniques d'attaques physiques par injection de fautes.

Organisation. Dans ce chapitre, nous proposons des études portant sur la principale attaque de référence, la Bit-Flip Attack (BFA), démontrée par Rakin *et al.* à ICCV 2019 [6]. La BFA est une attaque boîte blanche non-ciblée rendant inutilisable des modèles CNN de l'état de l'art (e.g., ResNet) en seulement une dizaine de *bit-flips*.

Dans un premier temps (4.1), nous apportons une analyse critique de la BFA et démontrons que son évaluation est complexe à cause de sa grande variabilité, d'une part par rapport aux paramètres intrinsèques de l'attaque (4.1.2) et, d'autre part, par rapport aux types de modèles et certains hyper-paramètres d'apprentissage (4.1.3). Ces analyses soulignent l'importance d'une définition appropriée du modèle de menace et d'un ensemble de bonnes pratiques permettant d'évaluer correctement la sensibilité d'un modèle contre ce type d'attaque et d'éviter des biais d'évaluation pouvant conduire à un faux sentiment de sécurité.

Dans un second temps (4.2), nous nous plaçons dans un contexte d'évaluation sécuritaire pratique, sur banc de test, avec un microcontrôleur soumit à de l'injection laser. Cette étude est la première réalisée de ce type et repose sur un modèle de faute *bit-set* déjà démontré sur mémoire Flash (ARM Cortex-M). Après une première caractérisation sur un neurone (4.2.6) et un modèle (4.2.7), nous montrons comment adapter la BFA théorique aux spécificités de l'injection laser (4.2.8) avant de discuter des limitations et perspectives de ces premières expériences (4.3).

Projets, encadrements et publications associées. Les travaux sur les menaces par injection de fautes contre les paramètres ont été réalisés dans le cadre des projets (Cf. section 1.3) :

- ANR PICTURE (AAPG 2020, 2021-2024, coordinateur)
- IRT NANOELEC, Programme PULSE (2021-2025, responsable *Implémentation Sécurisée d'IA*)
- EU ECSEL INSECTT (2020-2023, responsable *AI Validation & Verification*)

Ces travaux ont été réalisées à travers l'encadrement de la thèse de **Kevin HECTOR** (ANR PICTURE, 2021-2024) et du post-doctorat de **Mathieu DUMONT** (EU ECSEL INSECTT, 2021-2023) avec les collaborations de **Jean-Max DUTERTRE** (MSE) et **Simon PONTIE** (CEA LETI).

Une partie des résultats présentés dans ce chapitre ont été publiés dans les deux publications suivantes :

- [147] Hector, K., Moëllic, P. A., Dumont, M., & Dutertre, J. M. *A closer look at evaluating the Bit-Flip Attack against deep neural networks*. In 2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS) (pp. 1-5). IEEE [1].
- [148] Dumont, M., Hector, K., Moëllic, P. A., Dutertre, J. M., Pontié, S. *Evaluation of parameter-based attacks against embedded neural networks with laser injection*. In International Conference on Computer Safety, Reliability, and Security (SafeComp) (pp. 259-272).

4.1 Analyse et évaluation de la Bit-Flip Attack

4.1.1 Positionnement, modèles et bases de données

Positionnement. Dans le contexte actuel de standardisation et régulation des systèmes de ML, l'évaluation des risques est devenue une priorité. L'état de l'art montre de réelles difficultés pour des protocoles d'évaluation robustes vis-à-vis de menaces pourtant analysées depuis plus de 10 ans [149, 150]. Dans les sections suivantes, nous montrons que les attaques sur les paramètres, plus particulièrement la BFA, n'échappent pas à ce problème et qu'il est urgent de définir des méthodes permettant d'analyser finement la robustesse d'un modèle vis-à-vis de telles menaces.

1. https://gitlab.emse.fr/securityml/closerlook-bfa

Dans un premier temps, nous nous plaçons dans le contexte d'une évaluation boîte blanche, réalisée par un concepteur d'un modèle qui a la capacité d'entraîner et d'interroger le modèle sans aucune restriction. Cela nous permet d'interroger la pertinence du modèle de menace original de la BFA, proposé dans [6], et utilisé dans la majorité des références traitant de la BFA ou de ses variantes (non-ciblées). Une analyse précise de [6] et [75] montre des problèmes de variabilités de l'attaque qui ne sont pas réellement traitées et soulèvent de nombreuses questions d'évaluation. Aussi, nous étudions l'influence de paramètres intrinsèques de la BFA ainsi que l'impact de l'architecture et d'hyper-paramètres du modèle cible.

Bases de données et architectures des modèles. Comme dans la majorité des travaux traitant de ce type de menaces, nous utilisons la base de données CIFAR-10. Nous repartons des études des deux principales références, [6] et [75], en utilisant les deux architectures standards de l'état de l'art, VGG-11 et ResNet-20. Nous renvoyons au répertoire public de l'étude pour le détail des architectures [2]. VGG-11 [151] est composé de 8 couches de convolution et 3 couches denses. ResNet-20 [53] comprend 19 couches de convolution et une seule couche dense. Par soucis de concision, pour ResNet-20, nous appelons "Étage" un *residual block* qui est classiquement composé d'une branche principale avec deux couches de convolution et d'une branche secondaire (*skip connection*) qui améliore le flux des gradients et réduit le risque de *vanishing gradient* [53]. Après la première couche de convolution, ResNet-20 est ainsi composé de trois ensembles de 3 étages (par exemple, "*Étage 1.0*", "*1.1*", "*1.2*"). Chaque couche de convolution est suivie d'une couche de normalisation (*batch normalization* - BN). Tous les filtres de convolution ont une taille de 3×3 et tous les filtres de *pooling* ont une taille de 2×2 sauf en fin de modèle ResNet-20 (8×8).

Nous étudions aussi les modèles MLP qui sont étrangement exclus de l'état de l'art. Pour l'étude des MLP, nous utilisons MNIST avec deux modèles : un MLP classique, nommé simplement **MLP** (tableau 4.1) et un modèle hybride, nommé **C-CNN** (tableau 4.2) contenant le modèle MLP auquel est ajouté une couche de convolution en début de modèle.

Comme dans [6] et [75], les paramètres des modèles sont quantifiés sur 8 bits avec le même schéma de quantification défini dans le chapitre précédent par l'équation 3.4 (page 38).

Table 4.1 – Architecture dU MLP

Dense 1 - (512) + ReLU
Dense 2 - (256) + ReLU
Dense 3 - (128) + ReLU
Dense 4 - (10) + Softmax

Table 4.2 – Architecture de C-CNN

Conv. 1 - (32)
Dense 1 - (512) + ReLU
Dense 2 - (256) + ReLU
Dense 3 - (128) + ReLU
Dense 4 - (10) + Softmax

Pour les modèles VGG-11 et ResNet-20, nous adaptons certains hyper-paramètres d'apprentissage de [6] et [75] pour améliorer la vitesse de convergence et raccourcir le temps d'apprentissage. Ces paramètres sont résumés dans le tableau 4.3.

Table 4.3 – Hyperparamètres d'entraînement pour VGG-11 et ResNet-20.

Hyperparamètres	VGG-11	ResNet-20
Batch Normalization	Non	Oui
Dropout	0.5	Non
Initialisation des paramètres	Normale	
Optimiseur	SGD	
Learning rate	$\lambda = 0.1$	
Scheduler	exponentiel (0.95)	
Epochs	40	
Taille du *batch*	128	

4.1.2 Analyse et critique du modèle de menace original

2. https://gitlab.emse.fr/securityml/closerlook-bfa

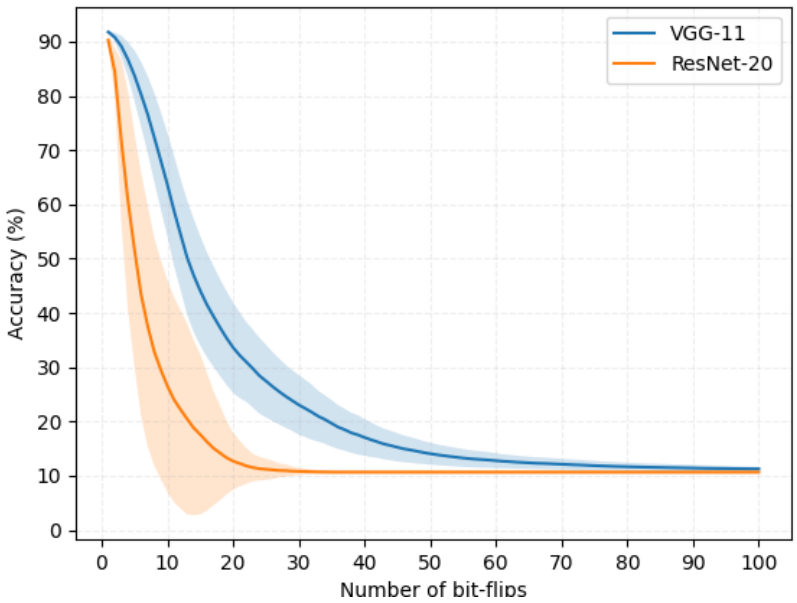


FIGURE 4.1 – Résultats de la BFA sur ResNet-20 et VGG-11. Moyenne (écart-type) sur 5 modèles et 5 ensembles d'attaques.

4.1.2.1 Rappel du modèle de menace

Comme nous l'avons détaillé dans le chapitre précédent (section 3.6.1, page 43), la BFA cherche à altérer *l'accuracy* d'un modèle M_W en sélectionnant les bits les plus sensibles et en les attaquant avec un modèle de faute de type *bit-flip*. La BFA étant une attaque boite-blanche, l'attaquant a une connaissance parfaite du modèle victime (son architecture, ses paramètres, fonctions d'activation...) et peut donc calculer les gradients $\nabla\mathcal{L}_w$. L'attaquant connaît aussi l'ensemble d'entraînement (ou un ensemble de validation et de test proche de celui d'entraînement) lui permettant de créer un ou plusieurs ensembles d'attaques pour sélectionner les paramètres à fauter. Il peut accéder au modèle victime (ou à des clones) et réaliser autant de requêtes que nécessaire, sans limite ou contrainte.

Comme souligné dans [6], l'objectif de l'attaquant est *jusqu'au-boutiste* dans le sens où il cherche à diminuer la performance du modèle victime, mesurée par son *accuracy*, jusqu'à un niveau de prédiction aléatoire ($\approx 1/|Y|$, avec $|Y|$ le nombre de labels). Rakin *et al.* dans [6] utilisent un objectif de 11% pour CIFAR-10. L'attaquant n'a aucun *budget*, c'est-à-dire qu'il n'est pas limité par un nombre maximum de *bit-flips* qui serait fixé soit par la technique d'injection (qui peut s'avérer couteuse et complexe) soit par d'éventuelles capacités de détection du défenseur. Dans l'état de l'art faisant suite à la présentation de la BFA, rares sont les travaux qui fixent un tel budget (citons [117] qui impose 30 bit-flips).

En simulant la BFA sur VGG-11 et ResNet-20 (simulation Python à partir du code d'origine fourni par les auteurs de la BFA), nous obtenons les résultats de la figure 4.1. Outre l'efficacité de l'attaque, une première observation est que le nombre de fautes nécessaires pour atteindre un niveau de prédiction aléatoire n'est pas totalement pertinent pour évaluer l'efficacité de l'attaque. En effet, si les premières fautes sont extrêmement performantes, on remarque surtout un effet plateau en dessous de 20% qui témoigne d'une accumulation très importante de fautes pour atteindre l'objectif des 11%. Cela implique une surévaluation du nombre de fautes qu'il est nécessaire de réaliser contre le modèle.

Les attaques précédentes ont été réalisées en considérant un ensemble d'attaque composé de 100 images échantillonnées à partir de l'ensemble d'apprentissage. La taille de l'ensemble d'attaque n'a pas une influence significative sur la performance de la BFA. Par exemple, en ne considérant que VGG-11 et en effectuant 10 attaques avec des images échantillonnées pour trois tailles d'ensemble d'attaque (50, 100 et 500 images), nous obtenons les résultats du tableau 4.4. Le plus intéressant dans ces résultats est le fort écart type observé quel que soit la taille de l'ensemble d'attaque. Pour 500 images, il faut en moyenne 47 *bit-flips* mais la variabilité est importante : l'attaque la plus performante permet de faire chuter le modèle à 10.98% en seulement 33 *bit-flips* quand une autre attaque doit rassembler 80 *bit-flips*.

En considérant maintenant les deux modèles VGG-11 et ResNet-20, en fixant la taille de l'ensemble d'attaque à 100 images et en réalisant 5 attaques, toujours en utilisant un tirage aléatoire

Table 4.4 – BFA contre VGG-11 en variant l'ensemble d'attaque. Moyenne (écart-type) du nombre de *bit-flips* pour atteindre 11% d'accuracy.

Taille d'attaque	50	100	500
# *bit-flips*	49.5 (21.05)	69.5 (38.71)	47 (17.19)

des images dans l'ensemble de test, nous obtenons les résultats du tableau 4.5. Pour cette expérience, nous utilisons aussi plusieurs objectifs d'attaque.

Table 4.5 – Moyenne (écart-type) du nombre de *bit-flips* sur 5 attaques pour atteindre 11%, 25%, 50% et 75% *d'accuracy*.

Objectif *d'accuracy* (%)	ResNet-20	VGG-11
11 [6]	20.6 (5.08)	72 (36.01)
25	8.8 (1.83)	14.2 (4.21)
50	6.2 (1.72)	6.8 (0.75)
75	3.4 (0.8)	3 (0)

Nous observons une baisse significative de la variabilité de l'attaque dès lors que l'on considère un objectif plus réaliste par rapport à celui, extrême, choisi dans [6] (*random-guess level*). Ainsi, pour un objectif de 25%, l'écart-type n'est plus que de 4 *bit-flips* pour VGG-11 et chute sous 1 *bit-flip* à partir de 50%. Si on réalise l'évaluation d'un modèle avec une seule attaque et l'objectif original à 11%, il n'est pas possible d'avoir confiance dans l'estimation de la robustesse du modèle.

Atteindre une performance comparable à des prédictions aléatoires est un objectif discutable puisque que l'on peut considérer qu'une attaque faisant régresser un modèle initialement à 90% à 25% est déjà un succès pour l'attaquant et que, dans la majorité des cas d'usage, un tel modèle serait considéré comme parfaitement *inutilisable*. De plus, un tel objectif implique de multiplier significativement les fautes (effet plateau rapidement observé dans la figure 4.1) ce qui peut paraître contradictoire pour une menace qui repose sur de l'injection de fautes qui repose traditionnellement sur le concept (plus réaliste) de *budget de l'attaquant*, c'est-à-dire un nombre maximal de fautes qui dépend de la technique d'injection ou de contraintes imposées par d'éventuelles techniques de détection. Dans le cadre de la BFA, il paraît plus judicieux d'utiliser un ou plusieurs budgets pour évaluer correctement l'efficacité réelle de la menace.

4.1.3 Influence du modèle

Dans les sections suivantes, nous nous intéressons à l'influence du modèle (son architecture, initialisation et apprentissage) dans la performance de la BFA. L'objectif est d'apporter des éléments permettant d'expliquer un certain niveau de robustesse *naturelle* d'un modèle et de définir des premiers éléments de *bonnes pratiques* permettant d'apprendre des modèles intrinsèquement plus résistants aux attaques de type BFA.

4.1.3.1 Effet de l'initialisation et de la dispersion des paramètres

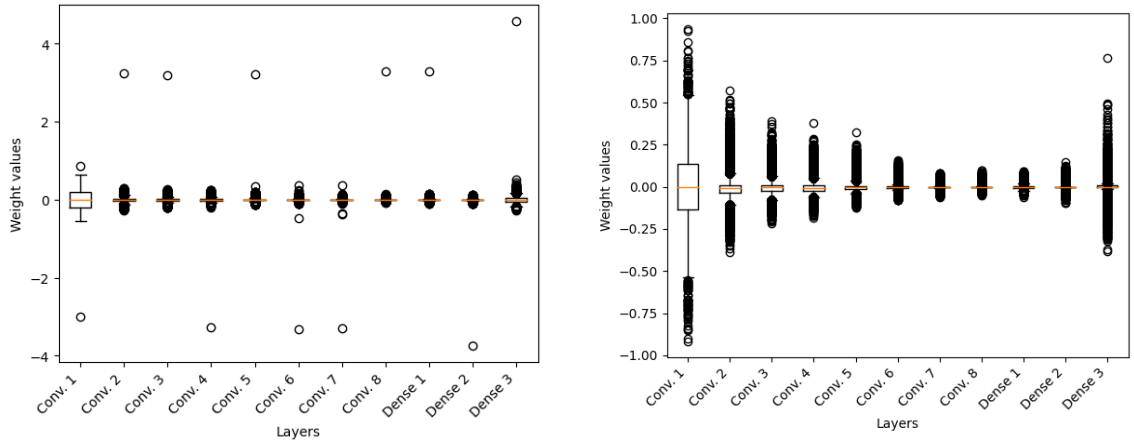
Par la nature même de la BFA, les premiers éléments à prendre en compte est la façon dont les paramètres sont *initialisés*, *optimisés* pendant l'apprentissage et *distribués* dans le modèle (notamment avec la quantification).

Initialisation des paramètres. Une première expérience est d'entraîner plusieurs fois un modèle avec des valeurs initiales des paramètres différentes (initialisation selon une distribution Normale). Pour ce faire, nous reprenons l'expérience qui a conduit au tableau 4.5 mais en ajoutant 4 modèles VGG-11 et ResNet-20. Ces modèles sont numérotés de "2" à "5" dans le tableau 4.6 ("1" est le modèle utilisé dans le tableau 4.5). Rappelons que la moyenne (et la déviation standard) sont toujours calculées pour 5 attaques en faisant varier l'ensemble d'attaque (taille 100).

Comme précédemment, nous observons des différences significatives de performances pour l'objectif initial (11%), par exemple pour VGG-11, entre les 4^{ime} et 5^{ime} modèles avec respectivement 42 et 113 *bit-flips* en moyenne. Néanmoins, pour des budgets plus cohérents (75%, 50%,

TABLE 4.6 – Moyenne (Écart-type) du nombre de bit-flips sur 5 attaques pour atteindre 11%, 25%, 50% et 75% *d'accuracy*, pour 5 modèles entraînés.

Modèle	objectif *d'accuracy* (%)	**1**	**2**	**3**	**4**	**5**
VGG-11	11 [6]	72 (**36.01**)	55.4 (**20.92**)	81.6 (**25.5**)	42.6 (**9.02**)	113.2 (**51.92**)
	25	14.2 (4.21)	13.8 (2.71)	16.0 (2.19)	13.2 (1.47)	19.6 (3.77)
	50	6.8 (0.75)	7 (1.1)	8.4 (1.36)	6.6 (0.8)	9(1.1)
	75	3 (0)	3.2 (0.4)	3.4 (0.49)	3.2 (0.4)	4 (0)
ResNet-20	11 [6]	20.6 (**5.08**)	18.8 (**8.28**)	21.4 (**4.49**)	27.4 (**11.22**)	7.0 (**2.09**)
	25	8.8 (1.83)	8 (0.63)	9.6 (1.02)	12.4 (1.02)	3.6 (0.49)
	50	6.2 (1.72)	4.4 (0.49)	5.4 (0.8)	6.6 (0.8)	2.4 (0.49)
	75	3.4 (0.8)	2.2 (0.4)	3 (0.63)	3.6 (0.49)	1.6 (0.49)



(a) (b)

FIGURE 4.2 – VGG-11 : (a) distribution (problématique) des paramètres dans le modèle de [75] et (b) dans un de nos modèles.

voire 25%), la variabilité est moins importante. De même, si on utilise différents échantillonnages, Normale et Uniforme et leurs variantes de Xavier [152] (qui permet de conserver la même variance à travers toutes les couches), nous n'observons pas de réelles différences.

Quantification et dispersion des paramètres. Si la nature de la distribution pour l'initialisation des paramètres ne semble pas avoir une influence notable sur la BFA, leur dispersion a nécessairement une forte importance du fait de leur quantification. Pour illustrer cette importance, prenons un paramètre en précision flottante sur 32 bits : $w_{32} = 0.0123$. Ce paramètre est quantifié selon le schéma de l'équation 3.4 (comme dans [6] et [75]). Le pas de quantification (par couche) dépend de la valeur maximale des paramètres. Faisons l'hypothèse que la valeur absolue maximale des paramètres de la couche de w est égale à 0.1. La valeur quantifiée sera alors $w_8 = 16 = b00010000$. Par contre, si la valeur absolue maximale des paramètres est de 0.5, la valeur quantifiée sera $w_8 = 3 = b00000011$. Si une faute de type *bit-flip* est réalisée avec succès sur le bit de poids fort (MSB), dans le premier cas, le paramètre quantifié fauté sera $\tilde{w}_8 = -112$ soit $\tilde{w}_{32} = -0.0882$ après retour en précision 32 bits et, dans le second cas, nous aurons $\tilde{w}_8 = -125$ soit $\tilde{w}_{32} = -0.4921$. Ainsi, dans le premier cas, la faute induit une différence de $\Delta w = 0.1005$ pour un pas de quantification $p_q = 0.00078$ et, dans le second cas, la différence est beaucoup plus grande, avec $\Delta w = 0.5044$ pour un pas de quantification $p_q = 0.0039$. L'erreur Δw induite par la faute sera d'autant plus importante que la dispersion des paramètres de la couche est élevée.

C'est précisément l'influence de la dispersion des paramètres qui soulève plusieurs interrogations par rapport à [75]. Dans le répertoire public associé [3], les auteurs fournissent le modèle VGG-11 avec lequel ils réussissent à atteindre un objectif de *random-guess level* en seulement 10 *bit-flips*, alors que – dans le meilleur des cas de nos expériences – nous devons combiner 4 fois plus de fautes. En analysant ce modèle nous observons que la distribution des valeurs des para-

3. https://github.com/elliothe/BFA/issues/7

mètres est *très* particulière car fortement déséquilibrée : tous les paramètres sont (classiquement) très proches de zéro, à l'exception d'un seul paramètre par couche qui possède une valeur absolue proche de 3. La figure 4.2 représente la distribution problématique du modèle dans [75] (à gauche) ainsi que celle d'un de nos modèles utilisés dans nos expériences (à droite)[4]. Si nous n'avons pas d'explication quant à cette distribution, il est clair qu'en *surestimant* la borne maximale définissant le pas de quantification, elle va grandement favoriser l'effet de l'attaque.

4.1.3.2 Influence du *learning rate*

La dispersion des paramètres dépend sensiblement de leur optimisation pendant l'apprentissage (équation 2.22, page 20). L'hyperparamètre le plus important étant le *learning rate*, λ, nous évaluons son influence en utilisant deux valeurs initiales : 0.1 et 0.01. Les poids sont initialisés à l'aide d'une distribution normale.

La figure 4.5 montre l'impact de λ pour VGG-11 et ResNet-20. Nous n'observons aucune influence de λ sur ResNet-20, quel que soit l'objectif de l'adversaire. Pour VGG-11, $\lambda = 0.01$ fournit plus de robustesse lorsque l'objectif est de faire chuter la précision en dessous de 40% (une différence de près de 15 *bit-flips* est nécessaire pour atteindre 20%).

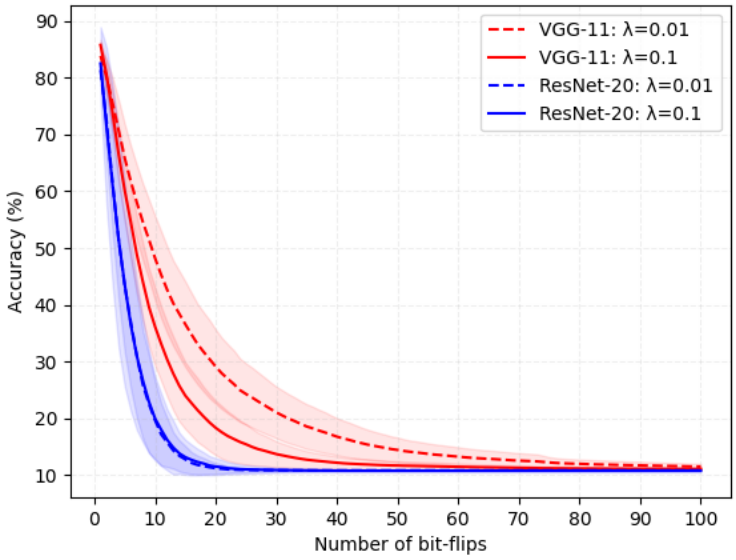


Figure 4.3 – Influence du *learning rate* λ sur la BFA (VGG-11 et ResNet-20).

Ce résultat s'explique par le fait que, pour les CNNs, la BFA cible principalement les paramètres en début de modèle. Les paramètres d'un CNN étant partagés, une faute est accumulée sur l'ensemble du tenseur d'entrée. Or, pendant la rétropropagation des gradients et l'optimisation des paramètres (équation 2.22, page 20), comme les valeurs des gradients et des paramètres sont proches de 0, si λ est trop faible, seuls les paramètres des dernières couches vont évoluer. Une valeur de λ plus grande permet de mettre à jour plus efficacement les paramètres des premières couches.

Il est donc probable que l'architecture du modèle ait une forte influence sur la BFA, ce que nous étudions par la suite.

4.1.3.3 Influence de l'architecture.

Cas du ResNet par rapport à VGG-11. Le tableau 4.6 et la figure 4.3 sur l'impact du *learning rate* ont montré des différences entre VGG-11 et ResNet-20. Pour l'objectif à 11%, il faut combiner beaucoup moins de *bit-flips* pour ResNet-20 que pour VGG-11 et les profils d'attaque pour les deux valeurs de λ sont sensiblement les mêmes pour ResNet-20. Les tableaux 4.7 et 4.8 présentent la localisation des *bit-flips* entre les deux modèles et les figures 4.4a et 4.4b la distribution des gradients ($\nabla_w \mathcal{L}$). Dans les tableaux 4.7 et 4.8, le terme "dommage" associé à une couche cor-

4. Tous nos modèles ont des distributions similaires, sans *outliers*.

respond à la perte cumulée d'*accuracy* avant et après les fautes sur cette couche (la somme sur toutes les couches vaut 100%).

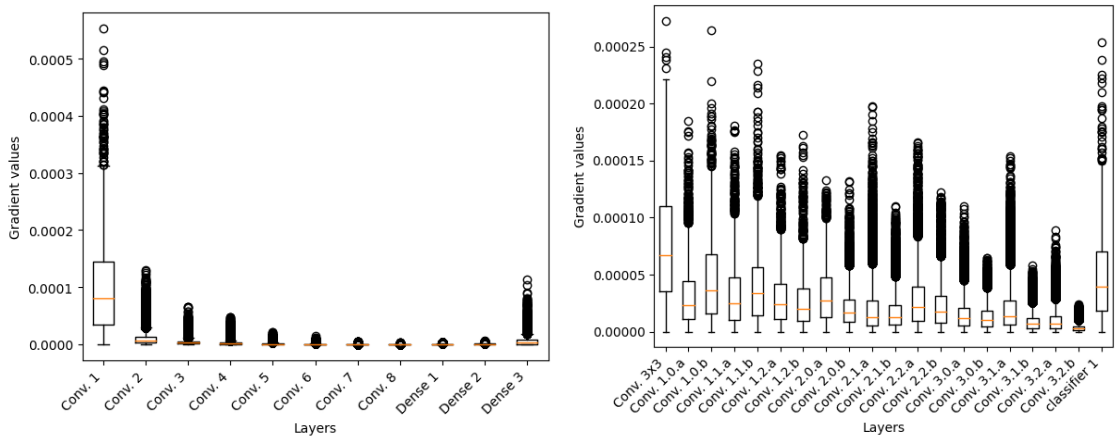
Table 4.7 – ResNet-20 : Distribution des bit-flips et dommages (contribution).

Couche	Dist. bit-flips (%)	Dist. dommages (%)
Conv. 1	**21.35**	**23.87**
Étage 1.0	14.16	21.49
Étage 1.1	11.76	27.0
Étage 1.2	2.4	0.61
Étage 2.0	8.5	7.12
Étage 2.1	8.28	7.41
Étage 2.2	7.18	6.01
Étage 3.0	3.27	0.73
Étage 3.1	3.49	0.2
Étage 3.2	14.81	4.62
Dense 1	4.80	0.95

Table 4.8 – VGG-11 : Distribution des bit-flips et dommages (contribution).

Couche	Dist. bit-flips (%)	Dist. dommages (%)
Conv. 1	**22.27**	**50.71**
Conv. 2	15.38	25.64
Conv. 3	14.47	10.75
Conv. 4	13.89	5.11
Conv. 5	12.94	4.9
Conv. 6	6.24	0.89
Conv. 7	2.06	0.01
Conv. 8	0	0
Denses	12.75	1.99

Tout d'abord, les deux modèles concentrent les gradients ($\nabla_w \mathcal{L}$) les plus élevés en début de modèles. Rappelons que, pour une couche de convolution, les paramètres sont ceux des filtres de convolutions et sont partagés et beaucoup moins nombreux que pour un modèle MLP classique. Comme mentionné dans [6] ou [75], l'erreur induite par les *bit-flips* effectués sur les premières couches de convolution est accumulée (puisqu'un paramètre est partagé sur l'intégralité du tenseur d'entrée) et propagée à travers le réseau. Cette accumulation et propagation de l'erreur (proche du phénomène de diffusion de la perturbation pour les *adversarial examples*) explique la forte efficacité de la BFA sur les CNNs. Néanmoins, il y a malgré tout une réelle différence de distribution entre les deux modèles, avec une distribution beaucoup plus diluée sur l'ensemble du modèle des fautes ainsi que des valeurs des gradients pour le ResNet par rapport à VGG-11.



(a) VGG-11 ($\lambda = 0.01$)

(b) ResNet-20 ($\lambda = 0.01$)

Figure 4.4 – Distribution (par couche) des valeurs absolues des gradients pour VGG-11 et ResNet-20.

L'explication tient dans la particularité des modèles de la famille ResNet, à savoir les connections résiduelles (ou *skip connections*) qui permettent de limiter la décroissance des valeurs des gradients le long du modèle. Ceci implique que les gradients en début de modèle sont plus importants (et plus homogènes par rapport à ceux en fin de modèle) que pour VGG-11. Avec la rétropropagation des gradients, cela implique aussi que les paramètres en début de modèle vont avoir une meilleure dispersion de leurs valeurs.

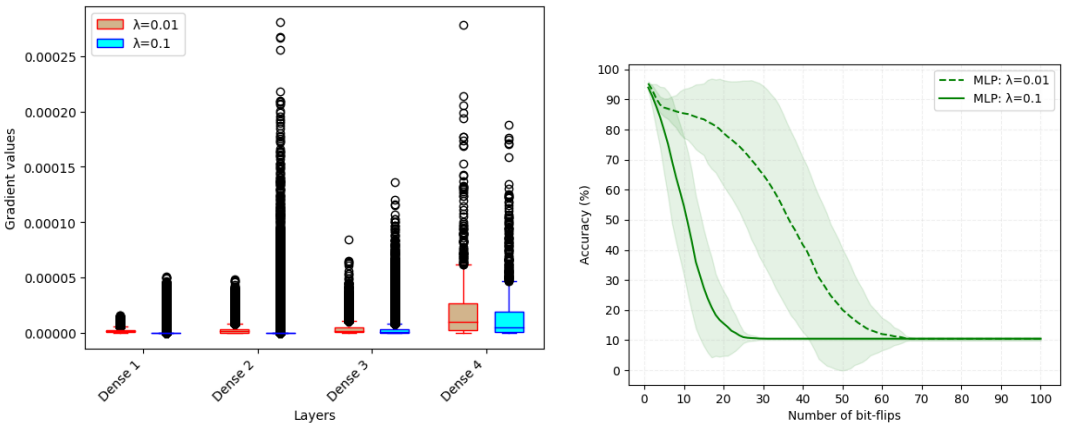
Multilayer Perceptrons. Les modèles MLP sont totalement exclus des analyses de l'état de l'art, alors même qu'ils constituent une famille importante en IA et une brique de base essentielle

pour les modèles modernes comme les architectures Transformer. Plus précisément, nous supposons qu'en utilisant des paramètres qui ne sont pas partagés, les modèles MLP doivent avoir une distribution de gradients sensiblement différente par rapport aux modèles CNNs, mais aussi un comportement différent vis-à-vis de la BFA en fonction du *learning rate*. Pour nos expériences, nous utilisons le modèle MLP décrit dans la section 4.1.1 (tableau 4.1) entraîné sur MNIST.

Comme observé dans la figure 4.5a, la distribution des gradients est significativement différente pour le MLP, avec la majorité des gradients les plus élevés situés à la fin du modèle. Une autre conséquence importante est liée à la relation entre la valeur du gradient de la *loss* par rapport à un paramètre d'une couche l et les valeurs des paramètres des couches ultérieures. Cette relation est donnée dans l'équation 4.1 :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w_{i,j}}} = a_i \rho_j, \text{ avec : } \rho_j = \Big(\sum_{k \in (l+1)} \mathbf{w_{j,k}} \rho_k \Big) \phi'(o_j) \tag{4.1}$$

où $w_{i,j}$ est le paramètre qui connecte le neurone i de la couche $(l-1)$ et le neurone j de la couche l, a est l'activation correspondant à la sortie o du neurone par la fonction d'activation ϕ : $a_j = \phi(o_j)$ avec $o_j = \sum_k w_{k,j} a_k$. Une inversion de bit sur $w_{j,k}$ modifiera la valeur de $\nabla_{w_{i,j}} \mathcal{L}$. Ainsi, un changement de la valeur du paramètre (consécutif à un *bit-flip*) dans une couche l altère directement la valeur des gradients des couches précédentes. Ce phénomène ne se produit pas (ou est limité) pour des fautes ciblant les premières couches, ce qui n'est pas le cas pour le MLP puisque les gradients les plus élevés se trouvent à la fin du réseau.



(a) Distribution des gradients avec deux λ, MLP (MNIST).

(b) MLP entraîné sur MNIST

FIGURE 4.5 – Performance de la BFA pour le modèle MLP avec deux *learning rate*.

Cette différence de comportement entre le MLP les CNNs est particulièrement forte quand on analyse l'influence du learning rate λ dans la figure 4.5b. Pour le MLP, on observe une différence d'environ 30 *bit-flips* entre les deux λ pour atteindre le seuil de 40%. Si l'on analyse la distribution des *bit-flips* et des gradients (Tableau 4.9 et figure 4.5a), on remarque que pour $\lambda = 0.01$, tous les *bit-flips* sont concentrés sur la dernière couche. La distribution est plus équilibrée pour $\lambda = 0.1$ puisque les gradients les plus importants sont étalés sur la seconde, troisième et la dernière couches (Cf. figure 4.5a).

TABLE 4.9 – Influence du *learning rate* (λ) : distribution des *bit-flips* (%) par couche (MLP) (objectif : *random-guess level*).

Layer	$\lambda = 0.01$	$\lambda = 0.1$
Dense 1	0	0
Dense 2	0	**56**
Dense 3	0	20
Dense 4	**100**	24

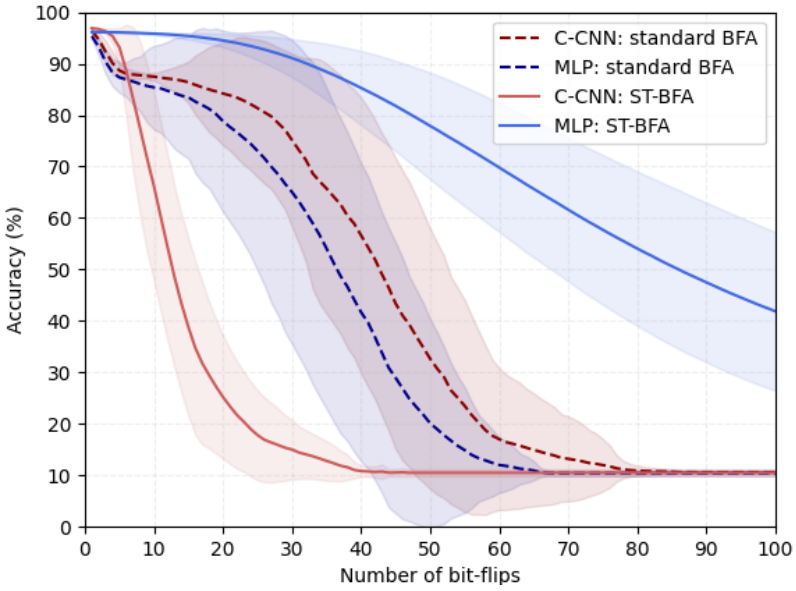


Figure 4.6 – Résultats des BFA (C-CNN et MLP)

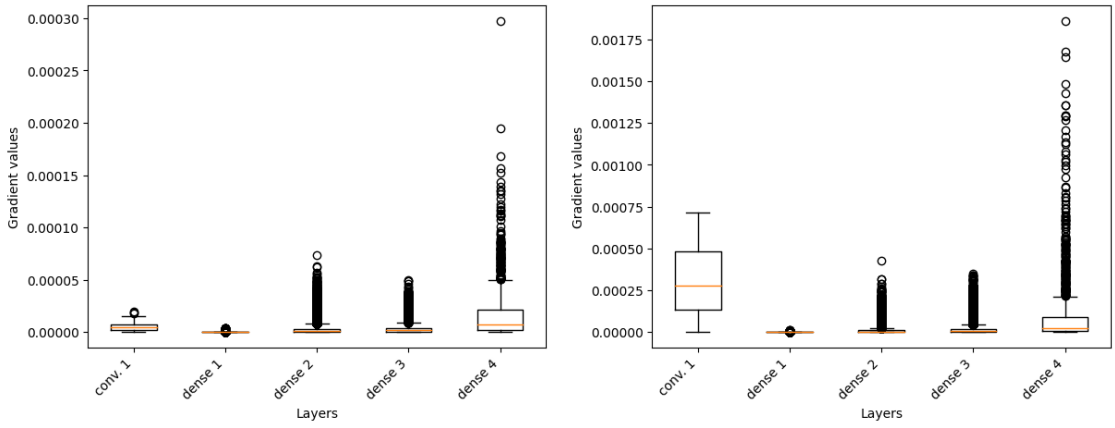
Architecture hybride et limitation de la BFA. L'analyse de la BFA sur notre modèle MLP (et le fait qu'une faute en fin de modèle se répercute sur les gradients des couches antérieures) soulève une question sur l'efficacité de la méthode de sélection des bits à fauter qui, de façon itérative, sélectionne les bits à fauter les uns après les autres. En effet, tant que les paramètres les plus sensibles sont situés dans la ou les premières couches, la méthode itérative de la BFA est pertinente, mais dès lors que ces paramètres se situent en fin de modèle, la méthode peut difficilement évaluer si une combinaison de *bit-flips* (qui pourrait bénéficier de la propagation d'erreur) est plus efficace qu'un seul *bit-flip* associé au gradient le plus élevé en fin du modèle.

Pour démontrer et illustrer cette possible limitation, nous ajoutons simplement une couche de convolution au début de notre modèle MLP pour former le modèle C-CNN (tableau 4.2). C-CNN possède une distribution de gradients très proche de celle du MLP original (comme le montre les figures 4.7a et 4.5a). Les performances de la BFA sur MLP et C-CNN sont aussi similaires comme le montre les courbes pointillées de la figure 4.6, avec des fautes se concentrant presque exclusivement sur la dernière couche.

Néanmoins, puisque l'on a ajouté une couche de convolution dont on connaît le pouvoir d'accumulation des fautes, on peut contraindre la BFA à se concentrer uniquement sur cette couche additionnelle. Nous appelons cette attaque contrainte ST-BFA (pour *Spatially Targeted BFA*) qui correspond aux courbes continues (bleu et rouge) dans la figure 4.6. Le résultat est étonnant : pour le modèle C-CNN, la ST-BFA est bien plus efficace que la BFA, puisqu'on observe une différence de plus de 50 *bit-flips* pour atteindre un objectif de 20% d'*accuracy*.

Il est très intéressant de constater le comportement respectif de la BFA et de la ST-BFA pour les premières fautes. Pour les cinq premières, la BFA est effectivement plus efficace mais cette tendance s'inverse grâce à l'effet cumulatif de la couche de convolution. On observe cette inversion des gradients dans la figure 4.7b qui montre la distribution des gradients après 10 *bit-flips* ciblant exclusivement la dernière couche. On voit à quel point les fautes modifient les gradients des autres couches avec une augmentation significative de ceux de la première couche de convolution. Plus particulièrement, la valeur moyenne des gradients de *conv1* est multipliée par 25 après 10 fautes (voir le niveau des gradients de *conv1* dans les figures 4.7a et 4.7b).

Cette expérience démontre une réelle difficulté pour l'évaluation de la BFA puisqu'évaluer la robustesse d'un modèle comme C-CNN avec la BFA standard conduit à un niveau de sécurité faussement élevé. Il suffit de modifier très simplement l'attaque pour la rendre sensiblement plus efficace. A l'inverse, la ST-BFA est inadaptée pour le modèle MLP classique (courbes bleues). Une évaluation rigoureuse de la robustesse d'un modèle pour des attaques de type BFA doit donc reposer sur une analyse fine de la distribution des gradients et de la nature des couches afin de déterminer si des attaques adaptées aux caractéristiques du modèle doivent être aussi évaluées en complément de la BFA standard.



(a) C-CNN ($\lambda = 0.01$)

(b) C-CNN ($\lambda = 0.01$), après 10 *bit-flips*.

FIGURE 4.7 – Distribution des gradients pour C-CNN avant le 1^{er} (à gauche) et après le 10^{ime} (à droite) *bit-flip*. Les 10 *bit-flips* ciblent tous la dernière couche (*dense 4*). Après 10 *bit-flips*, les gradients de la couche de convolution sont significativement plus élevés.

4.2 Évaluation pratique sur microcontrôleur avec injection de fautes laser

4.2.1 Positionnement

Dans la section précédente, nous avons adopté la perspective d'un concepteur de système de ML cherchant à évaluer la robustesse d'un DNN face à des perturbations, telles que celles engendrées par la Bit-Flip Attack. Cette évaluation reposait sur une série de simulations, au cours desquelles différents paramètres liés à l'attaque ou au DNN (notamment son architecture, les hyperparamètres d'apprentissage) étaient modifiés. Ici, nous considérons une évaluation pratique faisant intervenir une technique d'injection de fautes correspondant à la cible sur laquelle est déployée le modèle. Le contexte est différent puisque l'on se place dans le cas d'une évaluation sécuritaire qui, par exemple, serait effectuée dans un laboratoire de certification. Par conséquent, nous nous positionnons du point de vue d'un *évaluateur* et non d'un *adversaire*. Nous considérons un réseau de neurone profond embarqué dans un microcontrôleur 32 bits de la famille Cortex-M. Notre méthode d'injection de fautes est un laser (LFI) qui est un moyen d'injection avancé (spatialement et temporellement très précis) et une technique de référence utilisée dans de nombreux centres d'évaluation de sécurité.

Ces expériences sont les premières à démontrer la faisabilité et l'adéquation de la caractérisation d'une perturbation adverse contre les paramètres contre des microcontrôleurs Cortex-M grâce à l'injection de fautes laser.

4.2.2 Objectifs et hypothèses de l'évaluateur

L'objectif principal est d'évaluer la robustesse d'un modèle face à des injections de fautes précises et réelles (i.e., non simulées) qui cherchent à diminuer la précision moyenne sur un ensemble de test. Un objectif secondaire est de minimiser le coût de l'évaluation grâce à une stratégie réduisant le nombre de fautes à injecter, c'est-à-dire éviter une recherche exhaustive qui pourrait être irréaliste selon la complexité du modèle cible. Classiquement, pour les tests de sécurité, l'évaluateur simule un *worst-case* scénario, avec un adversaire qui dispose d'une connaissance parfaite du modèle (attaque en boîte blanche) et peut interroger le modèle sans limitation. L'évaluateur a un accès complet à la cible et peut effectuer des caractérisations élémentaires pour adapter et optimiser la configuration de l'injection de fautes.

4.2.3 Cible, setup et modèle de faute

Notre carte cible intègre un ARM Cortex-M3 fonctionnant à 8 MHz et comprenant 128 kB de mémoire Flash. Les dimensions de la puce sont de 3×2.5 mm. L'injection laser nécessite un accès direct à la surface sensible du circuit, aussi le boîtier du microcontrôleur a été mécaniquement

"ouvert" (gravure). La cible a ensuite été montée sur une carte de test compatible avec la plateforme ChipWhisperer CW308[5]. Notre plateforme d'analyse par injection de fautes laser est la même que celle décrite dans la section 3.6.2.2.

Nous considérons un modèle de faute *bit-set* pour l'injection laser, précédemment expliqué et démontré dans [88] pour les mémoires NOR-Flash des microcontrôleurs Cortex-M. La faute force un bit ciblé à une valeur logique 1 : lorsque le bit est déjà à 1, la faute n'a aucun effet. Lorsqu'une mémoire Flash est fautée par une impulsion laser en phase de lecture, le *bit-set* induit est transitoire : il affecte les données lues au moment du tir, tandis que la valeur stockée reste inchangée. L'injection laser permet une faute très localisée dans la mémoire flash : son effet est limité à la ligne de bits situé dans la zone du faisceau laser et, selon le diamètre du faisceau laser : jusqu'à deux bits adjacents peuvent être simultanément défaillants [88].

4.2.4 Datasets et modèles

Pour répondre aux contraintes de notre cible, nous nous restreignons à des modèles MLP et considérons la base de données IRIS, composée de 150 échantillons, chacun contenant 4 valeurs réelles représentant 3 types différents de fleurs ($|Y| = 3$). Pour IRIS, nous créons deux modèles simples pour nos premières caractérisations élémentaires : le modèle `IRIS_A` est simplement composé d'un seul neurone (avec donc 4 paramètres) et le modèle `IRIS_B` avec une couche de 4 neurones. Seul le modèle `IRIS_B` est fonctionnel et atteint une précision de 96% sur l'ensemble de test. Enfin, nous utilisons MNIST avec un modèle, simplement noté `MNIST`, composé d'une couche cachée de 10 neurones. Les entrées sont compressées sur $\mathbb{R}^{50}$ grâce à une analyse en composantes principales (PCA) classique. Le modèle possède 620 paramètres et atteint une *accuracy* de 92% sur l'ensemble de test. Tous les modèles utilisent ReLU comme fonction d'activation.

4.2.5 Implémentation du modèle sur MCU

Comme pour les expériences du chapitre 3 (3.6.2.2, page 49), après apprentissage sur TensorFlow, les modèles sont déployés avec la librairie NNoM avec une quantification sur 8 bits. NNoM permet un accès complet au code d'inférence nous permettant d'insérer des *triggers* pour le déclenchement des tirs laser (lecture des valeurs depuis la mémoire Flash).

```
1  while (rowCnt){
2      //pA : adresse, entree
3      //pB : adresse, parametre
4      for (int j = 0; j < dim_vec; j++){ //boucle sur les parametres
5          q7_t inA = *pA++;   //chargement de l'entree dans inA, increment de l'adresse
6          q7_t inB = *pB++;  //chargement du parametre dans inB, increment de l'adresse
7          ip_out += inA * inB; //somme ponderee
8      }
9      *pO++ = (q7_t)__NNOM_SSAT((ip_out >> out_shift), 8);
10     rowCnt--;}
```

Listing 4.1 – Code C de la somme pondérée dans une couche dense.

Le code C dans le Listing 4.1 est un extrait du calcul de base d'une inférence provenant de NNoM, à savoir la somme pondérée entre les entrées d'une couche et les paramètres du modèle, avant l'activation. Le code débute par le chargement des valeurs d'entrée et de paramètres (`inA` et `inB`, lignes 5 et 6), puis on effectue la multiplication et l'accumulation dans une valeur intermédiaire (`ip_out`, ligne 7). La ligne 9 correspond à la quantification. Le code assembleur correspondant (Listing 4.2) représente l'initialisation du paramètre dans la ligne 6 du code C. La valeur du paramètre, stockée dans la Flash, est chargée dans le registre `r3` (ligne 5).

```
1 ;q7_t inB = *pB++       ;initialisation du parametre
2 ldr     r3, [r7, #80]   ;Chargement de l'adresse du parametre
3 adds    r2, r3, #1      ;Prochaine adresse du parametre
4 str     r2, [r7, #80]   ;Chargement de la valeur d'entree dans le registre r2
5 ldrsb.w r3, [r3]        ;Chargement du parametre. TIR LASER
6 strb    r3, [r7,#23]    ;Enregistrement du parametre en SRAM
```

Listing 4.2 – Code assembler de la ligne 6 du code C, on cible l'instruction load, ligne 5.

5. https://rtfm.newae.com/Targets/CW308%20UFO/

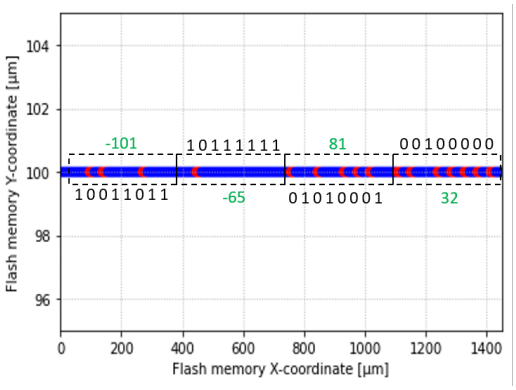
En se basant sur notre modèle de faute *bit-set*, si un faisceau laser est appliqué pendant l'exécution de l'instruction de chargement, une faute bit-set pourrait être induite directement sur la valeur chargée. Pour caractériser l'impact de l'impulsion laser, nous synchronisons le laser grâce à un signal de déclenchement dans le code C avant la ligne 6 et surveillons la valeur du paramètre avant et après le déclenchement à l'aide d'une communication UART.

Pour analyser l'efficacité et la faisabilité de l'injection de fautes par laser, nous avons d'abord démontré la précision des fautes induites sur un seul neurone composé de quatre poids. Ensuite, nous avons étendu l'analyse à des modèles fonctionnels formés sur IRIS et MNIST pour étudier l'impact au niveau du modèle.

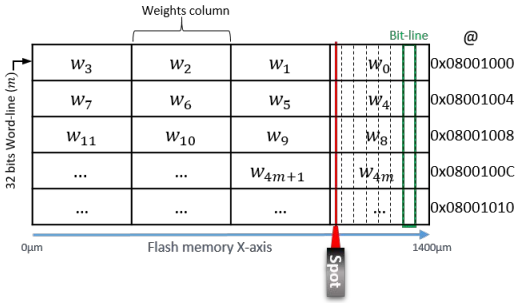
4.2.6 Caractérisation initiale sur un neurone

Pour cartographier la mémoire, nous suivons le protocole expérimental de [88] et fixé la puissance du laser à 170 mW et la largeur de l'impulsion à 200 ns. En choisissant l'objectif ×5, nous optons pour un diamètre de spot de 15 µm afin d'avoir une large zone d'effet du laser.

Ensuite, nous implémentons un neurone avec 4 paramètres (`IRIS_A`) et fixons la position du laser en $Y = 100$ µm, le laser se déplacera uniquement selon l'axe X de la Flash. La figure 4.8a montre que, lors du déplacement du laser, des fautes *bit-set* sont induites les unes après les autres sur l'ensemble de la ligne d'un mot de 32 bits et que tous les paramètres sont fautés avec précision. Nous répétons cette expérience avec différentes valeurs de paramètres et observons une parfaite reproductibilité (comme rapporté dans [88]). En observant les positions des paramètres et de leur MSB, on peut facilement conclure à la configuration *little-endian* de la Flash. La figure 4.8b illustre la manière dont les paramètres sont stockés en fonction de la position X.



(a)



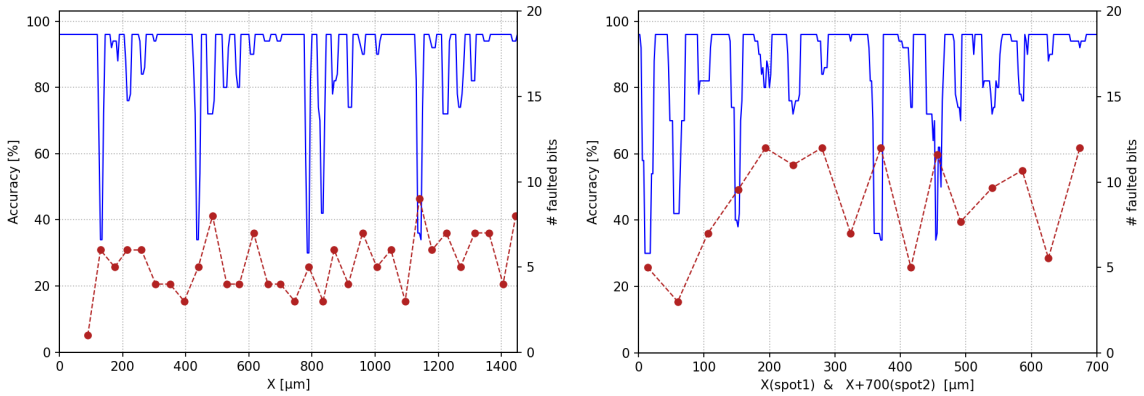
(b) m is the m^{th} word-line

FIGURE 4.8 – (a) Fautes *bit-sets* (points rouges) sur les 4 paramètres d'un neurone (modèle `IRIS_A`). (b) Schéma de l'organisation de la Flash avec les paramètres stockés.

4.2.7 Modèles de perceptron multicouche ciblés

Puisque nous pouvons modifier la valeur des paramètres associés à un neurone, l'étape suivante consiste à analyser un modèle cible entier. Pour cela, nous utilisons le modèle `IRIS_B`. Pour chaque position selon X, nous évaluons la robustesse du modèle sur 50 échantillons de test. Lors d'une inférence, toutes les instructions de chargement de paramètre déclenchent un tir laser (i.e., 40 tirs au total pour la couche cachée). En ciblant une ligne de bits à une position X donnée du laser, seuls les paramètres de la même colonne d'adresse sont fautés par un *bit-set*. À titre d'illustration, comme le montre la figure 4.8b, l'injection laser a effectivement induit des *bit-sets* dans les bits de poids forts des paramètres w_0, w_4, w_8, w_{4m} (avec m le m^{ime} mot de la ligne), qui appartiennent à la ligne de bits ciblée.

La figure 4.9a montre l'impact des tirs laser sur la performance du modèle (courbe bleue) sur l'ensemble de test. La courbe rouge représente le nombre de bits fautés. Quatre motifs réguliers de chute de performance correspondant aux quatre paramètres stockés sont clairement visibles. Les chutes observées correspondent aux coordonnées des bits de poids forts, avec un impact inversement proportionnel par rapport à la position du bit. Pour les MSBs, le modèle est proche d'un niveau de prédiction aléatoire (i.e., 30%). En moyenne, environ 6 fautes sont induites et la configuration la plus performante correspond à seulement 5 *bit-sets* en $X = 790$ µm qui fait chuter le modèle à une *accuracy* de 30%.



(a) Injection laser, mono-spot.

(b) Injection laser, bi-spot.

FIGURE 4.9 – Évolution de la performance avec injection de fautes laser pour le modèle IRIS_B. *Accuracy* moyenne sur 50 inférences (bleu) et nombre de bits fautés (rouge).

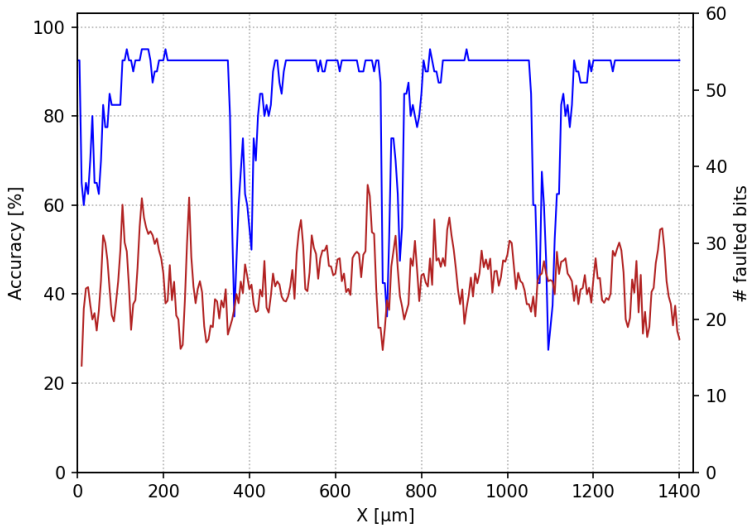


FIGURE 4.10 – Évolution de la performance avec injection de fautes laser pour le modèle MNIST. *Accuracy* moyenne sur 100 inférences (bleu) et nombre de bits fautés (rouge).

La principale limitation est que seulement un quart des paramètres peuvent être fautés lors d'une inférence. Cependant, comme notre plateforme laser intègre un deuxième spot, nous pouvons expérimenter une configuration où un spot cible la première ligne de bits (i.e., le MSB de la colonne de paramètres w_3 dans la figure 4.8b) et le deuxième spot cible la ligne de bits #16 (i.e., le MSB de la colonne de paramètres w_1). En conséquence, deux colonnes sont ciblées lors d'une inférence, et la moitié des paramètres sont susceptibles d'être fautés. Les résultats sont présentés dans la figure 4.9b : davantage de *bit-sets* sont injectées (jusqu'à 12), et la chute de performance pour les bits les moins significatifs est plus importante.

Pour étendre ces expériences, nous déployons le modèle MNIST qui comporte 620 poids (4,960 bits) en maintenant la même configuration expérimentale. Les résultats sont présentés dans la figure 4.10. En moyenne, nous observons plus de *bit-sets* induits. La chute de performance la plus importante (65,5%) est atteinte avec 25 *bit-sets* à $X = 1\,100\,\mu m$. Dans de rares cas, une légère amélioration de la précision apparaît, comme en $X = 1\,150\,\mu m$.

En comparaison avec IRIS_B (figure 4.9a), nous remarquons que les lignes de bits sont moins distinctes. En effet, même si la position des bits les plus significatifs peut être identifiée, d'autres bits sont difficiles à distinguer. En réalité, dans la figure 4.8b, la ligne de bits (boîte verte) représente toutes les lignes uniques connectées au même indice de bit pour différentes adresses de mots de 32 bits. Étant donné que le modèle MNIST comporte plus de paramètres, presque toutes les lignes de bits sont liées à une valeur de paramètre de poids stockée dans la mémoire Flash. Le spot laser, de taille 15 µm, est plus grand qu'une seule ligne de bits, ce qui explique pourquoi les indices des lignes de bits sont difficile à discerner. De plus, selon la position du laser, la zone effective du spot peut englober deux indices de ligne de bits différents.

4.2.8 Adapter la BFA à l'injection laser

Jusqu'à présent, nous avons attaqué de manière exhaustive tous les paramètres stockés en mémoire, mais cette stratégie *brute-force* peut s'avérer rédhibitoire pour des modèles plus complexes. Un objectif important pour un évaluateur est donc de sélectionner de manière optimale les bits les plus sensibles à fauter en priorité. Ainsi, comme dans [115], nous adaptons la BFA standard à notre modèle de faute et appelons cette attaque adaptée BSCA (pour *Bit-Set Constrained Attack*). Nous adaptons également l'objectif en utilisant un budget qui détermine le nombre maximal de fautes que l'évaluateur peut réaliser. Nous fixons le budget adverse à $S = 20$ *bit-sets*.

L'algorithme 2 (pseudo-code) détaille le fonctionnement de la BSCA qui prend en entrée le modèle cible M_W, ses paramètres W, l'ensemble des indices de colonne des paramètres $\mathcal{M}$, l'ensemble des indices de ligne de bits $\mathcal{B}$ et le budget S. En sortie, on obtient un nouveau modèle $M_{W'}^{m,b}$ avec W' et W qui diffèrent par au maximum S *bit-sets*. Comme pour la BFA, on commence par trier les bits selon $\nabla_b \mathcal{L}$ (ligne 5) mais on ne conservant que ceux à 0 (i.e., compatible avec un *bit-set*) et associés à m et b (ligne 6). Le meilleur *bit-set* correspond à la chute d'*accuracy* (*Acc*) la plus importante évaluée sur un ensemble de test $\left(X^{test}, Y^{test}\right)$ (ligne 7) et cette faute est appliquée au modèle (ligne 8). Le processus s'arrête, pour $m \in \mathcal{M}$ et $b \in \mathcal{B}$, une fois atteint le budget de S fautes. Ensuite, nous calculons l'*accuracy* sur $M_{W'}^{m,b}$. Ceci est appliqué sur toutes les colonnes m de paramètres et lignes de bits b et nous gardons le modèle fauté, simplement appelé $M_{W'}$, qui possède la performance la plus basse, évaluée sur $\left(X^{test}, Y^{test}\right)$. Le modèle final correspond donc à la colonne m^* et ligne de bits b^* : $m^*, b^* = \text{argmin}_{m,b}\, Acc(M_{W'}^{m,b}, X^{test}, Y^{test})$.

Algorithm 2 BSCA (*Bit-Set Constrained Attack*)

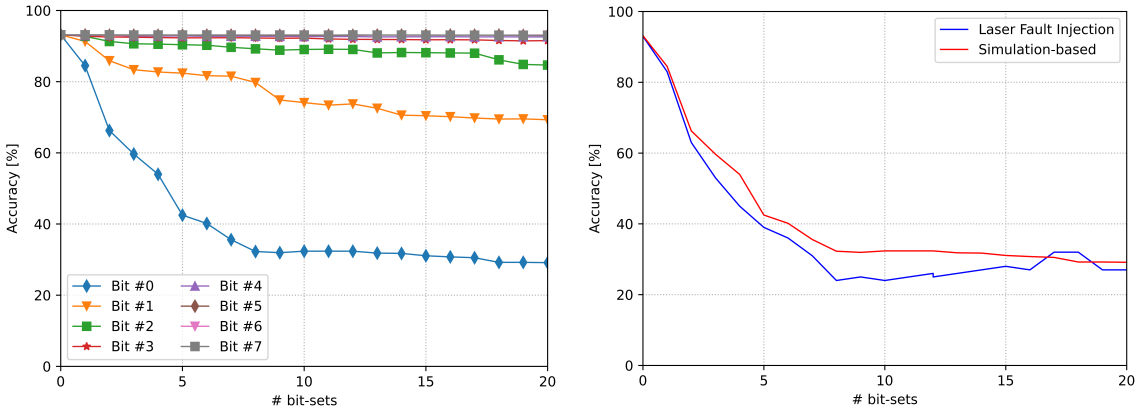
Input: Modèle cible M_W, ses paramètres W, ensemble des indices de colonne des paramètres $\mathcal{M}$, ensemble des indices de ligne de bits $\mathcal{B}$ et le budget S
Output: Modèle fauté $M_{W'}^{m,b}$

1: **for all** $m \in \mathcal{M}$ **do**
2: **for all** $b \in \mathcal{B}$ **do**
3: $W' \leftarrow W, Nb = 0$
4: **while** $Nb < S$ **do**
5: Rank bits with gradients(W') ▷ Tri des bits de W selon $\nabla_b \mathcal{L}$
6: Filter bits(m, b) ▷ Conserver les bits à 0 et associés à m et b
7: Pick best bit-set ▷ Sélection du meilleur *bit-set* selon la baisse d'*accuracy*
8: Apply Fault ▷ Application de la faute de façon permanente dans W'
9: $Nb \leftarrow Nb + 1$
10: **return** $M_{W'}^{m,b}$
11: **return** $m^*, b^* = \text{argmin}_{m,b}\, Acc\left(M_{W'}^{m,b}, (X^{test}, Y^{test})\right)$ et $M_{W'}^{m^*,b^*}$

Avec `MNIST`, nous simulons la BSCA pour trouver les 20 paramètres les plus significatifs à fauter, pour chaque colonne de paramètres. La figure 4.11a illustre l'effet des *bit-sets* induits sur les 8 bits d'un paramètre, uniquement pour la deuxième colonne de paramètres (le MSB à l'indice Bit #0). Parmi les paramètres les plus sensibles de la deuxième colonne, seules quelques fautes sur les bits les plus significatifs modifient efficacement la performance du modèle.

Expériences et résultats. L'idée est d'utiliser la BSCA pour guider l'injection de fautes laser. Pour cela, nous devons d'abord nous assurer que l'injection laser et la simulation par BSCA possèdent des performances quasi identiques. Nous exécutons donc une simulation BSCA (Python) sur toutes les colonnes de paramètres et lignes de bits. Le résultat de cette simulation est que le MSB de la deuxième colonne est le plus sensible. Par conséquent, contrairement aux expériences précédentes (*brute-force*), le tir laser intervient uniquement lorsque les 20 poids les plus sensibles sont lus depuis la Flash. La localisation du laser est définie en conséquence, $X = 760\,\mu m$, ainsi que la puissance du laser que l'on augmente à 360 mW pour garantir un taux de réussite plus élevé.

La courbe bleue dans la figure 4.11b représente les résultats expérimentaux (*accuracy* moyenne sur 100 inférences), tandis que la courbe rouge correspond aux simulations BSCA pour le MSB. Tout d'abord, nous pouvons constater que les résultats expérimentaux et les simulations sont presque identiques, ce qui signifie que nous pouvons guider notre injection laser avec une grande fiabilité et confiance. Le fait que les résultats expérimentaux montrent une diminution légèrement plus forte que les simulations s'explique par l'impact de la largeur du spot laser sur les cellules mémoire voisines. Pour un budget de seulement 5 fautes *bit-set* (0,1% des bits fautés), la



(a) Influence de l'indice du bit fauté pour la deuxième colonne de paramètres (simulation).

(b) Injections laser et simulations en ciblant les 20 MSB les plus sensibles pour la 2^{ime} colonne.

FIGURE 4.11 – Injections laser guidées par simulation sur le modèle `MNIST`.

performance du modèle chute à 39% (par rapport à sa performance nominale de 92%). Après 10 *bit-sets* ($Acc = 25\%$), les fautes les plus efficaces ont déjà été injectées et la performance ne diminue plus. Dans un contexte d'évaluation sécuritaire, cette observation permet de positionner le niveau de robustesse du modèle en fonction du budget adverse.

Nous discutons de l'ensemble de ces résultats dans la partie suivante.

4.3 Conclusion et discussions

Recommandations et bonnes pratiques pour l'évaluation de la BFA. Nos analyses présentées dans la section 4.1 montrent que l'évaluation de la BFA est une étape complexe qui peut conduire à un faux sentiment de sécurité du modèle. Nous récapitulons ci-dessous les principales recommandations destinées à aider les concepteurs à évaluer plus efficacement la résistance de leurs modèles face à une attaque de type BFA. Ces *bonnes pratiques* peuvent également servir aux évaluateurs en sécurité pour la conduite d'analyses de risques comme pour la formulation de recommandations en matière de schémas de défense.

— *Variabilité de l'attaque et modèle de menace.* Une évaluation correcte de l'efficacité de la BFA nécessite de réaliser plusieurs attaques en faisant varier l'ensemble d'attaques et, si cela est possible, en utilisant plusieurs instances du modèle (initialisation des paramètres). Pour éviter un effet plateau qui va entraîner une surévaluation du nombre de fautes à réaliser, il est nécessaire d'analyser l'efficacité de l'attaque en utilisant plusieurs budgets de l'attaquant, adaptés en fonction des techniques d'injection de fautes utilisées et des propriétés du système attaqué. L'utilisation de courbe indiquant la performance du modèle en fonction du nombre de fautes est un outil d'analyse indispensable pour comprendre la robustesse d'un modèle et, le cas échéant, le niveau de protection qu'il est nécessaire d'apporter.

— *Influence des paramètres d'apprentissage.* Nous avons montré que l'architecture du modèle a une importance fondamentale dans le comportement de la BFA, notamment vis-à-vis de certains hyper-paramètres d'entraînement. La nature du modèle étudiée et son contexte d'apprentissage sont donc des éléments fondamentaux à prendre en compte pour l'évaluation, plus particulièrement pour l'évaluation d'une protection.

— *Outils d'analyse.* Nous mettons aussi en avant que l'étude des distributions des paramètres et des gradients est indispensable pour comprendre l'efficacité de la BFA, surtout pour des architectures qui ne sont pas des CNNs standards, et proposer des adaptations de l'attaque (comme la ST-BFA avec notre modèle hybride C-CNN) qui reflètent mieux les faiblesses d'un modèle.

Caractérisation par injection laser. Nos premières expériences sur microcontrôleur par injections laser contribuent aux efforts pour développer des protocoles et des outils d'évaluation fiables pour la robustesse des modèles de réseaux de neurones embarqués. Rappelons que cette préoccupation est importante dans le contexte actuel des politiques de régulation et standardi-

sation en Europe et les réflexions sur les futurs schémas de certification des systèmes critiques basés sur l'IA. Il est intéressant de constater que nos résultats sont moins limités par la complexité du modèle [6] que par la complexité des mémoires que l'on cible. En effet, le principal défi est de caractériser finement l'organisation de la mémoire Flash pour cibler précisément les colonnes de paramètres et les lignes de bits. Par conséquent, il est important de considérer d'autres expériences ciblant d'autres plateformes de type microcontrôleur à base de mémoire Flash. Un autre point important est que, contrairement aux attaques par évasion, les attaques contre les paramètres manquent encore de maturité et qu'il est donc nécessaire de réfléchir à des mécanismes d'attaques plus performant que la BFA [114] qui pourront être réellement évalués. Ce constat s'élargit aux mécanismes de défense qu'il est nécessaire d'évaluer en pratique, par exemple les contre-mesures génériques traditionnelles contre l'injection de fautes [82], le *clipping* de paramètres et le renforcement de modèle [114], la quantification adaptative par *clustering* [140] ou les approches par détection [130] (Cf. section 3.6.1.3, page 47).

6. La BFA a été démontrée sur des modèles profonds de l'état de l'art avec des millions de paramètres [115].

Chapitre 5

Attaques physiques et extraction de modèle

Contents

Introduction. Parmi les menaces contre la confidentialité d'un système d'IA, celles portant sur les modèles ont pris de plus en plus d'importance, d'une part parce qu'il s'agit de répondre à des problèmes de propriétés intellectuelles (ou plus largement à la protection d'une expertise et d'un savoir-faire) et d'autre part, pour empêcher un adversaire à réaliser des attaques "boîtes blanches" plus puissantes. Nous avons discuté de cette menace d'*extraction de modèle* dans le chapitre 3 en insistant plus particulièrement sur un scénario de "fidélité" dans lequel l'objectif est de cloner un modèle. Dans ce cas de figure, un attaquant va chercher à extraire l'architecture *et* les valeurs des paramètres. Si plusieurs approches ont été proposées (apprentissage de modèle substitut, cryptanalyse), les attaques physiques, notamment les analyses *side-channel* (Cf. section 3.5, page 42) ont été récemment proposées contre des modèles embarqués sur différentes cibles (microcontrôleurs, FPGA...).

Organisation. Dans ce chapitre, nous proposons trois études qui mettent à profit des vecteurs d'attaque physique pour l'extraction de modèle. Les deux premiers travaux s'intéressent aux analyses *side-channel* d'inférences de réseaux de neurones profonds déployés sur un microcontrôleur ARM Cortex-M7. Nous montrons tout d'abord (5.1) que l'inférence de modèles standards, comme des MLPs ou des CNNs, fait intervenir des répétitions de blocs de calculs qu'il est aisé de détecter dans un nombre minimal de traces électromagnétiques. Ces informations permettent de déduire de nombreuses connaissances sur l'architecture d'un modèle. Ensuite, nous montrons dans la section 5.2 qu'une analyse *side-channel* plus complexe peut aussi extraire des informations sur la valeur des paramètres mais que plusieurs défis subsistent pour réellement passer à l'échelle de modèle de l'état de l'art. Enfin, dans la section 5.3, nous présentons une technique d'extraction de modèle qui mêle de l'injection de fautes et une méthode d'apprentissage de l'état de l'art d'un modèle substitut.

Projets, encadrements et publications associées. Comme pour le chapitre précédent, les travaux sur la confidentialité des modèles ont été réalisés dans le cadre des projets (Cf. section 1.3) :

- ANR PICTURE (AAPG 2020, 2021-2024, coordinateur)
- IRT NANOELEC, Programme PULSE (2021-2025, responsable *Implémentation Sécurisée d'IA*)
- EU ECSEL INSECTT (2020-2023, coordinateur France, responsable *AI Validation & Verification*)

Ces travaux ont été réalisées à travers l'encadrement de la thèse de **Raphaël JOUD** et de **Kevin HECTOR** (ANR PICTURE, 2021-2024) avec les collaborations de **Jean-Baptiste RIGAUD** (MSE), **Simon PONTIE** (CEA LETI) et **Jean-Max DUTERTRE** (MSE).

Une partie des résultats présentés dans ce chapitre ont été publiés dans les trois publications suivantes :

- [153] Joud, R., Moëllic, P. A., Pontié, S., Rigaud, J. B. (2022, November). A practical introduction to side-channel extraction of deep neural network parameters. In International Conference on Smart Card Research and Advanced Applications (CARDIS 2022), (pp. 45-65). Cham : Springer International Publishing [1].
- [106] Joud, R., Moëllic, P. A., Pontié, S., Rigaud, J. B. (2023, November). *Like an open book? Read neural network architecture with simple power analysis on 32-bit microcontrollers.* In International Conference on Smart Card Research and Advanced Applications (CARDIS 2023), (pp. 256-276). Cham : Springer International Publishing [2].
- [154] Hector, K., Moëllic, P. A., Dutertre, J. M., Dumont, M. (2023, September). Fault injection and safe-error attack for extraction of embedded neural network models. In European Symposium on Research in Computer Security. International Workshops, SECAI (pp. 644-664). Cham : Springer Nature Switzerland [3].

1. https://gitlab.emse.fr/securityml/SCANN-ex
2. https://gitlab.emse.fr/securityml/model-architecture-extraction
3. https://gitlab.emse.fr/securityml/lfi_sea_modelextraction

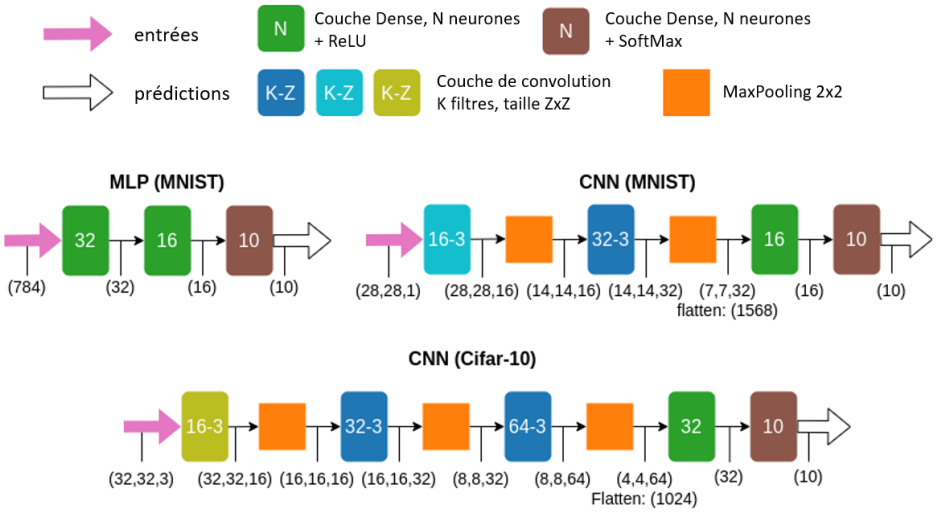


FIGURE 5.1 – Architecture des modèles utilisés.

5.1 Utilisation d'analyses side-channel pour l'extraction de l'architecture

Dans ce travail, nous nous intéressons à l'extraction d'architecture de modèles embarqués sur des microcontrôleurs 32 bits grâce à la bibliothèque ARM CMSIS-NN. Notre objectif est de déterminer jusqu'où un adversaire peut extraire des informations sur l'architecture d'un modèle victime en exploitant conjointement la connaissance de la bibliothèque de déploiement et un nombre limité de traces *side-channel* (une trace moyenne à partir d'une seule entrée). Nous montrons qu'avec des méthodes et une expertise classique en DL, presque toutes les informations et hyperparamètres les plus importants sont accessibles en raison de la forte *répétitivité* des calculs. Plus particulièrement, pour les CNNs, nous démontrons un effet *poupées russes* : un motif modèle EM régulier dans les traces est lié à un hyperparamètre et, en zoomant à l'intérieur de motif, on découvre d'autres motifs liés à d'autres hyperparamètres, et ainsi de suite.

5.1.1 Modèles et jeux de données

Nous utilisons le même formalisme que celui du chapitre 2 (page 19, équation 2.20), avec des entrées et tenseurs carrés ($H = W$). Nous utilisons MNIST et CIFAR-10. Nous avons donc $H_{in} = 28$ et $C_{in} = 1$ pour MNIST et $H_{in} = 32$ et $C_{in} = 3$ pour CIFAR-10. Pour MNIST, nous avons entraîné un MLP et un CNN avec la plateforme TensorFlow qui ont respectivement atteint 96% et 98% sur l'ensemble de test. Pour CIFAR-10, nous avons entraîné un CNN qui a atteint 76% sur l'ensemble de test. Les architectures de ces modèles sont illustrées dans la figure 5.1 où chaque type de couche a une couleur bien définie et la dimension des tenseurs est indiquée sous chaque couche. Les couches denses (verte) ont N_e neurones et les couches de convolution (bleue, olive et cyan) ont K noyaux de taille $Z \times Z$. Les différentes implémentations des couches de convolution sont détaillées dans la section 5.1.4.1. On utilise un *stride* standard de 1 et un padding de type *same* et uniquement la couche de Max pooling (orange) avec un noyau de taille 2 afin que le tenseur de sortie soit réduit de moitié. La fonction d'activation est ReLU, à l'exception de la dernière couche avec la fonction SoftMax.

On note $\mathcal{C}$ l'ensemble des couches considérées dans cette étude : $\mathcal{C} = \{$ *Dense*, *Convolution*, *MaxPooling* $\}$.

5.1.2 Plateforme matérielle et *setup* expérimental

Notre cible est un STM32H7 basée sur un ARM Cortex-M7 avec 2 MBytes de Flash et 1 MByte de SRAM, capable de supporter l'inférence de modèles MLPs et CNNs standards. Aucune ouverture (mécanique ou chimique) du MCU n'est effectuée. Nous mesurons les émissions EM provenant de la cible avec une sonde Langer [4] connectée à un amplificateur [5] puis enregistrées avec un oscil-

4. EMV-Technik LF-U 2,5
5. Fento HVA-200M-40-F, avec une bande passante de 200 MHz et un gain de 40 dB

loscope Lecroy[6]. L'attaquant cherche à récupérer l'architecture d'un modèle quantifié inconnu (M_W) avec autant de détails que possible. Nous notons $\mathcal{A}_M$ l'architecture du modèle M qui correspond à l'organisation de ses couches. $\mathcal{A}_M$ est définie par la nature de chaque couche, leurs connexions, leur taille et leurs hyper-paramètres (ceux non entraînables, par exemple le nombre de noyaux de convolution). Nous notons L le nombre de couches. Dans ce travail, nous considérons uniquement des modèles de type *feed-forward*.

Le tableau 5.1 répertorie toutes les informations que l'adversaire souhaite extraire grâce aux traces EM. Le contexte de l'attaque correspond à un cadre particulier de type boîte noire, car s'il ne connaît pas l'architecture ni les paramètres du modèle, il connaît la tâche effectuée ainsi que l'utilisation de la librairie de déploiement CMSIS-NN. Son expertise en DL lui permet de connaître les successions de couches les plus vraisemblables. De plus, l'attaquant a toute l'expertise nécessaire pour réaliser des campagnes *side-channel* mais, comme nous nous plaçons volontairement dans un cadre d'exploitation restreint, nous faisons l'hypothèse qu'il ne peut pas perturber l'exécution du programme d'inférence et peut collecter des traces provenant d'une seule entrée. Ainsi, il réalise plusieurs inférences à partir d'une seule entrée et exploite la moyenne des traces collectées.

Table 5.1 – Liste des hyper-paramètres à extraire pendant l'attaque. $\mathcal{C}$ = { *Dense*, *Convolution*, *MaxPooling* }.

Cible	Paramètres	Notation	Cible	Paramètres	Notation
$\mathcal{A}_M$	# couches	L	Couche Conv.	Dimension de sortie	H_{out}
	Type de couches	$\mathcal{C}$		# kernels	K
Dense Layer	# neurones	N_e		Taille des kernels	Z
MaxPool	Dimension de sortie	H_{out}		Stride, Padding	S, P
	Taille du filtre	Z_{pool}	Activation	ReLU ou non ?	Vrai/Faux

À l'échelle du modèle, l'inférence est un processus de type *feedforward* : le calcul de chaque couche est effectué l'un après l'autre. Le tenseur de sortie d'une couche devient le tenseur d'entrée de la suivante. Nous nous concentrons d'abord à l'échelle d'une couche en adoptant une méthodologie en deux étapes :

1. Nous analysons l'implémentation CMSIS-NN de chaque couche afin de révéler des répétitions de blocs de calcul susceptibles d'apparaître dans nos traces EM. Plus particulièrement, nous visons à établir un lien entre ces motifs dénombrables et les hyperparamètres de la couche.
2. Nous évaluons expérimentalement si ces hypothèses théoriques sont confirmées dans nos traces EM et analysons la complexité de l'extraction ainsi que des limites potentielles.

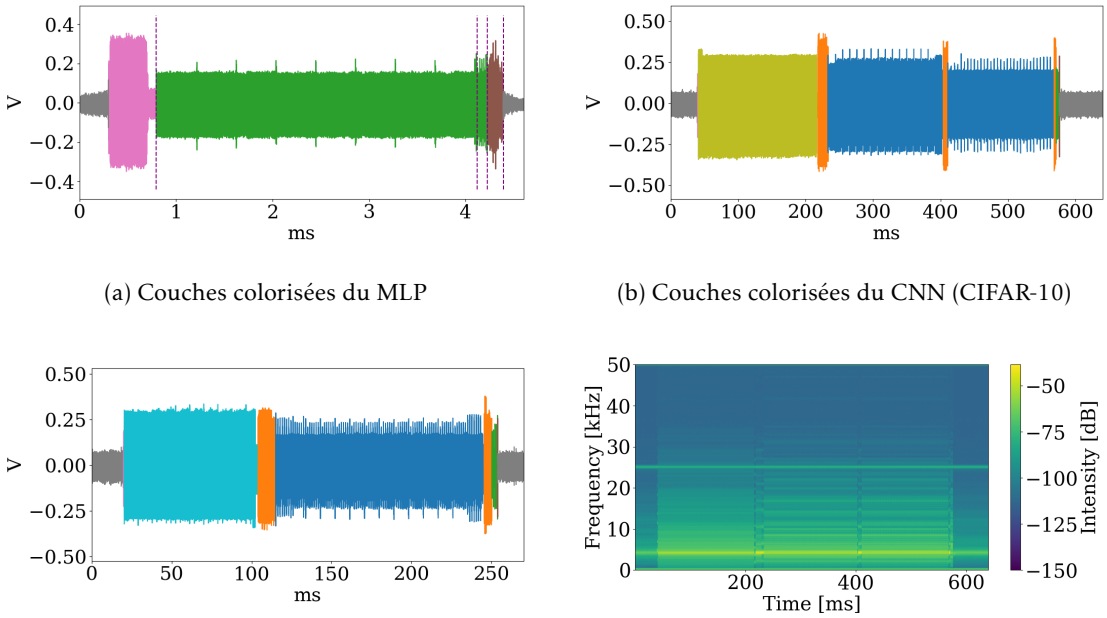
Notre approche repose sur le principe que l'attaque est réalisée en suivant le flux de calcul : lorsque l'adversaire cible la couche l, nous supposons que l'analyse de la couche $l-1$ est terminée et que la forme de son tenseur de sortie est donc connue, ce qui signifie que la forme du tenseur d'entrée de la couche l est également maîtrisée. Naturellement, nous supposons que la taille des entrées est connue de l'adversaire (par exemple, 28×28 pour MNIST et 32×32×3 pour CIFAR-10).

5.1.3 Méthodologie d'extraction de l'architecture

La première information exploitable par l'attaquant est la nature de la tâche réalisée par le modèle qui peut donner des indices sur le type d'architecture du modèle. Un modèle réalisant une tâche de détection ou de classification d'images sera probablement un CNN. La connaissance de la tâche peut fournir un ensemble de couches probables mais qui doivent être confirmées par une analyse préliminaire de la trace. De plus, l'expertise de l'attaquant lui permet de connaître des ordres logiques entre les couches. Par exemple, si la première couche est identifiée comme une couche de convolution, une association standard serait une autre couche de convolution ou une couche de *pooling*.

Le premier objectif est l'extraction de la structure globale $\mathcal{A}_M$, avec le nombre de couches, L, et leur nature. Ensuite, l'attaquant se concentre sur chaque couche, l'une après l'autre et, en fonction de leur nature, extrait un ensemble d'hyperparamètres facilitant la conception d'un modèle substitut, que l'on note M'_Θ.

6. WavePro 604HD-MS



(a) Couches colorisées du MLP

(b) Couches colorisées du CNN (CIFAR-10)

(c) Couches colorisées du CNN (MNIST)

(d) Spectogramme du CNN (CIFAR-10)

FIGURE 5.2 – Trace EM moyenne pour les trois modèles. Spectrogramme pour le CNN (CIFAR-10).

Trouver le nombre de couches L. Lors de l'analyse de la forme générale de la trace EM moyenne, la toute première observation est qu'elle peut être facilement divisée en plusieurs blocs dont les frontières sont généralement bien délimitées. Ces frontières apparaissent encore plus nettement en réalisant une analyse fréquentielle de la trace. La figure 5.2d illustre cette analyse sur le modèle CNN entraîné sur CIFAR-10. Pour nos modèles, cette analyse nous permet de compter avec précisions le nombre exact de couches L. La figure 5.2 illustre les blocs identifiés pour nos modèles avec les mêmes couleurs que dans la figure 5.1. Pour valider que les couches correspondent bien aux blocs identifiés, et uniquement à des fins d'illustration, nous ajoutons des *triggers* dans le code pour marquer clairement les frontières entre les couches.

Identifier la nature des couches. Une fois que l'attaquant a localisé la position des L couches sur la trace EM, il peut en déduire une complexité de chacune d'elle par rapport aux temps d'exécution. Typiquement, les couches de *pooling* et d'activation (surtout avec ReLU) sont bien moins complexes que les couches principales de calcul (i.e., dans notre cas, les couches denses et de convolution). Ainsi, le véritable défi est d'initier correctement le processus d'extraction en identifiant la première couche comme une couche dense ou une couche de convolution. Comme nous suivons un processus d'extraction qui suit l'ordre de calcul de l'inférence, ce "bon" départ est essentiel et influe fortement sur les hypothèses sur $\mathcal{A}_M$. La distinction entre couche dense et couche de convolution repose sur deux analyses :

— **Complexité / temps d'exécution.** Le premier indice est apporté par une analyse temporelle basique. Dans la littérature, la complexité peut être définie selon plusieurs manières, mais une référence est de considérer le nombre de multiplication-accumulation (MACs). La complexité MAC des couches de convolution et des couches denses sont les suivantes :

$$MAC_{conv} = (Z^2 \times C_{in}) \times (H_{out}^2 \times C_{out}) \tag{5.1}$$

$$MAC_{dense} = H_{in} \times H_{out} \tag{5.2}$$

La complexité de la première couche du modèle CNN sur MNIST est $MAC_{conv} = 112896$ et 4.5× moins pour la couche dense : $MAC_{dense} = 25088$. Notons que si la couche dense avait 4.5× plus de neurones, les complexités MAC seraient égales. Néanmoins, cela reviendrait à avoir un nombre de neurones parfaitement illogique pour la couche dense car un trop grand nombre de paramètres entraîne des problèmes critiques d'*overfitting*. Ceci fait aussi partie de l'analyse réalisée par l'attaquant grâce à son expertise en DL.

— **Analyse visuelle des motifs EM :** pour l'analyse des traces, l'attaquant s'appuie principalement sur des activités EM spécifiques. Comme nous le détaillons dans les sections

suivantes, le profil, la répétition et l'imbrication de *patterns* observés entre une couche de convolution et une couche dense sont très différents, ce qui permet de différentier les deux natures de couche.

Une fois la première couche identifiée, la comparaison des autres couches couplée avec l'expertise de l'attaquant permettent de réduire les architectures $\mathcal{A}_M$ probables, si tant est qu'un doute existe quant à la nature de certaines couches.

Extraction des hyperparamètres. Une fois l'architecture globale connue, l'attaquant peut extraire chacun des hyperparamètres des différentes couches. Dans les sections suivantes, nous détaillons la méthodologie d'extraction pour chaque type de couche.

5.1.4 Couche de convolution (H_{out})

5.1.4.1 Extraire H_{out}

Dans CMSIS-NN, les fonctions de convolution reposent sur deux étapes décrites dans l'algorithme. 3 : Im2col et General Matrix Multiply (GeMM). Im2col est une technique standard d'optimisation qui permet de transformer une convolution (équation 2.20, page 19) en un simple produit matriciel. Toutes les zones locales d'un tenseur obtenues par la fenêtre glissante du filtre sont représentées sous forme de vecteurs colonnes (d'où le nom "Im2col") et, tous les filtres sont représentés sous forme de vecteurs lignes. La convolution est alors équivalente au produit matriciel de ces deux représentations, permettant une accélération significative de l'exécution, au prix d'une plus grande empreinte mémoire. Dans CMSIS-NN, ce problème de mémoire est géré en effectuant Im2col. Pour réduire cette empreinte, CMSIS-NN effectue Im2col de façon itérative, chaque itération ne considérant qu'un petit ensemble des vecteurs colonnes [76]. En fonction de la taille des tenseurs d'entrée et de sortie, trois fonctions de convolution différentes sont proposées[7] :

— `arm_convolve_HWC_q7_fast()` (cf. code couleur dans la figure 5.1) est destinée aux tenseurs avec C_{in} multiple de 4 (en raison des opérations SIMD) et C_{out} multiple de 2 (en raison de la multiplication matricielle qui est appliquée sur des éléments de taille 2×2). Le calcul est accéléré pour la gestion du *padding* en divisant le tenseur d'entrée en blocs de taille 3×3.

— `arm_convolve_HWC_q7_RGB()` (cf. code couleur dans la figure 5.1) est exclusivement optimisée pour des entrées avec 3 canaux (typiquement des images RGB).

— `arm_convolve_HWC_q7_basic()` (cf. code couleur dans la figure 5.1) est utilisée dans les autres cas et présente une structure très similaire à la variante RGB ci-dessus.

Algorithm 3 Implémentation générale de la convolution

Input: I_{in} Tenseur d'entrée de taille $H_{in} \times H_{in} \times C_{in}$, Ker (tenseur des filtres), S, P, I_{out} Tenseur de sortie de taille $H_{out} \times H_{out} \times C_{out}$
Output: Sortie I_{out}

1: **for** $i_y \leftarrow 0,\ i_y < H_{out},\ i_y{+}{+}1$ **do** ▷ Itération selon over H_{out} (y-axis)
2: **for** $i_x \leftarrow 0,\ i_x < H_{out},\ i_x{+}{+}1$ **do** ▷ idem (x-axis)
3: $\text{buff}_{in} \leftarrow im2col(I_{in}, i_y, i_x, S, P, H_{in}, C_{in})$ ▷ Application de la conversion im2col
4: **if** $len(\text{buff}_{in}) == 2 \times C_{in} \times Z^2$ **then**
5: $GeMM(\text{buff}_{in}, Ker, C_{out}, C_{in} \times Z^2, I_{out})$ ▷ Multiplication matricielle
6: $\text{buff}_{in} \leftarrow 0$ ▷ *Reset* du buffer

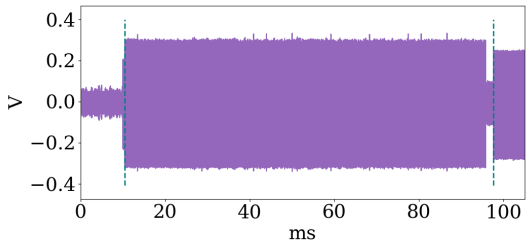
L'extraction de l'hyperparamètre H_{out} repose sur le même principe, quelle que soit l'implémentation de la convolution. Comme présenté dans l'algorithme 3, la boucle principale (ligne 1) itère sur la taille du tenseur H_{out}. Pour chaque itération, nous avons deux grandes étapes de calcul avec im2col et la multiplication matricielle (GeMM) qui consomment la majorité des opérations et sont susceptibles d'induire de fortes activités électromagnétiques, plus particulièrement GeMM (décrite dans l'algorithme 4) qui comporte beaucoup de multiplications.

La fonction GeMM est appelée toutes les deux itérations (ligne 4, instruction `if`) lorsque le *buffer* $\texttt{buff}_{in}$ est chargé avec deux *colonnes d'entrée* grâce à im2col (ligne 3). En conséquence, lors de l'exécution de l'algorithme 3, GeMM (ligne 5) est appelée $H_{out} \times H_{out}/2$ fois.

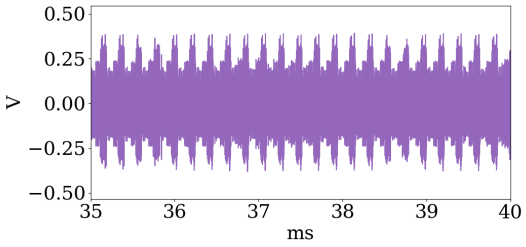
Si l'on note N_p le nombre de motifs réguliers résultant de la fonction GeMM sur la partie de la trace EM correspondant à la couche de convolution ciblée, alors nous pouvons relier N_p à H_{out} de la façon suivante :

$$N_p = \frac{H_{out} \times H_{out}}{2}, \quad \text{soit} \quad H_{out} = \sqrt{2N_p}.$$

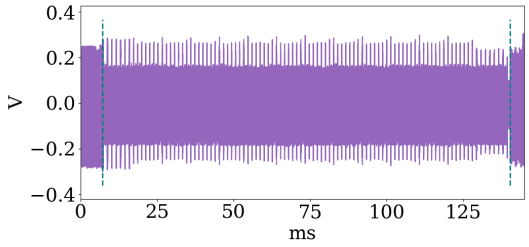
7. Des fonctions équivalentes sont disponibles pour des tenseurs d'entrée non carrés.



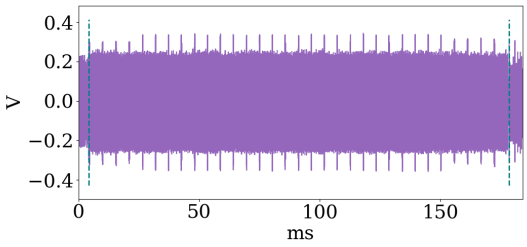
(a) 1^{ere} convolution (`basic`), MNIST CNN, 392 motifs attendus



(b) Zoom sur (a) avec 23 motifs visibles



(c) 2^{nde} convolution (`fast`), MNIST CNN, 98 motifs visibles



(d) 3^{ieme} convolution (`fast`), CIFAR-10 CNN, 32 motifs visibles

FIGURE 5.3 – Traces EM pour des couches de convolution et motifs relatifs à H_{out}.

Après cette analyse, nous évaluons nos traces EM pour chaque couche de convolution de nos deux modèles CNN (MNIST et CIFAR-10). Nous observons bien le nombre correct de motifs N_p, comme indiqué dans le tableau 5.2. La figure 5.3a illustre la première couche de convolution pour le CNN sur MNIST avec $N_p = 392$ motifs (la figure 5.3b présente un zoom sur 23 motifs entre 35 et 40 ms). Les figures 5.3c et 5.3d illustrent des couches de convolution `fast` des CNNs MNIST et CIFAR-10, avec respectivement $N_p = 98$ et 32 motifs.

Selon les couches observées, ce dénombrement "à la main" de N_p pour déduire H_{out} peut être fastidieux. Néanmoins, comme les motifs sont très bien définis (voir les figures 5.3b et 5.3d), un attaquant peut utiliser des outils classiques de détection de motifs (à partir de la sélection d'un motif de référence) pour automatiser le processus. Nous considérons que l'utilisation et l'évaluation de ce type de techniques est en dehors du cadre de ce travail.

Une fois extrait H_{out}, nous pouvons regarder à l'intérieur d'un motif pour trouver d'autres répétitions qui correspondent à la fonction GeMM. L'analyse de ces nouveaux motifs et du code permet d'extraire le nombre de filtres K ainsi que leur taille Z.

5.1.4.2 Extraire K

Le calcul de la multiplication matricielle (algorithme 4) repose sur une boucle `for` principale qui itère sur `rowCnt` (ligne 1) qui est égale à $C_{out}/2$ avec C_{out} le nombre de canaux de sortie qui est aussi équivalent à K pour une couche de convolution. Ainsi, en déterminant le nombre d'itérations de la boucle `for` (lignes 2–9) nous pouvons retrouver la valeur de K. En notant N_p le nombre de motifs EM réguliers résultant des calculs à l'intérieur de cette boucle, nous posons :

$$K = 2 \times N_p.$$

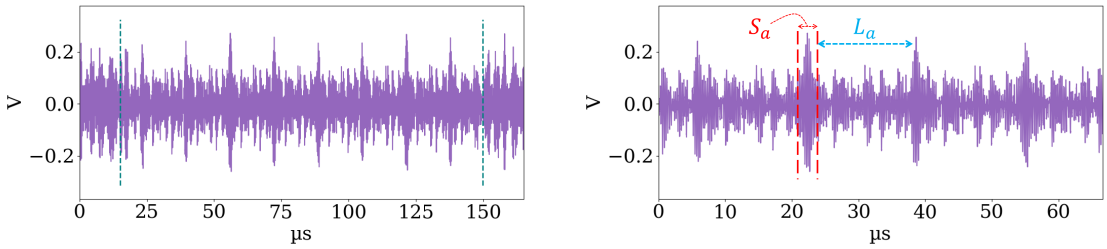
Algorithm 4 Produit matriciel pour la convolution (**GeMM**)

Input: buff_{in}, Ker, C_{out}, $C_{in} \times Z^2$, I_{out}, $bias$ (bias tensor)
Output: (partiellement) I_{out}
1: $rowCnt \leftarrow C_{out} >> 1$ ▷ Initialisé $rowCnt = K/2$
2: **for** $rowCnt > 0$, $rowCnt - -1$ **do** ▷ Itération sur $K/2$
3: $sum, sum1, sum2, sum3 = init\_sum(bias, C_{in} \times Z^2)$
4: $colCnt \leftarrow (C_{in} \times Z^2) >> 2$ ▷ $colCnt$ comme dans Eq. 5.3
5: **for** $colCnt > 0$, $colCnt - -1$ **do**
6: $simd\_mac(sum, sum1, sum2, sum3, \text{buff}_{in}, Ker)$
7: $apply\_mac(sum, sum1, sum2, sum3, C_{out}, I_{out})$
8: $Manage\_remainder\_if\_any(C_{out}, \text{buff}_{in}, bias, C_{in} \times Z^2)$

Table 5.2 – H_{out} et le nombre de motifs attendu

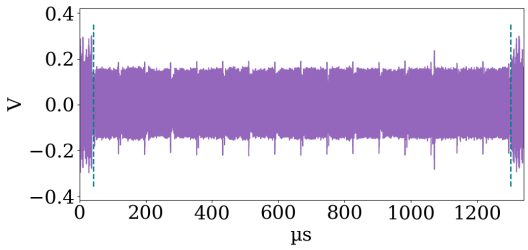
Couche	MNIST		CIFAR-10	
	H_{out}	N_p	H_{out}	N_p
1	28	392	32	512
2	14	98	16	128
3	∅	∅	8	32

Nous évaluons le nombre de motifs réguliers apparaissant dans nos traces en effectuant un zoom sur le premier motif à partir duquel nous avons précédemment extrait H_{out}. Pour chaque couche de convolution, nous observons bien N_p nouveaux motifs réguliers. Ces groupes de N_p motifs sont visibles tout au long de l'activité EM associée à l'ensemble de la couche de convolution, correspondant à chaque appel de la fonction GeMM de l'algorithme 3. Plus précisément, un motif est constitué d'un pic (Sa) suivi d'une période d'activité plus faible (La) (figures 5.4b et 5.4d). Les figures 5.4a et 5.4c représentent les traces EM pour la boucle `for` pour les deux premières couches de convolution du CNN MNIST, avec respectivement $N_p = 8$ et 16, ce qui correspond à $K = 16$ et 32 filtres. Nous arrivons à extraire avec succès les valeurs de K pour toutes les autres couches de convolution de nos deux modèles CNN sur MNIST et CIFAR-10.

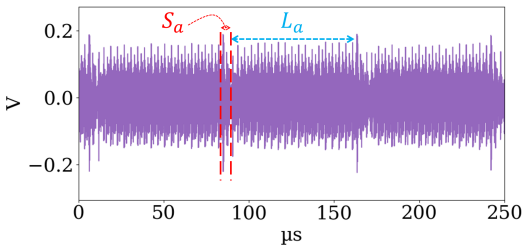


(a) 1^{re} conv., 8 motifs visibles, $K = 16$

(b) Zoom sur la 1^{re} conv.



(c) 2^{nd} conv., 16 motifs visibles, $K = 32$



(d) Zoom sur la 2^{nd} conv.

Figure 5.4 – Traces de l'exécution de GeMM pour le modèle CNN (MNIST)

5.1.4.3 Extraire Z

Comme le nombre de filtres K, leur taille Z est directement utilisée dans l'algorithme 4. La boucle interne `for` (lignes 5–7) itère sur la variable `colCnt`, qui dépend directement de Z. Cette partie représente le calcul principal de la convolution (avec des accumulations basées sur SIMD, ligne 6). L'affectation de `colCnt` (ligne 4) est définie par l'équation 5.3 :

$$colCnt = \tfrac{1}{4}(C_{in} \times Z^2) \tag{5.3}$$

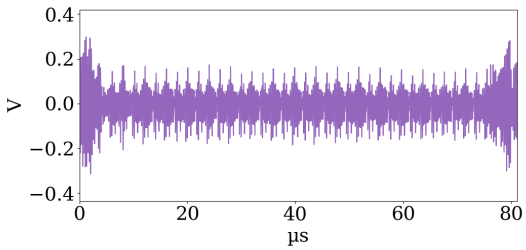
Ainsi, comme précédemment, nous supposons que si ces calculs réguliers génèrent des motifs EM particuliers, alors il est possible d'estimer Z. Néanmoins, cette hypothèse repose sur la connaissance de C_{in}. Nous rappelons que l'attaquant est supposé maîtriser les dimensions du tenseur de sortie de la couche précédente (pour pouvoir passer à la suivante), ce qui signifie qu'il connaît C_{in} (et, initialement, il connaît parfaitement les dimensions des images d'entrée). Donc, si l'activité EM liée à la boucle interne `for` peut être distinguée du reste avec N_p motifs réguliers,

alors Z peut être estimé par :

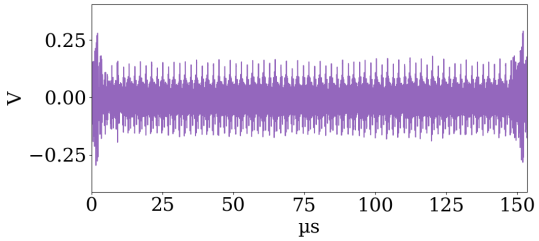
$$Z = \sqrt{\frac{4 \times N_p}{C_{in}}}$$

En observant l'activité EM entre deux pics *La* (figure 5.4), nous identifions bien des motifs qui sont illustrés dans la figure 5.5. Les activités EM liées aux accumulations basées sur SIMD de la deuxième et la troisième couches de convolution du CNN CIFAR-10 sont respectivement présentées dans les figures 5.5a et 5.5b. Pour la deuxième couche, avec $C_{in} = 16$, nous pouvons compter $N_p = 36$ motifs, ce qui donne correctement $Z = 3$. Le même résultat (correct) est obtenu pour la troisième couche avec $C_{in} = 32$ et $N_p = 64$ motifs visibles.

Il est important de noter que la division de $C_{in} \times Z^2$ par 4 peut générer des restes à traiter séparément par la suite. Dans ce cas, une autre activité EM apparaît après celle attendue pour `colCnt`. Une difficulté est que les motifs à observer peuvent devenir très petits et difficiles à distinguer les uns des autres. De plus, la valeur de C_{in} influence fortement le nombre de motifs à dénombrer. Par exemple, le nombre de canaux d'entrée dans la première couche de convolution du CNN MNIST est nécessairement égal à 1, car les entrées sont des images en niveaux de gris. Avec des noyaux de taille 3×3, seulement deux motifs doivent apparaître, avec un reste de division traité séparément, générant une activité EM différente. Ainsi, l'estimation de K peut être difficile surtout quand C_{in} est faible.



(a) 2^{nd} couche de convolution, CNN CIFAR-10, *36 motifs visibles avec* $C_{in} = 16$



(b) 3^{ime} couche de convolution, CNN CIFAR-10, *72 motifs visibles avec* $C_{in} = 32$

FIGURE 5.5 – Zoom sur l'opération de multiplication matricielle, avec $Z = 3$

Une fois que H_{in}, $H_{out} = K$ et Z sont connus, comme nous avons logiquement $P < Z$ (le *padding* est inférieur à la taille du filtre) et grâce à l'équation 2.19 (page 19), il est possible de déduire le *stride* S et le *padding* P. En calculant S pour les différentes valeurs possibles de P, une seule valeur entière sera obtenue pour S.

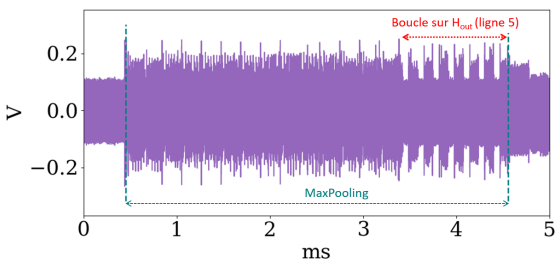
5.1.5 Couche de *pooling*

L'implémentation CMSIS-NN du *MaxPooling* (algorithme 5) repose d'abord sur l'opération de *pooling* selon l'axe x (lignes 1 à 4) avec deux boucles imbriquées sur H_{in} et H_{out} et une seconde application du *pooling* (lignes 5 à 7) selon l'axe y avec une seule boucle sur H_{out}. Ce dernier bloc de calcul est le plus intéressant car il ne traite que d'un seul hyperparamètre (H_{out}). L'opération de *pooling* en elle-même sélectionne d'abord les zones concernées (ligne 3 et 6) puis le calcul de la statistique utilisée (dans notre cas le maximum, ligne 4 et 7). L'opération de pooling est classiquement réalisée en deux étapes : (1) la sélection et l'allocation des zones locales, puis (2) le calcul statistique (ici, le maximum). Ainsi, nous nous attendons à observer deux activités EM distinctes liées à ces boucles séparées, la dernière (et plus courte) étant directement liée à H_{out}.

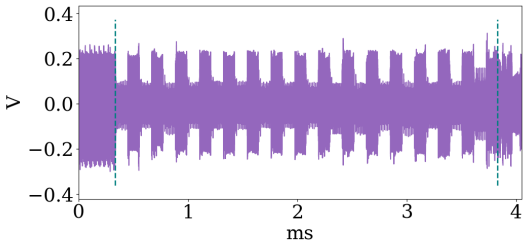
Algorithm 5 MaxPool - `arm_maxpool_q7_HWC`

Input: I_{in} input tenor of size $H_{in}^2 \cdot C_{in}$, I_{out} output tenor of size $H_{out}^2 \cdot C_{out}$, P, S, H_{ker}
Output: Filled I_{out}

1: **for** $i_y \leftarrow 0$, $i_y < H_{in}$, i_y ++1 **do** ▷ Pooling along x axis
2: **for** $i_x \leftarrow 0$, $i_x < H_{out}$, i_x ++1 **do**
3: $win_{start}, win_{stop} \leftarrow set\_window(i_y, i_x, I_{in}, H_{ker}, P, S)$
4: $compare\_and\_remplace\_if\_larger(win_{start}, win_{stop}, i_y, i_x, I_{in})$
5: **for** $i_y \leftarrow 0$, $i_y < H_{out}$, i_y ++1 **do** ▷ Directly iterates over H_{out}
6: $row_{start}, row_{stop} \leftarrow set\_rows(i_y, I_{in}, I_{out}, H_{ker}, P, S)$
7: $compare\_replace\_then\_apply(row_{start}, row_{stop}, I_{in}, I_{out})$



(a) Trace de la 2^{nd} couche MaxPool du modèle CNN MNIST, $H_{out} = 7$



(b) Zoom sur la dernière boucle de la première couche MaxPool du modèle CNN CIFAR-10, $H_{out} = 16$

FIGURE 5.6 – Couches MaxPool des modèles CNN. Les traits bleus sont des *triggers* ajoutés spécifiquement pour ces traces.

Pour le modèle CNN sur MNIST, l'activité EM de la seconde couche de MaxPooling est représentée dans la figure 5.6a. On observe une séparation claire entre, d'une part, les deux premières boucles imbriquées (lignes 1 à 4) et d'autre part la dernière boucle sur H_{out} (lignes 5 à 7). La seconde partie fait apparaître des motifs plus facilement identifiables. Un zoom sur cette seconde partie est donné dans la figure 5.6b pour la première couche du modèle CNN sur CIFAR-10. Les traits en bleu représentent des triggers placés spécialement pour s'assurer que les émanations correspondent bien à la boucle `for` itérant sur H_{out}. En nommant N_p le nombre de motifs dans cette zone, on peut trouver directement H_{out}. Ceci est vérifié pour toutes les couches de *MaxPooling* de nos modèles, sans difficulté. Notons qu'il est aussi possible de trouver H_{out} en analysant la première zone d'émanations mais le dénombrement est plus complexe car les motifs sont moins identifiables.

Une fois identifié H_{out}, comme l'attaquant connaît H_{in}, il peut trouver Z_{pool} puisque $Z_{pool} = H_{out}/H_{in}$.

5.1.6 Couche dense

L'algorithme 6 décrit les opérations effectuées pour une couche dense. Elle est construite autour d'une boucle `for` sur la variable `rowCnt`, correspondant à $N_e/4$, N_e étant le nombre de neurones. Dans CMSIS-NN, une couche dense traite les neurones par groupe de 4. La fonction `init_sum_with_bias` (ligne 3) initialise quatre variables de somme, utilisées pour le calcul des sommes pondérées et l'ajout des valeurs de biais grâce à l'opération de multiplication-accumulation SIMD (`__SMLAD`) dans la fonction `simd_mac` (ligne 6).

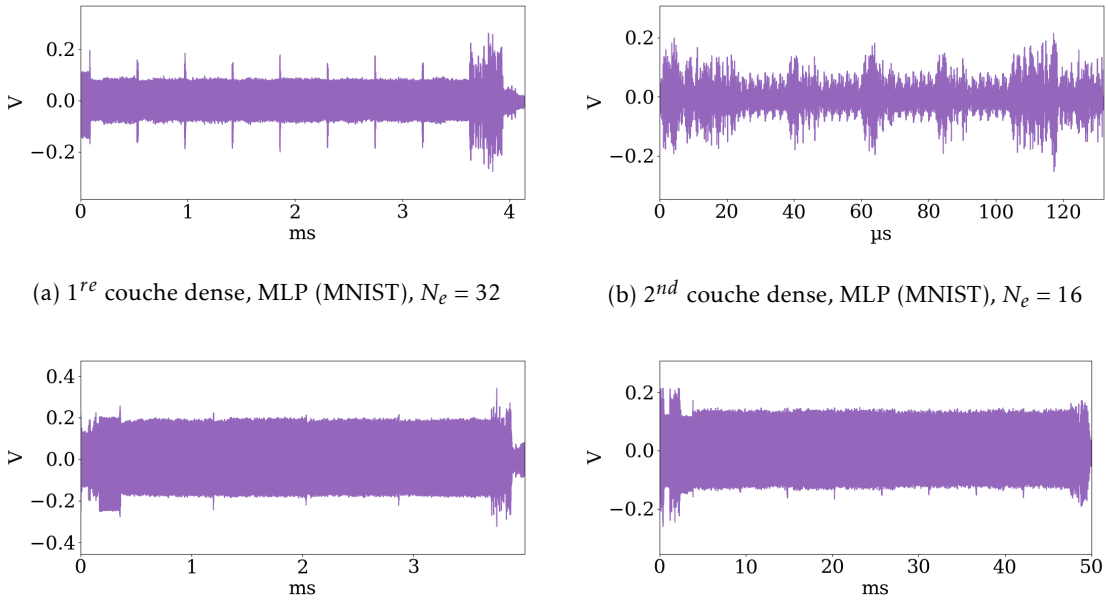
En posant de nouveau N_p le nombre de motifs que l'on peut dénombrer dans les traces EM, nous devons avoir $N_e = 4 \times N_p$.

Algorithm 6 Dense layer - `arm_fully_connected_q7_opt`

Input: I_{in} input vector of size H_{in}, ker weight vector of size N_e, $bias$ bias matrix of size N_e, I_{out} output vector of size H_{out}, P, S, H_{ker}
Output: Filled I_{out}

1: $rowCnt \leftarrow N_e >> 2$ ▷ Nb. neurons divided by 4
2: **for** $rowCnt > 0$, $rowCnt--1$ **do** ▷ Iterate directly over $N_e/4$
3: $sum, sum1, sum2, sum3 = init\_sum\_with\_bias(bias, rowCnt)$
4: $colCnt \leftarrow H_{in} >> 2$
5: **for** $colCnt > 0$, $colCnt--1$ **do**
6: $simd\_mac(sum, sum1, sum2, sum3, ker, colCnt)$
7: $apply\_mac(sum, sum1, sum2, sum3, rowCnt, I_{out})$
8: $rowCnt \leftarrow N_e$ & 0x3 ▷ Manage remainders if any
9: **for** $rowCnt > 0$, $rowCnt--1$ **do**
10: $sum = init\_sum\_with\_bias(bias, rowCnt)$
11: $colCnt \leftarrow H_{in} >> 2$
12: **for** $colCnt > 0$, $colCnt--1$ **do**
13: $mac(sum, ker, colCnt)$
14: $apply\_mac(sum, rowCnt, I_{out})$

L'analyse des traces montre effectivement une activité EM significative, comme dans la figure 5.7. Les motifs répétés sont très différents de ceux observés pour les couches de convolution (ou de *pooling*), avec des parties uniformes séparées par des pics très distincts que l'on observe bien sur la figure 5.7a. Logiquement, la longueur des motifs EM est directement liée au nombre d'entrées de la couche, et donc au nombre de neurones. Ainsi, des couches denses qui traitent des entrées de plus grande dimension seront plus simples à analyser.



(a) 1^{re} couche dense, MLP (MNIST), $N_e = 32$

(b) 2^{nd} couche dense, MLP (MNIST), $N_e = 16$

(c) Couche dense, CNN (MNIST), $N_e = 16$

(d) Couche dense, CNN (CIFAR-10), $N_e = 32$

FIGURE 5.7 – Traces EM de couches denses.

A partir des traces EM, nous dénombrons les N_p motifs qui sont bien séparés par les pics d'activité, pour chaque couche dense de nos modèles MLP et CNN et nous vérifions bien la relation entre N_p et le nombre de neurones N_e de la couche. Les figures 5.7a et 5.7b illustrent les deux couches denses du modèle MLP MNIST avec respectivement 8 et 4 motifs qui correspondent parfaitement à $N_e = 32$ et $N_e = 16$ neurones. De même, les figures 5.7c (CNN MNIST) et 5.7d (CNN CIFAR-10) montrent des couches denses avec respectivement $N_e = 16$ (4 motifs) et $N_e = 32$ (8 motifs) neurones.

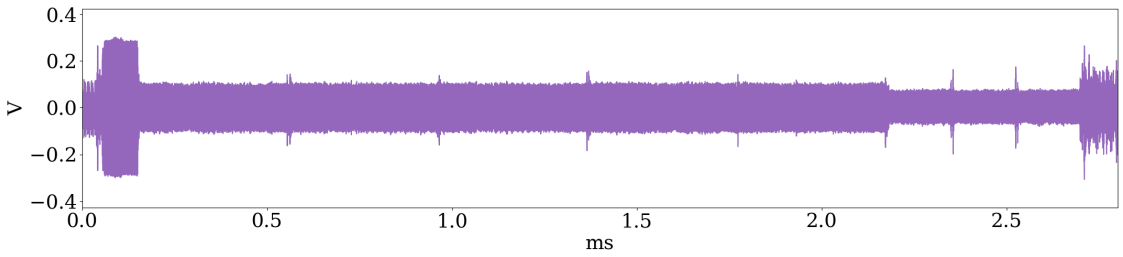
Néanmoins, cette approche à une limitation qui tient aux cas particuliers des couches avec un nombre de neurones non multiple de 4. Dans ces cas, des neurones restants doivent encore être traités après la boucle sur les paquets de 4 neurones. La ligne 10 de l'algorithme 6 vérifie s'il y a des restes et, le cas échéant, les neurones restant sont traités avec une boucle `for` supplémentaire. Les neurones sont alors traités un par un, avec une seule variable d'accumulateur initialisée puis utilisée dans l'appel de la fonction de multiplication-accumulation à la ligne 15.

Pour illustrer ce phénomène, nous entraînons un autre modèle MLP sur MNIST, composé de 4 couches denses avec respectivement 23, 18, 13 et 10 neurones (i.e., reste respectif : 3, 2, 1, 2). La figure 5.8 représente l'activité EM de chaque couche pour ce nouveau modèle. Pour la première couche ($N_e = 23$), nous observons clairement (figure 5.8a) trois blocs distincts après la séquence principale de 5 motifs, c'est-à-dire 5 groupes de 4 neurones complétés par les 3 neurones restants, traités indépendamment. Or, nous ne pouvons faire une analyse similaire sur les deux autres traces. Cette difficulté provient de la réduction de la forme du tenseur d'entrée entre la première couche, la deuxième et la troisième couche.

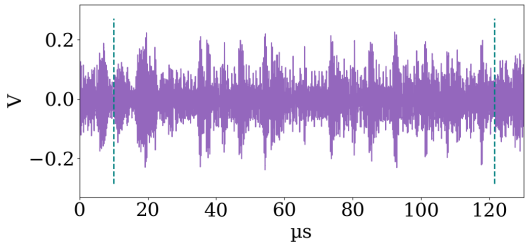
Pour vérifier que les neurones sont gérés de la même manière dans ces deux cas, des *triggers* sont activés et désactivés à l'intérieur de la boucle extérieure `for` sur `rowCnt` (représentés par des rectangles dans les figures 5.8d et 5.8e). Notons que cela illustre aussi les modifications de la forme des motifs EM qui sont induites par l'utilisation de *triggers* autour du code observé. En effet, ceux-ci deviennent plus visibles dans les figures 5.8d et 5.8e par rapport aux figures 5.8b et 5.8c.

5.1.7 Activation

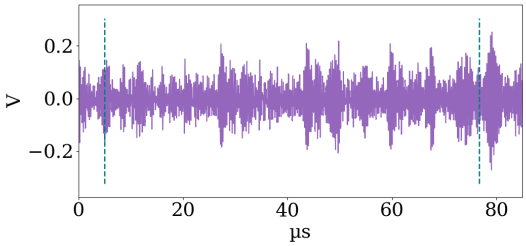
La connaissance de la fonction d'activation est indispensable pour l'estimation des tenseurs d'entrée d'une couche, notamment dans un scénario d'extraction des paramètres (Cf. section suivante). Par exemple, la fonction ReLU va nécessairement transformer la sortie d'une couche dans $\mathbb{R}^+$. Rappelons que les fonctions d'activations les plus populaires sont Sigmoid, Tanh, Softmax et ReLU (Cf. chapitre 2, page 18). ReLU est très largement utilisée dans les modèles MLPs et CNNs pour son efficacité à l'entraînement et sa simplicité d'implémentation par rapport à Sigmoid,



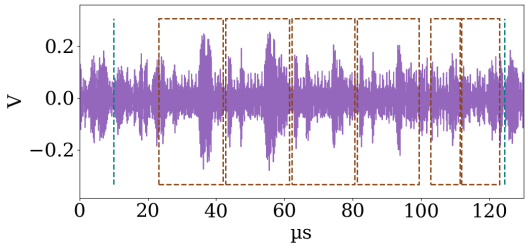
(a) 1^{ere} dense, $N = 23 \Leftrightarrow 5 + 3$ *motifs*



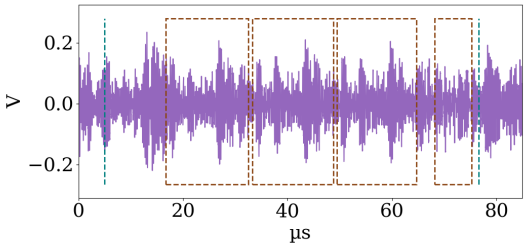
(b) 2^{nde} dense $N = 18 \Leftrightarrow 4 + 2$ *motifs*



(c) 3^{ieme} dense $N = 13 \Leftrightarrow 3 + 1$ *motifs*



(d) Comme (b) mais avec des *triggers*



(e) Comme (c) mais avec des *triggers*

FIGURE 5.8 – Traces EM des couches denses pour le modèle MLP adapté (nombre de neurones). Les motifs attendus apparaissent clairement pour la première couche mais pas pour les deux autres.

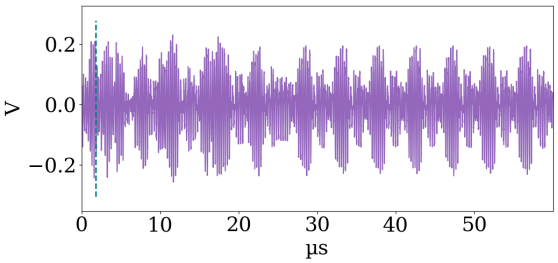
Tanh et Softmax qui impliquent des opérations de division ou exponentielle, coûteuses en temps de calcul. Aussi, nous considérons que la connaissance que souhaite acquérir l'attaquant est "la fonction d'activation est-elle ReLU ?".

L'état de l'art a déjà montré que l'étude du temps d'exécution permettait de déterminer la nature de l'activation – comme dans [105] sur microcontrôleur – car les différentes fonctions ont des profils temporels très différents, surtout pour ReLU qui est bien plus rapide que les autres options. Le tableau (*a*) de la figure 5.9 montre le temps d'exécution des trois couches du modèle CNN sur MNIST en utilisant des activations différentes. Cependant, en plus des temps d'exécution, l'analyse des motifs EM apporte des informations supplémentaires permettant de différencier ReLU des autres activations. Les traces de la figure 5.9 sont des zooms sur le début de la deuxième couche d'activation du modèle CNN (MNIST) pour lequel nous avons utilisé trois activations différentes. La trace 5.9b, correspondant à ReLU, présente des motifs réguliers et facilement distinguables (d'une durée de quelques μs) qui diffèrent de ceux de Sigmoid et Tanh. Notons, que si la séparation entre ReLU et les deux autres activations ne pose pas de difficulté, faire la différence entre Sigmoid et Tanh apparaît plus complexe à réaliser.

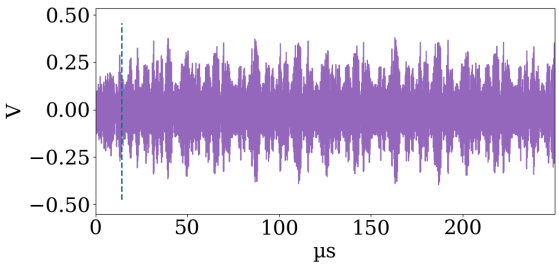
Cette étude propose une méthodologie d'extraction de l'architecture pour des modèles MLP et CNN standards déployés sur un microcontrôleur caractéristique de l'IA embarqué (Cortex-M7). La méthode repose sur la simple observation d'une trace EM moyennée et des connaissances liées, d'une part, à la librairie de déploiement CMSIS-NN et, d'autre part, aux pratiques usuelles en DL. La très grande régularité des calculs effectués lors de l'inférence se traduit par la présence de nombreux motifs dans les traces que l'on peut dénombrer et relier aux hyperparamètres des différentes couches. La méthode s'avère très efficace même s'il existe quelques limitations que

Couche	ReLU	Sigmoid	Tanh
1	1.84ms	8.24ms (×4.5)	8.24ms (×4.5)
2	0.92ms	3.21ms (×3.5)	3.58ms (×3.9)
3	7.60μs	15.60μs (×2)	20.64μs (×2.7)

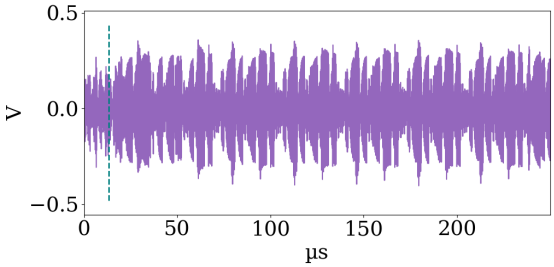
(a) Temps de traitement par couche en fonction de la fonction d'activation.



(b) Avec ReLU



(c) Avec Sigmoid



(d) Avec Tanh

FIGURE 5.9 – CNN (MNIST) : zoom sur le début de la seconde couche en fonction de la fonction d'activation et temps de traitement associé.

nous discutons en fin de chapitre. Néanmoins, dans un scénario d'extraction par "fidélité", l'attaquant qui dispose de l'architecture du modèle cible va vouloir aller plus loin, et extraire les valeurs des paramètres entraînables du modèle. Dans la section suivante, nous proposons une étude approfondie des capacités d'une analyse *side-channel* pour ce type d'extraction.

5.2 Utilisation d'analyses side-channel pour l'extraction des paramètres

Puisque nous traitons de modèles DNN déployés sur des cibles de type microcontrôleurs 32-bit, la principale référence sur l'extraction de modèle par analyse *side-channel* sur ce type de cible est [105] (CSI-NN) de Batina *et al.* . Bien que CSI-NN couvre aussi l'extraction de l'architecture, le cœur de leur approche concerne les valeurs des paramètres qui sont stockés en valeurs flottantes 32-bit sous la norme IEEE-754 (Cf. chapitre 2, page 15). Les deux plateformes mentionnées dans [105] sont un ATmega328P (ouvert) et un SAM3X8E ARM Cortex-M3 pour lesquels des opérations en virgule flottante sont effectuées sans unité de traitement spécifique (FPU). Néanmoins, pour l'extraction des paramètres, la principale plateforme utilisée est l'ATmega328P (8-bit).

Leur étude ne concerne pas les biais (qui ne sont pas mentionnés) et l'objectif est de retrouver la valeur des paramètres mais avec un objectif de faible précision (10^{-2}). La technique d'analyse *side-channel* est la CPA et va donc chercher, pour chaque paramètre, à corréler les traces EM capturées avec plusieurs hypothèses de la valeur du paramètre et un modèle de fuite en poids de Hamming (HW). L'opération ciblée est la multiplication entre le paramètre visé et une entrée, plus particulièrement le poids de Hamming (HW) du résultat de la multiplication (instructions STD vers les registres 8 bits). Les hypothèses de la valeur d'un paramètre sont construites en considérant une plage réaliste de valeurs $[-N,+N]$ et la précision souhaitée $p = 10^{-2}$. Il y a donc $1 + 2N/p$ valeurs possibles pour un paramètre. L'extraction par CPA concerne d'abord un octet contenant le signe et les 7 premiers bits de l'exposant, puis le reste de la mantisse (trois registres de 8 bits). Notons qu'aucune mention n'est faite sur l'adaptation de leur technique pour le passage sur le microcontrôleur 32-bit (Cortex-M3).

Leur méthodologie est évaluée avec un modèle MLP entraîné sur MNIST et un CNN quantifié sur 8 bits entraîné sur CIFAR-10. L'implémentation, compilation et déploiement des inférences sur les cibles ne sont pas publiques. Une limitation importante de l'étude est que les caractéristiques du CNN par rapport au MLP (paramètres partagés, quantification) qui doivent avoir une influence forte sur les profils et l'exploitation des fuites ne sont pas discutées. Les auteurs concluent sur l'extraction des deux modèles permettant de construire des modèles substituts

possédant une différence d'*accuracy* moyenne sur l'ensemble de test de 0,01% et 0,36%, avec une erreur moyenne des paramètres de 0,0025 pour le MLP.

Puisque la désynchronisation est importante (la multiplication logicielle et la fonction d'activation ne sont pas en temps constant), les traces EM sont synchronisées à chaque fois en fonction du neurone ciblé. Néanmoins, il est important de mentionner que l'objectif de CSI-NN est d'extraire trois informations : (1) la nature des activations par analyse temporelle, (2) la valeur des paramètres par CPA et (3) la séparation entre les couches (information induite par le succès ou non de la CPA sur les paramètres). La façon dont les différentes analyses s'articulent entre-elles (notamment pour les problématiques de désynchronisation) n'est pas détaillée dans [105]. Néanmoins, dans cette section, nous nous comparons uniquement sur la méthodologie d'extraction des valeurs des paramètres.

L'étude proposée dans [105] soulève de nombreuses questions ouvertes et il ne nous pas été possible de reproduire la majorité des résultats sur l'extraction des valeurs des paramètres. Par conséquent, à travers l'élaboration d'une nouvelle méthodologie d'extraction (toujours dans un scénario basé sur la *fidélité*), nous proposons d'évaluer les capacités d'un attaquant qui utiliserait une analyse CPA standard pour extraire les paramètres d'un modèle MLP déployé pour l'inférence dans le microcontrôleur 32 bits Cortex-M7, déjà utilisé dans les expériences précédentes pour l'extraction de l'architecture. Plus précisément, par rapport aux commentaires précédents sur CSI-NN :

— Nous cherchons à extraire le plus précisément possible les valeurs des paramètres par analyse *side-channel*.
— Nous mettons en lumière plusieurs problèmes qui ne sont pas mentionnés dans la littérature et qui sont liés à la complexité d'un modèle DNN.
— Nous proposons de partir de l'opération de base (i.e., la multiplication entre deux `float32`) puis d'aller jusqu'à un modèle fonctionnel complet.
— Nous ne prétendons pas être capables de récupérer entièrement tous les paramètres d'un modèle MLP de l'état de l'art, notamment car l'extraction des biais n'est pas encore résolue par CPA.
— Nous considérons une cible de l'état de l'art pour du DL embarquée (FPU, capacités mémoires adaptées). Bien que nous ayons adaptées la cible pour les besoins de l'acquisition EM nous n'avons pas eu besoin d'ouvrir le composant.
— Les traces et implémentations sont publiquement disponibles [8].

5.2.1 Plateforme et *setup* expérimental

Notre plateforme matérielle est identique à celle de nos précédentes expériences sur l'extraction d'architecture. La carte STM32H7 est basée sur un ARM Cortex-M7 avec 2 Mo de mémoire flash et 1 Mo de RAM. Nous avons ajouté un oscillateur externe (quartz de 25 MHz) pour avoir un signal d'horloge plus stable [9]. Le CPU fonctionne ainsi également à 25 MHz. Les émanations sont mesurées avec la même configuration expérimentale que dans la section 5.1.2. Pour réduire le bruit et faciliter l'exploitation des fuites, toutes les traces sont moyennées sur 50 exécutions du programme.

Programme d'inférence. Pour nos études, nous devons maîtriser le programme d'inférence afin de comprendre correctement les propriétés des fuites *side-channel* et leur exploitation potentielle. Aussi, nous implémentons nos propres programmes en C qui sont compilés avec le niveau d'optimisation `O0` de gcc, afin de garantir que chaque multiplication soit suivie de l'instruction `STR` enregistrant le résultat dans la SRAM.

Comme dans [113], certaines approches d'extraction se basent sur une analyse temporelle. Dans notre cas, nous faisons l'hypothèse que le programme cible est protégé contre ce type de menaces avec une implémentation en temps constant des principales opérations, plus particulièrement la fonction d'activation ReLU, dont le code est proposé dans le listing 5.1. Pour évaluer cette implémentation, nous avons mesuré 1000 inférences et avons obtenu un écart-type inférieur au cycle d'horloge.

Enfin, nous utilisons l'unité de calcul en virgule flottante (FPU) du STM32H7 qui permet d'effectuer les calculs en virgule flottante en un seul cycle au lieu de passer par la bibliothèque du

8. `https://gitlab.emse.fr/securityml/SCANN-ex.git`
9. Cf. documentation STMicroelectronics : *STM32H7 System Reset and clock control RCC*

```
float layer_neuron_res; // input
int sign,mask,pre_v,post_v;
void *ppre_v,*ppost_v;

sign=(layer_neuron_res>0.0);
mask=0-sign;
ppre_v=(void*)(&(layer_neuron_res));
pre_v=*((int*)ppre_v);
post_v=pre_v & mask;
ppost_v=(void*)(&(post_v));
layer_neuron_res=*((float *)ppost_v); // output
```

Listing 5.1 – ReLU

compilateur C. Lorsqu'il est disponible, l'utilisation du FPU est préférable car il accélère l'exécution du programme et réduit la charge du CPU.

5.2.2 Difficultés de l'extraction des paramètres

L'extraction des paramètres par une analyse *side-channel* de type CPA soulève plusieurs problèmes liés à la nature même des DNNs.

La première difficulté concerne la gestion des entrées utilisées par l'attaquant pour faire varier la valeur intermédiaire ciblé par la CPA. Ce point est fondamental pour que l'analyse statistique de la CPA fasse ressortir la meilleure hypothèse par rapport aux observations (traces). Or, dans le cas d'un DNN qui utilise ReLU ($max(x, 0)$) comme activation, tous les tenseurs d'entrée – dès la première couche – qui vont être utilisés par toutes les couches intermédiaires auront des valeurs positives ou nulles. Si l'attaquant vise une opération particulière (en l'occurrence la multiplication) après la première couche, il ne pourra pas utiliser d'entrée lui permettant d'avoir des valeurs positives et négatives.

Le second point est lié à la nature d'un MLP qui est un modèle entièrement connecté. La sortie activée d'un neurone est connectée à *tous* les neurones de la couche suivante. Une entrée est donc impliquée dans autant de multiplications que de neurones dans la couche considérée. Dans le cas d'une CPA, cela implique que la connaissance de l'ordre des neurones est obligatoire pour associer correctement les résultats de la CPA aux neurones. Si l'on suppose que l'attaquant possède un clone de la cible lui permettant de *profiler* des couches élémentaires, il est alors possible de s'assurer de cet ordre.

Une autre difficulté importante est directement reliée au point précédent et concerne l'impossibilité de poursuivre l'attaque dès lors qu'une seule erreur d'extraction a été faite. En effet, l'attaquant réalise son extraction, couche par couche et paramètre par paramètre. Une fois que l'extraction d'une couche a été réalisé, les paramètres extraits lui servent de calculer les entrées de la prochaine couche. Comme les neurones sont entièrement connectés les uns aux autres, une seule erreur dans l'estimation de la valeur d'un paramètre rendra l'estimation de l'entrée de la prochaine couche erronée et donc la CPA ne sera plus pertinente pour tous les futurs paramètres à extraire.

L'opération de multiplication entre le paramètre cible et une entrée est effectivement l'opération privilégiée par une CPA [105]. Néanmoins, le calcul d'un neurone est une somme pondérée (opération multiplication-accumulation) qui fait souvent intervenir l'ajout d'un *biais*. Le biais n'est pas directement combiné à une entrée et son addition peut intervenir à différents moments du calcul en fonction du choix d'implémentation de la librairie de déploiement. Par exemple, le *buffer* qui va accumuler les résultats de multiplication entre un paramètre et une entrée peut très simplement être initiée à la valeur du biais. Cela rend l'extraction du biais beaucoup plus complexe que les autres paramètres. Le cas des biais est traité dans d'autres approches [43, 113] mais, pour les analyses *side-channel*, ils ne sont pas mentionnés [105]. Ici, nous les excluons de notre champ d'étude mais apportons une première analyse en fin de section pour souligner les difficultés de l'exploitation des fuites de l'opération d'addition.

Enfin, une dernière difficulté est liée à la réalisation pratique de la CPA pour extraire tous les paramètres du modèle et la capacité de l'attaquant à se positionner précisément dans la trace, pour réaliser ses analyses successives. Pour la plateforme utilisée dans [105] (MCU 8-bit), des motifs EM visibles permettait de déterminer visuellement les opérations dans la trace. Dans notre cas, cela s'avère impossible. Pour ces premières analyses, on se situe dans un scénario où l'attaquant a déjà réalisé un *profiling* temporel sur des clones de la cible et donc en mesure de découper la trace pour isoler les portions correspondant aux différents paramètres qu'il cherche à extraire.

Méthodologie. Notre méthodologie commence par l'analyse de l'opération la plus basique d'un modèle : la multiplication entre un paramètre et une entrée. Ensuite, nous considérons un neurone possédant plusieurs paramètres (i.e., une somme pondérée, sans biais). Après cette étape, nous pouvons analyser une couche entière puis un modèle MLP. Traiter avec un modèle entier signifie récupérer les paramètres couche par couche, suivant le flux du réseau : les extractions d'une couche l sont utilisées pour déduire les entrées de la couche $l+1$. Nous évaluons notre technique d'extraction sur des traces simulées et réelles avec notre cible STM32H7.

Contrairement à l'objectif de Batina *et al.* qui cherchaient à extraire les paramètres avec une fidélité faible (10^{-2}), pour réduire le nombre d'hypothèses, nous souhaitons savoir quels niveaux de précision peut être atteints par une CPA. Notre autre objectif est de proposer une analyse d'extraction suffisamment efficace (i.e., rapide) et éviter une CPA *brute-force* avec 2^{32} hypothèses pour chaque paramètre. Pour cela, nous utilisons une stratégie proche de [105] qui réduit la complexité par bloc de bits.

5.2.3 Opération de multiplication

Nous visons l'opération de multiplication faites par le FPU (`vmul.f32`) $c = w \times x$ entre deux opérandes au format IEEE-754 (32 bits, cf. équation 2.1) : le paramètre secret w du modèle et une entrée connue x. Notre modèle de fuite est le poids de Hamming de c, noté $HW(c)$. Pour réaliser une CPA sur cette opération, nous pouvons calculer $HW(c)$ pour toutes les 2^{32} valeurs possibles de c. Néanmoins, le nombre d'hypothèses devient beaucoup trop importants et le processus est irréaliste à l'échelle d'un modèle MLP classique, aussi notre méthode est basée sur un découpage par bloc des parties de l'exposant et de la mantisse.

L'idée principale est de réaliser des CPA de façon itérative pour extraire une valeur de plus en plus précise (méthode *coarse-to-fine*). Dans un premier temps, nous ne considérons que l'extraction de $|w|$, c'est-à-dire que l'extraction du bit de signe de w est réalisée ultérieurement. Ce principe de CPA itératives ne débute qu'après une première CPA qui permet d'extraire une première valeur de $|w|$ en utilisant uniquement 16 bits : les 8 de l'exposant et les 8 premiers de la mantisse. Cette CPA repose donc sur 2^{16} hypothèses. Pour réduire le nombre d'hypothèses, nous ne considérons que des valeurs dans un intervalle de taille d, centré sur une valeur C ($[C-d/2, C+d/2]$), fixés par l'attaquant. Ceci est réaliste car les paramètres d'un DNN sont des petites valeurs généralement < 1 (conditions aléatoires, *learning rate*, gradients). Une CPA classique permet de conserver les N hypothèses qui possèdent la plus grande corrélation.

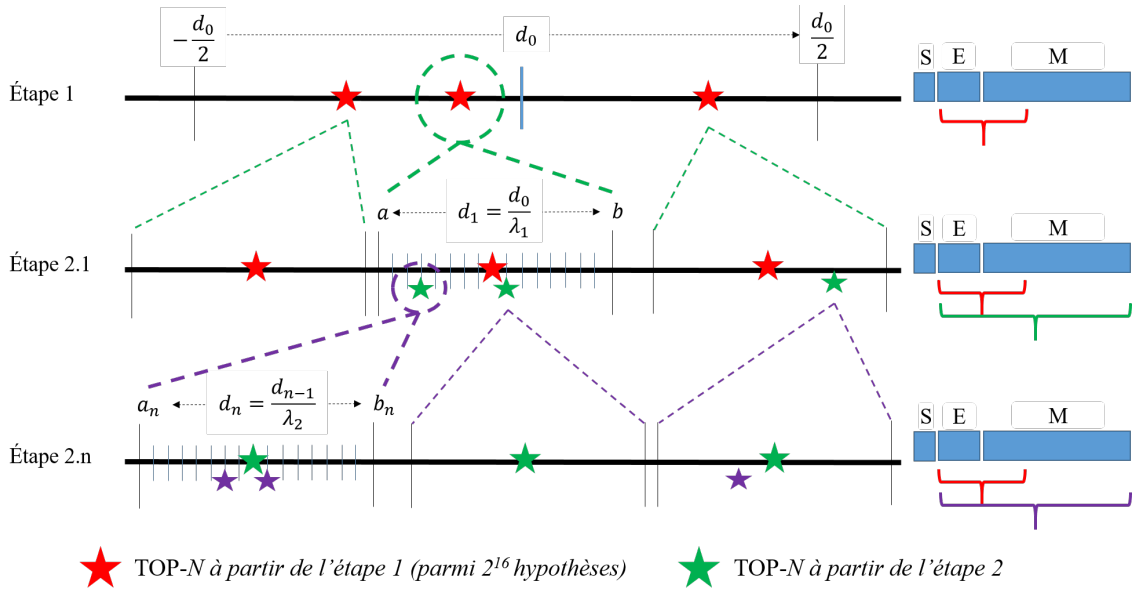


FIGURE 5.10 – Schéma d'extraction itératif. S est le bit de signe. E est la partie exposant et M la mantisse.

L'obtention de ces N valeurs permet d'initier le processus itératif avec une mise en cascade de CPA permettant d'affiner la précision des meilleures hypothèses. On pose m le nombre d'itérations. Pour commencer le processus, pour chacune des N valeurs conservées après la première CPA, notées $\hat{w}_i$, on construit un ensemble d'hypothèses mais en utilisant cette fois-ci tous les bits d'information (sauf le signe). Comme précédemment, on se restreint à un intervalle autour de $\hat{w}_i$

et dont la taille va dépendre de l'itération parmi m (l'intervalle est de plus en plus petit à mesure que l'on cherche une précision plus importante). Toujours pour se restreindre à un nombre raisonnable d'hypothèse, on considère N hypothèses en les échantillonnant uniformément dans cet intervalle. Puisque nous avons N valeurs de l'étape précédente et pour chacune d'entre elles N hypothèses, nous calculons donc $N \times N$ scores de corrélations. De la même manière, nous ne conservons que les N meilleures hypothèses et le processus peut recommencer en utilisant une taille d'intervalle plus petite. Le schéma de la figure 5.10 illustre la méthodologie d'extraction. La variation de la taille d de l'intervalle est faite par un facteur λ ($d \leftarrow d/\lambda$). Néanmoins, nous avons observé une meilleure convergence du processus en utilisant deux tailles d'intervalle d et deux facteurs λ différents entre la première étape et les étapes itératives (d_0, d_1, λ_1, λ_2 dans la figure 5.10). Typiquement, pour nos expériences, nous posons $N = 5$, $d_0 = 5$, $C = 2.5$, $m = 3$, $\lambda_1 = 100$ et $\lambda_2 = 50$.

Simulations. Nous générons aléatoirement 5 000 valeurs de paramètres w. Pour chaque paramètre, 1000 traces simulées sont composées de 3 données, la première et dernière étant une variable aléatoire et la seconde le poids de Hamming $HW(w \times x)$ avec x une donnée aléatoire. On ajoute un bruit gaussien ($\mu = 0$, σ) à l'ensemble de la trace qui nous permet de mesurer l'efficacité de la méthode en fonction de niveaux de bruit $\sigma = 1$ et 10. Les résultats sont présentés dans le tableau 5.3. Même avec le plus grand niveau de bruit, l'extraction a un taux de succès proche de 90% quand on considère une erreur inférieure à 10^{-6}.

Table 5.3 – Taux de succès (SR) de l'extraction d'un paramètre à partir d'une multiplication, en fonction du bruit σ pour différent objectif de précision (ϵ_{rr}).

	$\epsilon_{rr} \leq$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
SR	$\sigma^2 = 1$	1	0.999	0.988	0.986	0.969	0.948	0.812	0.754
	$\sigma^2 = 10^2$	0.999	0.982	0.943	0.930	0.898	0.865	0.696	0.624

Expériences sur Cortex-M7. Pour les expériences sur le STM32H7, nous utilisons deux valeurs secrètes positives qui sont codées en dur dans le programme. Les entrées x sont générées par un script Python et envoyées par l'intermédiaire de l'interface UART. Nous obtenons 3,000 traces (moyenne sur 50 traces). Le programme effectue deux multiplications avec les valeurs secrètes et des entrées distinctes. La figure 5.11) représente une trace.

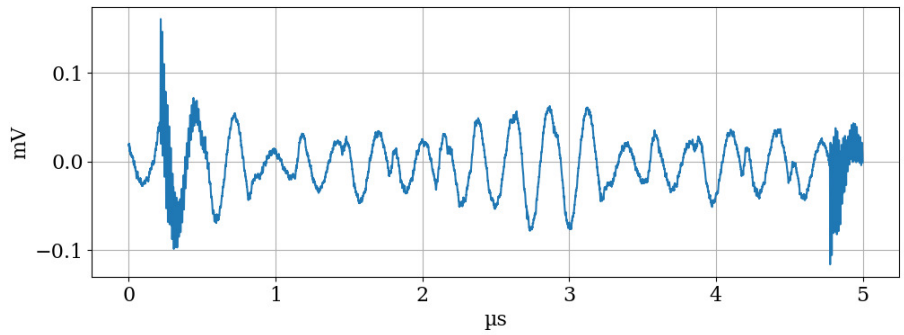


Figure 5.11 – Trace EM moyenne pour deux multiplications sur le STM32H7.

Le programme est compilé en 00 (gcc), et chaque multiplication est suivie d'une instruction *store* STR enregistrant le résultat $w \times x$ dans la SRAM. Comme le montre le tableau 5.4, notre méthode d'extraction permet de récupérer les paramètres avec une précision élevée.

Table 5.4 – Résultats d'extraction pour deux multiplications.

Valeur correcte	ϵ_{rr}
0.793281	1.87e-08
0.33147	1.27e-08

5.2.4 Extraction d'un neurone

Nous pouvons désormais passer à l'extraction des paramètres d'un seul neurone. La principale différence est que la sortie d'un neurone est l'accumulation des différentes multiplications entre les paramètres w_i et les entrées x_i du neurone (somme pondérée) qui sera ensuite activée. L'accumulation est traitée par deux instructions FPU successives (`fmul` puis `fadd`, comme dans nos expériences avec le Listing 5.2) ou une instruction dédiée multiplication-addition.

Pour pouvoir extraire les paramètres, les uns après les autres, il faut tout d'abord connaître l'ordre des opérations, et faire des hypothèses intermédiaires sur le résultat de l'accumulation.

```
;layer1_neuron_res[i] += input[j] * weight_layer1[i][j];
[...]                 ; r3 = accumulator address (SRAM)
vldr      s14, [r3]   ; Load accumulator
[...]                 ; r3 = input address (SRAM)
vldr      s13, [r3]   ; Load input
[...]                 ; r3 = weight address (FLASH)
vldr      s15, [r3]   ; Load weight
vmul.f32  s15, s13, s15  ; Multiplication (FPU)
vadd.f32  s15, s14, s15  ; Addition (FPU)
[...]                 ; r3 = accumulator address (SRAM)
vstr      s15, [r3]   ; store result
```

Listing 5.2 – Code assembleur de la multiplication-accumulation avec FPU.

Comme discuté préalablement, nous faisons l'hypothèse que l'attaquant connaît l'ordre de calcul. L'extraction du premier paramètre w_0, à partir de la multiplication $w_0 \times x_0$, n'est pas problématique puisqu'elle correspond à notre précédente expérience. Une fois w_0 récupéré, la méthode d'extraction peut être appliquée pour w_1 en faisant des hypothèses non pas sur $w_1 \times x_1$ mais sur l'accumulation $w_0 \times x_0 + w_1 \times x_1$ car w_0 est connu. La précision de l'extraction de w_1 sera donc fortement dépendante de la qualité de celle de w_0.

Expériences sur Cortex-M7. Nous appliquons cette méthode sur 3000 traces moyennées capturant l'inférence d'un neurone à 4 paramètres. Le tableau 5.5 présente nos résultats, avec une extraction très précise pour les 4 paramètres et une erreur moyenne de 10^{-7}.

Table 5.5 – Erreur d'extraction (ϵ_{rr}) pour les 4 paramètres d'un neurone. Cortex-M7, 3,000 traces moyennées.

Paramètre	Valeur correcte	ϵ_{rr}
w_0	0.366193473339	5.96e-08
w_1	0.90820813179	3.58e-07
w_2	0.522847533226	5.96e-08
w_3	0.00123456	4.21e-08

5.2.5 Extraire le signe

Les expériences précédentes ont montré la pertinence de la méthode pour extraire la valeur absolue des paramètres d'un neurone. Néanmoins, il manque encore une information importante avec le bit de signe. Comme nous l'avons souligné, l'utilisation de la fonction d'activation ReLU a une influence considérable sur la distribution des entrées d'un neurone puisque celles qui sont négatives sont fixées à 0. Ainsi, il n'est pas possible de faire varier le signe du résultat de la multiplication en faisant varier les entrées du modèle. Néanmoins, on peut tirer parti de l'accumulation à l'échelle du neurone et de l'effet du signe de chaque paramètre. L'idée est donc de faire des hypothèses non pas sur chaque signe de chaque paramètre mais sur les alternances de signe pendant le calcul de la somme pondérée.

Nous pouvons illustrer ce principe avec deux paramètres, w_0 et w_1 et deux entrées x_0 et $x_1 \geq 0$. L'accumulation est donc $acc = w_0 \times x_0$, puis $acc = acc + w_1 \times x_1$. En ayant x_0 et x_1 strictement positives, les variations de acc ne sont pas les mêmes selon que $sign(w_0) \neq sign(w_1)$ ou $sign(w_0) = sign(w_1)$. Il est donc possible de faire des hypothèses sur $|acc|$ et de déterminer par CPA si w_1 possède le même signe que w_0, même si celui-ci reste inconnu pour le moment. A la fin

de l'accumulation du neurone, il reste à déterminer le signe du premier paramètre w_0 pour avoir l'intégralité des signes des autres paramètres. Cette vérification est faite simplement par rapport à la sortie de ReLU du neurone, c'est-à-dire que l'on suppose que les deux hypothèses sur $sign(w_0)$ vont conduire à deux sorties différentes du neurone.

Simulations. Nous générons des traces simulées à 50 dimensions pour un neurone à m entrées avec m choisi aléatoirement dans $[\![2;8]\!]$. Nous générons 5000 neurones avec m paramètres (signés), chacun traitant 3000 ensembles d'entrées positives (15 millions de traces au total). Pour construire les traces, nous positionnons uniformément les m points de fuite des accumulations ainsi qu'un point complémentaire pour la sortie du neurone (après ReLU). Les autres dimensions de la trace sont des valeurs tirées uniformément. Nous ajoutons un faible bruit gaussien sur toute la trace ($\sigma^2 = 0.5$). Nous obtenons 78.8% des neurones qui sont extraits avec tous les signes correctement associés et 91.6% des paramètres ont leur signe correctement attribué. Les résultats d'extraction des valeurs des paramètres sont dans le tableau 5.6.

Table 5.6 – Extraction d'un neurone avec des traces simulées. *Success-rate* (SR) de l'extraction des paramètres (avec un signe correctement retrouvé), trié par seuils de l'erreur d'estimation (ϵ_{rr}).

$\epsilon_{rr} \leq$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
SR	0.999	0.991	0.962	0.928	0.865	0.793	0.654	0.61

Expériences sur Cortex-M7. Nous utilisons 5000 traces moyennées pour l'inférence d'un neurone avec des paramètres signés. Avec le Tableau 5.7, nous observons que l'inversion du signe et les valeurs relatives ont été correctement appliquées.

Table 5.7 – Erreur d'extraction (ϵ_{rr}) pour les 4 paramètres signés d'un neurone sur Cortex-M7.

Paramètre	Valeur correcte	ϵ_{rr}	Signe correct
w_0	-0.813444	1.38e-07	✓
w_1	0.0671324	3.88e-08	✓
w_2	0.107843	2.34e-07	✓
w_3	0.604393	6.50e-08	✓

5.2.6 Couches de neurones

Pour le passage à l'échelle d'une couche de neurones, nous devons faire une autre hypothèse sur l'ordre des calculs des neurones pour rendre la CPA réalisable. Nous supposons donc que le calcul des neurones est effectué séquentiellement de haut en bas de la couche. Pour garantir qu'une valeur d'un paramètre déjà extraite ne soit pas associée de nouveau à un autre neurone, nous filtrons les positions temporelles des points de fuite pour qu'elles soient postérieures à celles du dernier paramètre extrait.

Sur le STM32H7, nous implémentons une couche de neurones avec 5 neurones et des entrées dans $\mathbb{R}^4$. Les 20 paramètres sont signés et les entrées utilisées sont aléatoirement positives ou négatives. A partir de 5,000 traces, l'application de notre méthodologie d'extraction permet d'obtenir une erreur d'extraction moyenne $\epsilon_{rr} = 1.03e^{-6}$, la plus mauvaise extraction donne $\epsilon_{rr} = 3.10e^{-6}$ et la plus précise est à $\epsilon_{rr} = 1.55e^{-7}$). Tous les signes des 20 paramètres ont été correctement associés.

Dans un second temps, nous déployons une succession de 5 couches denses possédant respectivement 5, 4, 3, 2 et 3 neurones. Pour ne se concentrer que sur la précision de l'extraction, nous simplifions le problème en ne considérant que des paramètres positifs et le modèle est alimenté avec des entrées positives dans $\mathbb{R}^4$. Nous avons donc un total de 64 paramètres. Après application de notre méthode, tous les poids ont été récupérés avec une erreur d'estimation inférieure à 10^{-6} ($SR = 95.31\%$ pour $\epsilon_{rr} < 10^{-7}$, meilleur $\epsilon_{rr} = 7.63e^{-10}$, pire $\epsilon_{rr} = 6.67e^{-6}$). Notons que ce modèle à 5 couches est totalement artificiel, il n'est pas le résultat d'un apprentissage pour réaliser une tâche particulière. Les paramètres sont simplement initialisés de façon aléatoire et se restreindre à des valeurs positives permet aussi d'éviter la présence de trop de neurones "morts" avec ReLU qui peut complètement éteindre des couches entières.

5.2.7 Le problème de l'extraction des biais

L'extraction du biais est un véritable défi pour pouvoir traiter des modèles fonctionnels de l'état de l'art. Si la somme pondérée entre les paramètres et les entrées est directement initialisée par la valeur du biais (comme dans CMSIS-NN [76]), notre méthodologie est applicable uniquement si l'on arrive à extraire le biais avec précision à partir de la première opération d'addition IEEE-754.

Pour illustrer la difficulté liée au biais, nous simulons des traces dans lesquelles le biais est ajouté à la fin du calcul du neurone : $\sum_j (w_{i,j} \times x_j) + b$. Nous simulons 5000 neurones avec m paramètres secrets $w_{0..m}$ (comme auparavant, m est choisi aléatoirement dans $[\![2;8]\!]$) ainsi qu'un biais secret b. 5000 traces simulées sont créées pour chaque neurone avec des entrées positives aléatoires. Les taux de succès (SR) sont présentés dans le tableau 5.8. Ces SR concernent uniquement les poids et le biais pour lesquels le signe a été correctement récupéré, soit 93.25% des paramètres et 92.14% des biais. Alors que le SR lié à l'extraction des poids reste cohérent avec les simulations précédentes, le SR correspondant à l'extraction du biais chute considérablement (par exemple, $SR = 35.8$ pour 10^{-3}).

L'explication réside dans l'opération d'addition IEEE-754 qui nécessite un alignement strict des exposants (contrairement à la multiplication). Ainsi, si $a \gg b$, alors $a + b = a$ car la valeur de b *disparaît* devant a. Dans notre contexte, comme les entrées x (contrôlées par l'attaquant qui vise à couvrir autant que possible la plage de valeurs `float32`) sont définies de manière aléatoire, puis multipliées par les paramètres, il est probable que $\sum_{j=0}^{n} w_j x_j \gg b$. Ainsi, l'information secrète liée au biais pourrait être difficilement récupérée en exploitant les traces EM par une simple CPA. Il est donc nécessaire de développer un modèle de fuite approprié pour l'addition et potentiellement de choisir des entrées dans une plage de valeur cohérente.

Table 5.8 – Extraction des valeurs des paramètres et des biais.

$\epsilon_{rr} \leq$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
SR weight	0.999	0.995	0.968	0.937	0.872	0.798	0.649	0.613
SR bias	0.817	0.563	0.358	0.197	0.0744	0.026	0.007	0.002

5.2.8 Conclusion

Avec ces expériences, nous montrons que l'extraction de l'ensemble des paramètres d'un modèle fonctionnel complet uniquement par CPA soulève de nombreuses difficultés dont beaucoup sont très peu considérées dans la littérature. Ces difficultés sont autant théoriques (extraction du signe, fuites de l'addition IEEE-754) que pratiques (alignement temporel complexe des analyses). Néanmoins, pour des modèles relativement simples (et sans biais), notre méthode basée sur des analyses itératives a démontré une précision d'extraction remarquable. Malgré ses limitations actuelles, l'analyse side-channel peut surtout apparaître comme une méthode complémentaire à d'autres approches de l'état de l'art, la combinaison d'approche étant une perspective intéressante pour étudier les capacités d'un attaquant évolué souhaitant réaliser une attaque d'extraction de modèle.

Dans la prochaine section, nous montrons que l'injection de faute est justement un vecteur d'attaque très puissant qui peut se combiner efficacement à des techniques standards d'apprentissage d'un modèle substitut.

5.3 Utilisation d'analyses par injection de fautes

5.3.1 Contexte de l'étude

Si de nombreux travaux ont utilisé les analyses *side-channel* pour des attaques d'extraction de modèle, l'injection de faute a été très peu étudiée dans cette configuration. La seule référence importante proposant une véritable exploitation de l'injection de faute pour l'extraction de modèles de l'état de l'art est DeepSteal [90]. DeepSteal concerne uniquement les modèles déployés sur des plateformes avec de la mémoire DRAM, car l'attaque repose sur l'injection de *bit-flips* qui sont réalisés par un mécanisme Rowhammer [84] (plus précisément, RamBleed [155]). Dans DeepSteal, l'objectif de l'attaquant est de construire un modèle substitut le plus fidèle possible du modèle victime dont il connaît l'architecture. L'attaque physique par injection de fautes ne

cherche pas à extraire parfaitement les valeurs de tous les paramètres du modèle victime, mais à retrouver les bits les plus significatifs de la majorité des paramètres du modèle. L'idée derrière DeepSteal est que ces informations parcellaires vont sensiblement améliorer l'apprentissage du modèle substitut, même dans un scénario où l'attaquant ne dispose que de très peu d'information sur la base d'apprentissage du modèle victime (i.e., un scénario dans lequel les techniques classiques par *active learning* sont limitées). Dans le modèle de menace initial de DeepSteal, l'attaquant dispose de moins de 10% de la base d'apprentissage du modèle victime et aucune donnée complémentaire. Pour contourner ce manque de donnée, l'apprentissage du modèle substitut est fortement contraint par les informations extraites sur les paramètres du modèles victimes qui vont agir comme une régularisation et significativement accélérer la convergence de l'apprentissage.

Si l'on s'intéresse aux déploiements de modèles sur des microcontrôleurs 32-bit, l'application directe de DeepSteal est impossible. Les plateformes que nous utilisons dans les travaux précédents possèdent des mémoires Flash et non DRAM et il n'est donc pas possible de réaliser des *bit-flips* par Rowhammer. Au contraire, nous avons démontré dans le chapitre 4 (4.2) la première attaque laser dans la mémoire Flash contre les paramètres d'un modèle embarqué à travers un modèle de fautes *bit-set*. L'objectif de cette étude est donc de trouver des stratégies alternatives permettant d'extraire des informations sur les valeurs des paramètres qui seraient stockés dans de la mémoire Flash d'un microcontrôleur 32-bit.

Pour cela nous proposons d'utiliser une technique d'exploitation d'injection de fautes principalement utilisée contre des implémentations cryptographiques, la *Safe Error Attack* (SEA). La SEA nous permet d'exploiter des fautes dites *data-dépendantes*, c'est-à-dire asymétrique comme les fautes *bit-set* ou *bit-reset*. Nous démontrons que le choix des données d'entrée a une influence importante sur l'efficacité de la SEA.

5.3.2 Modèle de menaces et formalisme

Nous considérons un attaquant qui cherche à voler les paramètres d'un modèle dont il connaît l'architecture afin de le copier ou de préparer des attaques futures. Pour reprendre la taxonomie de Jagielski *et al.* (Cf. 3.3.3.2, page 33), nous nous situons donc principalement dans un scénario dit de "fidélité". L'attaque cible un modèle DNN embarqué sur une cible de type microcontrôleur 32 bits et quantifié sur 8 bits. On se place dans un contexte classique pour l'extraction de modèles [43, 90] que l'on peut nommer "boîte grise" : l'attaquant connaît l'architecture du modèle victime mais pas ses paramètres et a accès à moins de 10% du jeu de données d'entraînement (ou des données très proches en termes de distribution). Rappelons que la connaissance de l'architecture n'est pas irréaliste car elle est soit déjà parfaitement connue, soit aisément identifiable ou extraire par d'autres moyens comme les analyses *side-channel* présentées dans la section 5.1. Un accès limité aux données d'entraînement correspond également à des applications réelles sans *benchmarks* disponibles publiquement, comme étudié dans de nombreuses méthodes d'extraction basées sur de l'*active learning* [33, 156–158].

L'attaquant peut questionner le modèle victime sans aucune contrainte ni limitation et collecter les prédictions de sorties sous la forme de vecteurs de scores normalisés, qui sont aussi quantifiés sous 8 bits. Enfin, il dispose de l'expertise nécessaire et des équipements adéquats pour réaliser des injections de fautes, par exemple d'un banc laser et même de plusieurs clones de la cible lui permettant de réaliser toutes les études préliminaires nécessaires à la caractérisation de la mémoire Flash. Comme nous l'avons vu dans la section 4.2 du chapitre 4, ce processus de profilage ne nécessite pas le programme d'inférence cible mais seulement des procédures simples de lecture/écriture en mémoire [88].

Formalisme. Dans la suite, nous allons utiliser le même formalise que dans les chapitres précédents. Le DNN victime est noté M_W avec W ses paramètres et le modèle substitut est noté M^s avec les paramètres W_s. M effectue une tâche de classification avec des entrées $x \in \mathcal{X}(\subset \mathbb{R}^d)$ et les prédictions de sortie $M_W(x) \in \mathbb{R}^K$, avec $\mathcal{X}$ le domaine des entrées et K le nombre de classes. Le label prédit est alors $y = \text{argmax}(M_W(x))$, et le label correct est noté y^*. Un ensemble d'entrées et de prédictions est respectivement noté X et Y. La *loss* est la *categorical cross-entropy* simplement notée $\mathcal{L}_{CE}$. Comme nous allons le décrire dans la suite, notre méthode est basée sur une SEA avec des fautes de type *bit-set* réalisées sur les paramètres du modèle victime. Les paramètres fautés sont notés $\widetilde{W}$ et $M_{\widetilde{W}}$ est le modèle fauté résultant et $\widetilde{Y}$ est un ensemble de prédictions fautées. Nos modèles sont quantifiés sur 8 bits avec des entiers signés (représentation en complément à

deux) : un paramètre w est représenté sous la forme $b_0 b_1 ... b_7$, avec b_i une valeur binaire et b_0 le bit de poids fort (MSB).

5.3.3 Setup expérimental

Plateforme de simulation. Un schéma de quantification classique revient à simplement transformer les paramètres sous 8 bits pour réduire l'empreinte mémoire du modèle. Lors de leur utilisation pour les calculs de l'inférence, ils sont remis à l'échelle en précision flottante. Les activations et les scores de prédictions ne sont donc pas quantifiés et restent en précision flottante 32 bits. Ce schéma est souvent utilisé en simulation pour sa simplicité mais ne reflète pas le comportement de systèmes embarqués réels. Pour notre étude, nous choisissons un schéma de quantification qui tient compte des activations et des scores de prédictions (aussi sous 8 bits) et qui offre un cadre plus difficile pour notre attaque basée sur la SEA.

Nous avons développé un *framework* Python basé sur PyTorch pour effectuer toutes nos simulations avec un schéma de quantification sur 8 bits correspondant à celui proposé dans NNoM [77] (librairie que nous avons déjà utilisée dans la section 3.6.2.2) et qui utilise comme *backend* la bibliothèque ARM CMSIS-NN [76]. Rappelons que NNoM est *open-source* avec un accès complet au code C et permet une quantification sur 8 bits pour les paramètres (dont les biais) et les fonctions d'activation avec un schéma de quantification symétrique uniforme en puissances de deux (équation 3.6, page 38). La quantification est aussi étendue aux scores de prédiction. Notre *framework* Python fournit ainsi les mêmes sorties (au niveau des couches et du modèle) que celles produites par NNoM sur des plateformes Cortex-M. Notons qu'une particularité (que nous avons conservée) est que NNoM ne travaille qu'avec des entiers signés sur 8 bits et ne met pas à l'échelle les scores de prédiction ($\in \mathbb{R}^+$) dans $[0, 255]$, mais dans $[0, 127]$.

Modèles et ensembles de données. Nous utilisons un MLP et un CNN. Le MLP est composé de trois couches entièrement connectées (128 - 64 - 10 neurones, sans biais) avec ReLU comme fonction d'activation. Le MLP est entraîné sur MNIST. Le CNN est composé de trois couches de convolution et d'une couche dense (sans biais). Ce modèle est une référence habituelle pour les modèles embarqués sur microcontrôleurs, présenté dans [76]. Il est entraîné sur CIFAR-10. Le tableau 5.9 détaille l'architecture des deux modèles.

Table 5.9 – Architecture du MLP (MNIST) et CNN (Cifar-10). NB : nous suivons la dénomination de PyTorch pour les couches denses avec "*Linear*". Les couches de convolution sont composées de filtres 5×5 et suivies par du *Pooling* (*Average*, 2x2).

Couche	# paramètres	Couche	# paramètres
Inputs (784)		Inputs (32,32)	
Linear 1 (128 neurones), ReLU	100352	Conv1 (32 filtres), ReLU	2400
Linear 2 (64 neurones), ReLU	8192	Conv2 (32 filtres), ReLU	25600
Linear 3 (10 neurones), Softmax	640	Conv3 (64 filtres), ReLU	51200
		Linear (10 neurones), Softmax	10240
MLP	109184	CNN	89440

5.3.4 Extraction de modèle avec la SEA

5.3.4.1 Principe de la méthode

Le principe général de notre méthode d'extraction repose sur trois étapes que nous illustrons avec la figure 5.12. Le cœur de notre approche est la SEA qui, à partir de fautes précises sur les différents paramètres du modèle victime, va permettre d'extraire des informations sur des bits de certains paramètres. La SEA repose sur un ensemble d'attaque qu'une première étape cherche à optimiser pour significativement améliorer le nombre de bits extraits par la SEA. Enfin, la dernière étape est l'utilisation des informations extraites sur la valeur des paramètres pour entraîner un modèle substitut le plus fidèlement possible au modèle victime. Cette dernière phase est identique au *Mean Clustering Training* proposé dans DeepSteal [90].

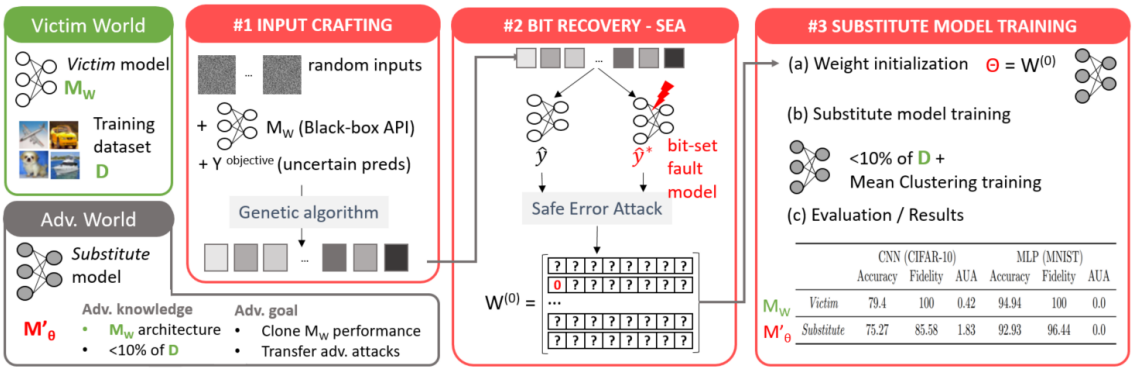


FIGURE 5.12 – L'adversaire fabrique des entrées et réalise une *safe error attack* en exploitant des prédictions fautées résultant de fautes *bit-set* sur les paramètres stockés en mémoire. L'objectif est de récupérer partiellement les bits des paramètres afin d'entraîner efficacement un modèle substitut qui imite le modèle victime avec une haute fidélité.

5.3.4.2 Exploitation de la *Safe-Error Attack*

La *Safe Error Attack* [159] repose sur le principe que la simple observation de la sortie d'un système qui a subi une faute (*la sortie est-elle identique à celle produite par le système en l'absence de faute ?*) peut donner des informations sur la valeur d'une donnée secrète. La SEA a principalement été appliquée dans des applications cryptographiques. Une illustration classique et naïve de son fonctionnement concerne une tâche de récupération de clé secrète pour un algorithme de chiffrement. A partir d'un message clair, l'algorithme produit un message chiffré en utilisant la clé secrète. L'attaquant utilise un message P et collecte le chiffré correspondant C en utilisant l'algorithme dans des conditions nominales. Il recommence l'opération mais en réalisant une faute de type *bit-set* sur le premier bit de la clé. Si le texte chiffré obtenu est erroné alors le bit de la clé est 0 (une véritable faute $0 \rightarrow 1$ a été injectée), s'il est exempt d'erreur, le bit de la clé est 1 (parce que le bit de la clé était effectivement protégé $1 \rightarrow 1$). En itérant l'attaque pour chaque bit, toute la clé peut être retrouvée.

Dans notre cas, les données fautées seront les bits des paramètres du modèle victime pendant l'inférence et la sortie observée sera simplement la prédiction du modèle sous la forme d'un score de prédiction. L'utilisation de la SEA est pertinente dans notre cas, car nous avons déjà démontré que l'on pouvait injecter des fautes de type *bit-set* avec un laser dans la mémoire Flash d'un microcontrôleur faisant tourner l'inférence d'un modèle.

Sur un ensemble d'entrées de test, noté X, la SEA repose sur la comparaison directe entre les prédictions Y du modèle victime M_W et celles $\widetilde{Y}$ produites par le modèle victime fauté $M_{\widetilde{W}}$. Nous vérifions ensuite la similarité entre Y et $\widetilde{Y}$, notée $S(Y, \widetilde{Y})$. $S(Y, \widetilde{Y})$ est `True` si nous avons une égalité stricte entre les deux matrices de scores, `False` sinon. Dans le contexte *bit-set*, trois cas sont possibles comme nous le résumons dans la table de vérité 5.10. Lorsque la faute est appliquée à un bit ayant la valeur 0, la faute peut conduire à un ensemble de prédictions différent du comportement normal ($Y \neq \widetilde{Y}$), donc l'adversaire peut conclure avec certitude que le bit est 0. Si la faute n'a pas d'impact sur les prédictions (par exemple le bit fauté n'est pas assez significatif) *ou* si le bit est déjà à 1, alors l'adversaire ne peut pas récupérer la valeur du bit. Ainsi, un objectif pour l'attaquant est d'optimiser la propagation de l'erreur lorsqu'une faute se produit ($0 \rightarrow 1$) afin que la faute ait plus de chances de provoquer une différence entre Y et $\widetilde{Y}$. Nous analysons comment la sélection des entrées peut être exploitée par un adversaire pour atteindre cet objectif dans la section suivante.

TABLE 5.10 – Table de vérité de la SEA

b_i	$\widetilde{b_i}$	$S(Y, \widetilde{Y})$	b_i estimation
0	1	False	0
0	1	True	?
1	1	True	?

5.3.4.3 Efficacité de l'ensemble d'attaque sur la SEA

Dans une première expérience, nous utilisons directement les données de l'ensemble de test de MNIST et CIFAR-10 pour constituer l'ensemble d'attaque. Les premiers résultats offrent une très forte hétérogénéité de l'efficacité de la SEA. Pour certaines données, les fautes ne perturbent pas les scores de prédiction quand pour d'autres les modifications sont très importantes et permettent donc d'extraire avec certitude plus de bits d'information. Expérimentalement, on distingue deux catégories de données en fonction de leurs scores de prédiction. Une première classe (ci-après appelée *Certain*) représente les entrées qui conduisent à des prédictions avec un seul label possédant le score maximal de 127 (e.g., $[0, 0, 0, 0, 0, 0, 0, 0, 0, 127]$). Par simplicité, nous appellerons ces données "*Certaines*". La deuxième classe (appelée "*Incertain*") correspond aux données qui conduisent à des scores de prédiction avec au moins deux labels ayant un score non nul (par exemple, $[0, 13, 0, 0, 0, 0, 0, 4, 0, 110]$). Le tableau 5.11 présente la précision des modèles MLP et CNN en fonction de ces catégories. La performance des modèles est naturellement plus forte sur les données *certaines*, au-dessus de 90% sur CIFAR-10 et proche de 100% sur MNIST. La plupart des erreurs de prédiction sont concentrées dans la classe *incertaine*.

Table 5.11 – *Accuracy* (%) selon la catégorie des entrées

Modèle	Tous	Incertain	Certain
CNN	79.4	57.07	92.23
MLP	94.94	68.07	98.2

Le tableau 5.13 présente la différence entre ces deux catégories d'entrées pour le modèle MLP et CNN, avec les détails à l'échelle des couches. La première colonne donne la proportion (%) d'entrées Certaines et Incertaines qui ne conduisent à aucune récupération de bit. Par exemple, pour le CNN, 14.23% des entrées Certaines ne conduisent à aucune récupération de bit (c'est-à-dire que les prédictions entre les modèles sans erreur et fautés sont strictement identiques sur ces données). La dernière colonne donne le nombre moyen de bits récupérés en utilisant la SEA sur ces entrées : pour un modèle, nous utilisons m entrées Certaines et m entrées Incertaines et calculons le nombre moyen de bits récupérés (et l'écart type) sur ces m entrées. Nous utilisons $m = 2000$ pour le MLP et $m = 3000$ pour le CNN. Par exemple, pour le MLP, les entrées de la catégorie Incertaine permettent d'extraire (en moyenne) 64 fois plus de bits que celles de la classe Certaine (8438 contre 131).

Une première observation est que les entrées Incertaines divulguent toujours des informations sur les bits et que les entrées *inutiles* pour la SEA sont surreprésentées dans la catégorie Certaine pour quelques couches des deux modèles. Ce résultat est plus disparate pour le modèle CNN, car les paramètres de la première couche de convolution sont plus facilement récupérés, quel que soit le type d'entrée (seulement 14.33% des prédictions Certaines sont inutiles pour la SEA). De plus, les données Incertaines permettent de récupérer, en moyenne, un nombre de bits d'information sur les paramètres significativement plus élevé que pour les entrées Certaines, avec un facteur 33 pour le CNN et 64 pour le MLP.

Table 5.12 – Type de prédictions (Certaines, Incertaines) sur 5,000 données aléatoires.

	MLP	CNN
C (%)	9.82	99.4
I (%)	90.18	0.06

Nous proposons un examen plus approfondi de ce phénomène en analysant la distribution (par couche) des valeurs absolues des gradients $\nabla_W \mathcal{L}_{CE}$, représentée dans la figure 5.13. Nous observons une différence claire de la sensibilité de la *loss* entre les deux catégories, avec des magnitudes élevées pour la classe Incertaine : pour les entrées conduisant à des prédictions incertaines, une modification des paramètres affectera fortement la valeur de la *loss* et, par conséquent, les prédictions ce qui est souhaité pour une meilleure performance de la SEA.

Cette première expérience démontre l'influence très importante de la nature de l'ensemble d'attaque sur l'exploitation d'une SEA pour récupérer des valeurs de bits. En se plaçant dans

TABLE 5.13 – Efficacité de l'extraction pour des données Certaines (**C**) et Incertaines **I** (CNN : $m = 3000$, MLP : $m = 2000$).

Couches	Taux $S(Y, \widetilde{Y})$ =True (%)		Moyenne # de bits retrouvés (std)	
	Certain	Incertain	Certain	Incertain
Linear 1	97.15	0.0	83	5976
Linear 2	92.6	0.0	40	2229
Linear 3	48.95	0.0	8	231
MLP	47.65	0.0	131	8438
Conv. 1	14.33	0.0	399	7144
Conv. 2	68.6	0.0	895	32380
Conv. 3	79.5	0.0	339	15062
Linear 1	67.5	0.0	24	803
CNN	14.23	0.0	1656	55388

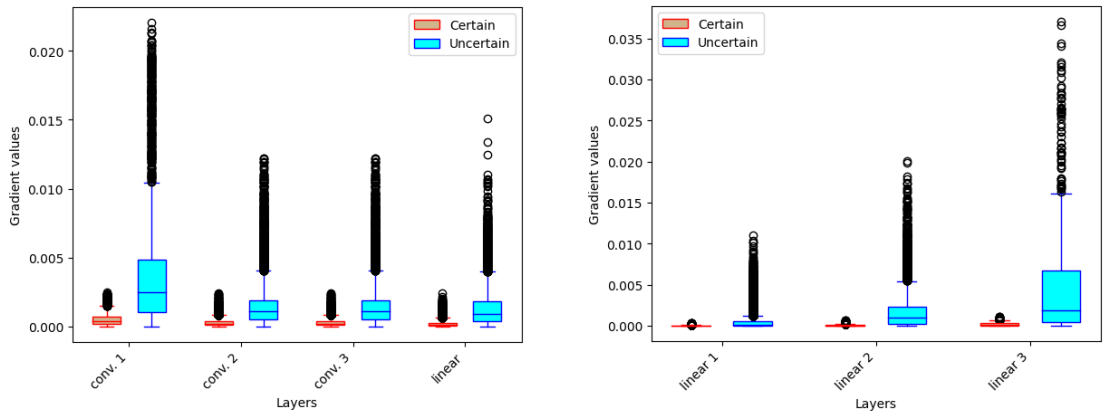
un modèle de menace où l'adversaire a un accès très limité à l'ensemble de données d'entraînement, ces résultats ouvrent la voie à une stratégie visant à générer des entrées conduisant à des prédictions *incertaines*.

5.3.4.4 Construction de données *incertaines*

Entrées aléatoires. La première question concerne l'efficacité des entrées générées aléatoirement. Avec un accès très limité à l'ensemble de données d'entraînement, l'utilisation d'entrées aléatoires illimitées pourrait constituer un réel avantage pour l'attaquant. Le tableau 5.12 montre la répartition entre les deux catégories de prédictions pour 5,000 entrées aléatoires suivant une distribution uniforme. Pour le modèle MLP, les entrées aléatoires permettent de générer un ensemble d'attaque avec 90,18% des entrées de type Incertaines. Néanmoins, pour le modèle CNN, seules des prédictions de type Certaine sont obtenues. En effet, nous constatons que les entrées aléatoires sont majoritairement associées au même label (classe 7 de CIFAR-10). Notons que nous observons le même comportement pour une autre architecture de CNN entraînée sur CIFAR-10 (VGG-8 avec ou sans biais).

Nous proposons alors une autre approche (boîte noire) pour créer des entrées de type Incertaine en utilisant un algorithme génétique.

Méthode de création boîte noire. Notre objectif est d'utiliser un algorithme génétique (AG) pour créer directement un ensemble d'attaque composé de données Incertaines. Notre GA va alterner différentes opérations à partir d'entrée générées aléatoirement jusqu'à atteindre un objectif



(a) CNN (b) MLP

FIGURE 5.13 – Distribution de $\nabla_w \mathcal{L}$ par couche pour des données Certaines et Incertaines. Les *boxplot* représentent la valeur médiane entre le premier et troisième quartile. La ligne bleue étend la boîte par 1.5x et les cercles noirs sont les *outliers*.

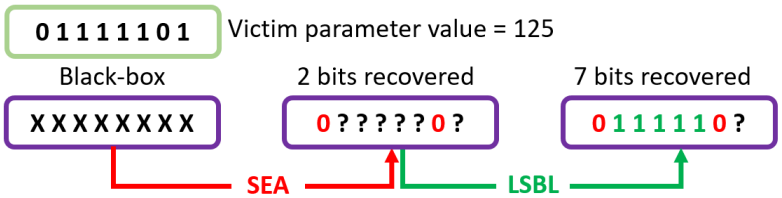


FIGURE 5.14 – Illustration du principe LSBL.

de prédiction. Les étapes de l'algorithme sont les suivantes :

1. *Initialisation de la population*. Un premier ensemble de données (appelé *population*) est constitué en générant des données aléatoirement avec une distribution uniforme des pixels dans $[V_{min}, V_{max}] \subset [-127;127]$. Plusieurs valeurs de (V_{min}, V_{max}) peuvent être fixées pour avoir plusieurs populations initiales et augmenter la diversité.
2. *Définition de l'objectif.* L'objectif est d'obtenir des données qui conduisent à des *scores cibles* Y^t de prédictions incertaines. Un score cible, y^t, est dans $\mathbb{R}^K$ avec $y^t(i) \in [0;127]$. C scores parmi K sont choisis au hasard et mis à 0, avec $C \in [0;K-2]$. Les $K-C$ scores restants non nuls sont définis aléatoirement et normalisé (Softmax) pour que $\sum_i y^t(i) = 127$.
3. *Évolution de la population.* De façon itérative, on construit une nouvelle population grâce à un ensemble de transformations élémentaires. À l'itération t, la nouvelle population résulte d'opérations de *sélection*, *croisement* et *mutation* entre les éléments de la population précédente. Pour déterminer quelle opération doit être effectuée sur chaque élément de la population, on trie les éléments par rapport à la loss $\mathcal{L}_{CE}$ (selon notre objectif de scores cibles). La *sélection* consiste à garder certains éléments de la génération précédente à $t-1$, la *mutation* applique un bruit gaussien sur les éléments sous-optimaux de la génération précédente et le *croisement* fusionne deux éléments de la génération précédente en permutant aléatoirement des parties de chaque élément.

Les itérations de la dernière étape se répètent jusqu'à obtenir un nombre d'éléments souhaités pour l'ensemble d'attaque et répondant aux scores cibles.

Dans la figure 5.15 (à gauche), nous observons sur le CNN que cette stratégie de création par algorithme génétique est bien plus efficace que par rapport à des entrées purement aléatoires (courbes pointillées). Ainsi, nous arrivons à extraire 80% et 90% du MSB des paramètres du CNN avec seulement 150 et 1500 entrées, respectivement.

5.3.4.5 Principe du *Least Significant Bit Leakage* (LSBL)

La table de vérité 5.10 montre l'ambiguïté de la SEA par rapport au modèle de faute *bit-set*. Cependant, la position du bit extrait apporte une information qui peut soulever le doute dans certains cas où les deux ensembles de prédictions Y et $\widetilde{Y}$ sont identiques. Nous appelons ce principe LSBL pour *Least Significant Bit Leakage*. L'idée est que si un bit b_k avec $k \in]0;7]$ d'un paramètre w a été récupéré par la SEA (donc $b_k = 0$ sans ambiguïté), alors tous les bits antérieurs b_i avec $i \in [0;k-1]$ qui sont indéfinis peuvent être estimés à 1.

LSBL s'explique par le fait que plus un *bit-set* est effectué sur un bit moins significatif que k, plus la variation du paramètre est faible. Ainsi, si une petite altération (à l'indice de bit k) modifie la prédiction, alors une altération plus importante (indice avant k) devrait également modifier Y (et même encore plus). Ce principe permet d'estimer les bits b_i, car si ces bits n'ont pas modifié la prédiction, c'est que leur valeur de bit est déjà à 1 (i.e., la faute ne modifie pas la valeur de w). La figure 5.14 illustre le LSBL avec un paramètre $w = 125$. Avec la SEA, nous récupérons b_0 et b_6. L'application du LSBL permet de déduire b_1 à b_5. Grâce au principe LSBL, le taux de bits récupérés augmente de 47,05% à 80,1% pour le modèle CNN avec 5,000 entrées générées par l'algorithme génétique. Par rapport à l'ensemble de nos expériences et évaluations sur le MLP et le CNN, l'erreur d'estimation des bits extraits uniquement par le LSBL est strictement inférieure à 1%. Cette heuristique permet donc de récupérer les valeurs supplémentaires de bits avec un taux d'erreur très marginal et qui n'a pas de conséquence sur l'efficacité générale de la méthode.

Dans la figure 5.15, nous montrons le pourcentage de bits récupérés en combinant la SEA et le principe LSBL en fonction du nombre d'entrées dans l'ensemble d'attaque. Nous proposons à la fois des entrées aléatoires et des entrées générées par l'algorithme génétique pour le CNN. Seules les données aléatoires sont présentées pour le MLP (les résultats sont identiques quand on utilise l'algorithme génétique). Nos résultats démontrent un taux élevé de bits récupérés (jusqu'à 90%

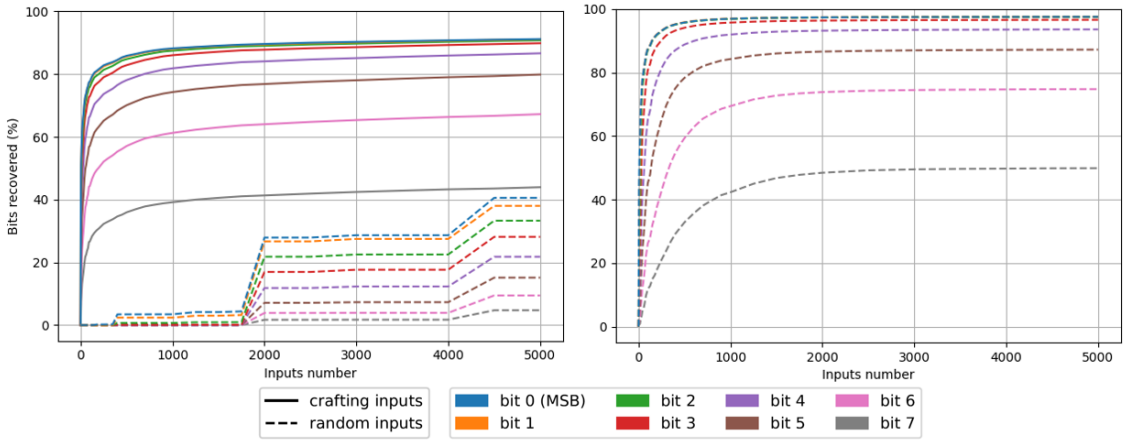


FIGURE 5.15 – Bits récupérés avec SEA et LSBL pour (à gauche) CNN et (à droite) MLP (entrées aléatoires uniquement).

des MSB) pour les deux modèles grâce à la SEA associée au principe LSBL. Notre méthode est également efficace pour les 6 bits les plus significatifs avec un taux de récupération supérieur à 80%. On remarque un effet plateau qui intervient rapidement puisque la majorité (80%) des bits récupérés le sont avec environ 250 entrées (CNN), puis, après 1500 entrées (CNN), l'utilisation de plus d'entrées permet uniquement de récupérer quelques bits supplémentaires. Le même effet est observé pour MLP.

5.3.4.6 Entraînement du modèle substitut

Pour l'entraînement du modèle substitut, à partir d'une faible portion des données d'apprentissage du modèle cible (8%) et des informations extraites par la SEA, nous utilisons la méthode proposée dans DeepSteal [90], appelée *Mean Clustering Training*. Notons que, sans aucun entraînement, c'est-à-dire si nous définissons simplement les bits récupérés à leurs valeurs estimées et initialisons aléatoirement tous les autres bits, le modèle substitut atteint une très faible performance avec une *accuracy* moyenne de 26,02% pour le CNN et 75,78% pour le MLP sur l'ensemble de test (avec au moins 90% des bits les plus significatifs récupérés). De la même manière, l'apprentissage du modèle substitut uniquement sur 8% des données d'apprentissage et sans utiliser les informations issues de la SEA (et LSBL) conduit à une absence notable de convergence et des problèmes sévères d'*overfitting*.

Comme dans [90], nous utilisons une nouvelle *loss* (notée $\mathcal{L}^{sub}$) qui ajoute à la *loss* standard de classification (*cross-entropy*, $\mathcal{L}_{CE}$) un terme de pénalité qui contraint les paramètres partiellement récupérés. L'apprentissage utilise un échantillonnage de seulement 8% de la base d'apprentissage et cherche donc classiquement à minimiser $\mathcal{L}^{sub}$, exprimée dans l'équation 5.4 :

$$\mathcal{L}^{sub} = \mathcal{L}_{CE}\Big(M_{\mathbf{W}}(x), y\Big) + \lambda \sum_{l=1}^{L} \|\mathbf{W}^l - \mathbf{W}^l_{mean}\| \tag{5.4}$$

avec L le nombre de couches, $\mathbf{W}^l$ est la matrice des paramètres associés à la couche l, $\mathbf{W}^l_{mean} = (\mathbf{W}^l_{min} + \mathbf{W}^l_{max})/2$ représentent les valeurs moyennes selon les valeurs *min* et *max*, mises à jour après chaque itération d'entraînement, et λ est un hyperparamètre équilibrant la force de la pénalité.

Les valeurs initiales sont calculées avec les MSB récupérés qui définissent les intervalles possibles des valeurs des paramètres partiellement extraits. Par exemple, si les deux premiers bits les plus significatifs de w sont 0, alors l'intervalle des valeurs possibles de w est $[0;63]$. Pour chaque *batch* d'apprentissage, l'algorithme met à jour W_{mean} avec les valeurs actuelles et les paramètres sont seuillés (*clipping*) en fonction de l'intervalle après chaque époque d'entraînement. L'objectif principal est d'éviter toute divergence des paramètres par rapport à l'information extraite par l'étape d'injection de défauts. Cette procédure d'entraînement est uniquement appliquée aux paramètres partiellement récupérés par la SEA. Pour ceux pour lesquels aucun bit n'a été extrait, aucune pénalité ne peut être appliquée et donc $\lambda = 0$. Au contraire, les paramètres entièrement récupérés (8 bits extraits) sont simplement gelés pendant l'entraînement. Comme démontré dans [90], minimiser $\mathcal{L}^{sub}$ ajoute suffisamment de contraintes pour stabiliser la convergence de l'apprentissage malgré le peu de données d'entraînement.

5.3.4.7 Évaluation

Pour avoir des résultats comparables à [90], nous utilisons le même ensemble de métriques. Le premier critère est lié à la performance du modèle substitut qui doit être proche de celle du modèle victime. Nous calculons donc simplement l'*accuracy* moyenne du modèle substitut sur l'ensemble de test de MNIST (pour MLP) et CIFAR-10 (pour CNN). Le deuxième critère correspond à notre scénario d'attaque d'extraction de modèle. Il s'agit de mesurer la *fidélité* entre les modèles substitut et victime, c'est-à-dire le taux de prédictions identiques (par rapport au label prédit) sur l'ensemble de test. Une fidélité à 100% indique que le modèle substitut prédit exactement le même label que le modèle victime sur l'ensemble de test, même si le label prédit est faux. Le dernier critère est l'*accuracy sous attaque* (AUA), qui correspond à la précision du modèle victime lorsqu'il est testé avec des *adversarial examples* construits à partir du modèle substitut. Si le modèle substitut parvient à imiter les comportements du modèle victime, alors les deux modèles répondront vraisemblablement de manière similaire lorsqu'ils seront confrontés à des attaques par évasion : la transférabilité des perturbations adverses sera maximale et nous nous attendons à une très faible AUA tant pour le modèle victime que pour le modèle substitut. Conformément à l'état de l'art, nous avons utilisé l'attaque l_∞-PGD [135] avec 40 étapes et un budget d'attaque de $\epsilon = 8/255$ pour CIFAR-10 et $\epsilon = 0.3$ pour MNIST.

Table 5.14 – Performance du modèle substitut. AUA : *Accuracy Under Attack*

Au moins % MSB extraits (+ autres si retrouvés)	CNN			MLP		
	Accuracy	Fidélité	AUA	*Accuracy*	Fidélité	AUA
90	75.27	85.58	1.83	92.93	96.44	0.0
80	69.36	77.00	5.55	92.09	95.48	0.01
70	54.59	61.10	12.99	90.52	93.66	0.1
60	40.55	44.66	34.84	64.50	66.56	12.50
Victime	79.4	100	0.42	94.94	100	0.0

Nos résultats sont résumés dans le tableau 5.14 en fonction du taux de récupération des MSB. Pour le CNN, la précision passe de 40,55% à 75,27% lorsque le taux de récupération des MSB augmente de 60% à 90% (64,50% à 92,93% pour le MLP). Nos meilleurs résultats pour le CNN et le MLP représentent une chute de précision de seulement 4% et 2%, respectivement, par rapport aux modèles victimes. Il est important de noter que nous atteignons également un taux de fidélité élevé avec 85,58% (CNN) et 96,44% (MLP) dans le meilleur cas (90% des MSB récupérés). En nous concentrant sur les résultats du CNN, la performance de notre approche est cohérente avec celles observées dans [90] avec différentes architectures [10]. Les résultats de l'AUA montrent que des *adversarial examples* peuvent être générés efficacement à partir des modèles substituts, puis appliqués sur les modèles victimes. Avec la meilleure quantité de MSB récupérée, les modèles victimes ont une AUA proche de celle obtenue en générant des exemples adverses dans un contexte boîte blanche (1.83% pour CNN et 0% pour MLP). En général, quel que soit le nombre de MSB utilisés pour entraîner les modèles substituts, les *adversarial examples* générés sur les modèles substituts sont suffisamment efficaces sur les modèles victimes pour obtenir une précision adverse inférieure à 35%.

5.3.5 Premières expériences pratiques avec injection laser

Dans le chapitre 4 (4.2), nous avons montré qu'il est possible de modifier très précisément la valeur de certains bits d'un paramètre d'un modèle DNN embarqué sur un Cortex-M, en utilisant des injections laser dans la mémoire Flash [88]. Ces injections conduisent à des fautes de type *bit-set*. Nous pouvons donc reprendre le *setup* expérimental et l'appliquer à notre méthodologie d'extraction.

Nous utilisons le même MLP que celui du chapitre 4 avec deux couches denses (50 - 10 neurones) entraînées sur MNIST (une réduction de dimension par ACP projette les images dans $\mathbb{R}^{50}$). Pour cette première expérience, on simplifie la mise en place de l'attaque en ajoutant un *trig-*

10. Sur CIFAR-10, avec 90% des MSB récupérés, [90] a construit un modèle substitut ResNet-18 avec une précision de 89.59% (victime : 93.16%), une fidélité de 91.6% et AUA de 1.61 (victime : 0%), ainsi qu'un modèle substitut VGG-11 avec une précision de 81.56% (victime : 89.96%), une fidélité de 83.33% et AUA de 18.55% (victime : 4.63%).

ger logiciel pour le déclenchement des tirs laser. Une autre simplification est la compilation du programme d'inférence sans optimisation (00).

L'attaque permet de récupérer au moins 90% des MSB en utilisant seulement 15 entrées (générées par l'algorithme génétique). L'attaque laser est très précise et répétable. Néanmoins, la principale limitation est que l'attaque nécessite beaucoup d'inférences (pour les injections de faute par bit et pour chaque entrée de l'ensemble d'attaque). Ainsi, le temps total de la SEA est donné par $T_{SEA} = N_{inputs} \times N_{bits} \times \delta_{inf}$, avec δ_{inf} le temps d'inférence. Pour cette expérience, δ_{inf} est un vrai problème puisque, sans aucune optimisation, nous obtenons $\delta_{inf} = 150$ ms en moyenne. L'attaque complète (*brute-force* sur tous les bits), dure donc 3 heures.

Cependant, il n'est pas nécessaire de cibler chaque bit du modèle. La durée de l'attaque peut être réduite, au moins, de moitié en ne ciblant que les 4 bits les plus significatifs, voire même à 20 minutes en ne prenant que le MSB comme dans [90]. L'utilisation d'autres librairies de déploiement plus optimisés (mais plus complexe à caractériser pour une attaque par injection de fautes) comme MCUNet [62] ou AIfES [160] permettrait aussi de réduire significativement le temps total de l'attaque. Notons que la comparaison de la complexité avec [90] est difficile, car les plateformes DRAM-CPU de [90] peuvent supporter des modèles beaucoup plus complexes que les microcontrôleurs 32 bits que nous utilisons [11].

5.3.6 Protection par ajout d'aléas

Les principales protections contre les attaques par injections de fautes incluent traditionnellement des mécanismes de randomisation, de la redondance ainsi que la vérification de l'intégrité des données ou du flot d'instructions. Cependant, ces techniques peuvent s'avérer incompatible avec la protection de la totalité d'un programme d'inférence.

Comme notre méthode d'extraction repose sur une SEA qui exploite les scores de prédiction, une protection logique est d'ajouter de l'aléa dans le modèle pour perturber les scores de prédiction sans altérer la performance globale du modèle. Par exemple, nous pouvons ajouter, à chaque inférence, un léger bruit aléatoire à la sortie de la dernière couche de convolution de notre CNN (64 filtres) dont le tenseur de sortie est de dimension $[8, 8, 64]$. A chaque inférence, nous échantillonnons, selon une loi Normale, 8 paramètres $\beta_i \subset [0.9, 1]$, chacun étant appliqué sur 8 canaux du tenseur de sortie. Cet ajout d'aléa ne baisse pas la performance moyenne du modèle (baisse d'*accuracy* de 0.1%) mais modifie suffisamment les scores de prédiction rendant la SEA inopérante.

Table 5.15 – Influence d'une moyenne sur N inférences pour l'ajout d'aléa dans la couche de convolution (CNN). Moyenne (std) des scores entre deux ensembles de prédictions (5,000 données générées).

	Moyenne sur N inférences par entrée			
N=	2	10	100	1000
ΔY (std)	7.7 (12.3)	2.4 (3.7)	0.8 (1.3)	0.3 (0.6)

Comme pour toutes défenses reposant sur de la randomisation, il est nécessaire de l'évaluer par rapport à un attaquant qui adapterait sa méthodologie pour contourner la protection (*adaptive attack*). La méthode la plus simple est de répéter N fois chaque inférence et d'effectuer une moyenne des scores pour diluer l'effet de l'aléa. Dans le tableau 5.15, nous représentons la moyenne et l'écart-type de la différence entre deux groupes de scores de prédiction (pour 5000 entrées) où chaque entrée est répétée N fois. Le score est la moyenne pour les N inférences et la différence entre les deux groupes de scores est moyennée sur les 10 labels et simplement notée ΔY. L'effet de la randomisation est effectivement dilué à mesure que N augmente et est quasiment nul pour $N = 1000$. Néanmoins, traitant d'injection de fautes, la répétition de chaque inférence 1000 fois paraît totalement rédhibitoire quel que soit le moyen d'injection (Rowhammer, laser...). L'ajout d'aléa apparaît donc comme une stratégie intéressante dès lors qu'elle ne perturbe pas la performance du modèle.

11. Dans [90], les auteurs ont utilisé ResNet-18, -34 et VGG-11 sur Cifar-10 avec un temps moyen d'extraction de 12 jours (extraction du premier MSB)

5.4 Conclusion et discussions

Complémentarité des approches d'extraction de modèle. Les études présentées dans ce chapitre montrent que les attaques physiques sont de puissants vecteurs d'attaques pour extraire des informations sensibles sur un modèle confidentiel. En premier lieu, les analyses *side-channel* sont des outils relativement simples et efficaces pour extraire l'architecture d'un DNN. Un code d'inférence fait intervenir de nombreuses répétitions de calculs bien spécifiques (multiplication-accumulation, activations non linéaires) qui se traduisent par l'apparition de motifs EM qu'il est aisé de détecter, de dénombrer, puis de relier à plusieurs hyper-paramètres du modèle. Néanmoins, l'utilisation d'une analyse *side-channel* standard, comme la CPA, pour extraire les valeurs secrètes des paramètres s'avère particulièrement complexe, dès lors qu'il s'agit de modèle de l'état de l'art. Même si nos premiers résultats démontrent une grande précision dans l'extraction de paramètres 32 bits pour des modèles de taille restreinte et sans biais, les difficultés théoriques et pratiques sont nombreuses et méritent plusieurs études complémentaires pour – entre autres – l'extraction des biais, la prise en compte de couches de convolution ou l'extraction de paramètres quantifiés. Enfin, nous avons montré que l'application d'une technique d'exploitation d'injection de fautes (*Safe-Error Attack*), contre l'inférence d'un modèle quantifié sur 8 bits, est une approche performante qui, quand elle est couplée à une méthode d'apprentissage de modèle substitut, permet d'atteindre des performances de l'état de l'art. Cette dernière étude met surtout en lumière la forte complémentarité des attaques physiques par rapport aux autres méthodes de la littérature. Une perspective très importante est donc la démonstration d'une combinaison de ces approches : les analyses *side-channel* et l'injection de fautes peuvent apporter de nombreuses informations sur le modèle victime (architecture, valeurs complète ou partielle de certains paramètres), qui peuvent être complétées par des approches par cryptanalyse [43–45] pour faciliter l'apprentissage d'un modèle substitut avec très peu de données d'entraînement.

Périmètres des modèles et des plateformes. Les principales limitations de l'ensemble des travaux présentés dans ce chapitre concernent la complexité des modèles étudiés et des plateformes matérielles, surtout pour les premières expérimentations d'injection de fautes laser contre les paramètres en mémoire Flash. Pour les analyses *side-channel*, nous avons choisi une cible correspondant à une plateforme Cortex-M parmi les plus puissantes (Cortex-M7, STM32H7) pour de l'IA embarquée, au moment des études. Le choix de nos modèles s'explique d'une part par ce qui est effectivement utilisé dans l'état de l'art et par la volonté d'étudier les couches les plus standards (dense, convolution, *pooling*) dans le DL moderne. A partir de ces travaux, il est possible d'étudier des architectures plus complexe (ResNet, Transformer...) et d'adapter nos méthodologies pour répondre aux spécificités de ces modèles. Il nous paraît aussi important de considérer d'autres cas d'usage que le *computer vision* par exemple avec des tâches de traitement automatique de la langue (NLP). Cependant, il est important de souligner que nos études sont principalement limitées par la complexité des cibles matérielles (par exemple de l'organisation de la mémoire Flash) plutôt que par la complexité des modèles en eux-mêmes (nombre de paramètres, nombre de couches). Aussi, les travaux futurs s'attacheront surtout à porter nos attaques physiques sur des cibles récentes, comme les nouvelles générations de plateformes microcontrôleurs avec accélérateurs dédiés à l'IA (e.g., Micro Coral, STM32N6/Cortex-M55+NPU).

Propositions et évaluations de protections. La communauté de la sécurité matérielle a une très longue expérience dans le développement de schémas de protections contre les analyses *side-channel* ou par injection de fautes. Ces contre-mesures sont généralement développées et évaluées pour des applications cryptographiques ou d'authentification. Un travail important doit alors être amorcé pour évaluer la pertinence de ces approches pour la protection de la confidentialité des modèles dans des systèmes d'IA embarquée, qui peuvent avoir des contraintes très fortes (mémoire, latence, énergie...). De plus, le développement et l'évaluation de protections efficaces doit prendre en compte leur complémentarité avec des approches algorithmiques qui visent à renforcer la robustesse intrinsèque des modèles, par exemple avec de la *Differential Privacy* ou des choix d'architecture (e.g., variantes de ReLU). Nos premières analyses montrent que les approches par randomisation (autant pour le *side-channel* que l'injection de fautes) sont très prometteuses car elles permettraient, d'une part, de réduire sensiblement l'impact de la protection sur le temps d'inférence et, d'autre part, de trouver un compromis entre le niveau de la protection et l'impact sur la performance nominale du modèle.

Chapitre 6

Conclusion

Contents

6.1 Contributions

Dans le cadre de l'obtention d'une VAE pour le diplôme de doctorat, ce manuscrit regroupe mes principales expériences scientifiques, pendant la période 2016-2024, au cours de laquelle j'ai coordonné et participé à de nombreuses actions sur la thématique de la *sécurité de l'IA embarquée*. Par souci de cohérence, un choix a été effectué parmi toutes mes expériences pour faire ressortir les contributions qui m'apparaissent comme les plus représentatives par rapport à l'état de l'art. Plus particulièrement, le manuscrit s'intéresse presque exclusivement à la caractérisation et à l'évaluation de menaces plutôt que de protections. Ce choix est justifié par le contexte actuel de la sécurité de l'IA dans l'Union Européenne. La mise en place progressive de l'*IA Act* a significativement bouleversé des communautés scientifiques et techniques qui mettent en œuvre des systèmes critiques basés sur l'IA (e.g., aéronautique, santé, défense, infrastructures énergétiques) et qui devront répondre à des exigences de sûreté et de sécurité. Au moment de la rédaction de ce manuscrit, les agences gouvernementales de sécurité sont confrontées à de multiples urgences pour identifier les menaces, définir des modèles de menaces pertinents en fonction des cas d'usage et – surtout – définir des protocoles robustes d'évaluation. Les contributions présentées dans ce manuscrit essaient d'apporter des éléments d'analyse de risque ainsi que des méthodes d'évaluation pour des menaces qui sont (pour le moment) trop peu prises en compte dans la communauté scientifique, même si l'importance de la sécurité de l'IA dans les communautés de la sécurité logicielle et matérielle prend un réel essor depuis plusieurs années.

La sécurité de l'IA s'est surtout développée avec l'émergence du *Deep Learning* et la démonstration d'une multitude de failles visant l'intégrité, la disponibilité et la confidentialité des modèles, que cela soit à l'apprentissage ou à l'inférence. Nous avons présenté dans le chapitre 3 un panorama, non-exhaustif, de ces menaces et avons surtout insisté sur l'importance de considérer une surface d'attaque qui tient compte autant des failles algorithmiques que celles liées aux implémentations logicielles et matérielles des modèles. Ces dernières sont extrêmement nombreuses, car un programme d'IA embarquée est rarement développé pour répondre à des critères de sécurité. Nous avons ainsi illustré ce défi de sécurité dans le chapitre 3 (3.6.2), avec un simple programme d'inférence d'un modèle standard de DL (CNN), sur un microcontrôleur 32-bit, entraîné par une plateforme classique (e.g., Tensorflow, PyTorch) et déployé par une librairie standard (e.g., CMISI-NN, CubeMX.AI, NNoM). Nous avons montré qu'un seul saut d'instruction, effectué par une injection laser ou une impulsion électromagnétique, peut significativement altérer l'inférence d'un modèle. La surface d'attaque de l'IA embarquée est d'autant plus complexe que toute une partie de l'IA moderne se concentre sur l'optimisation de modèles, certains avec des architectures très complexes, pour répondre aux exigences et contraintes de ces plateformes. Or, ces techniques d'optimisation ne sont pas sans conséquence sur la sécurité des modèles. Ainsi, nous avons pu voir dans le chapitre 3 (3.4.2.4) qu'une technique d'optimisation aussi standard et fondamentale que la quantification peut avoir une influence importante sur l'évaluation de la robustesse d'un modèle embarqué vis-à-vis d'attaques par évasion (*adversarial examples*).

En termes de menaces propres à l'IA embarquée, les principales contributions présentées dans ce manuscrit concernent les menaces contre les paramètres stockés en mémoire, que nous analysons dans le chapitre 4. Ces attaques sont de plus en plus étudiées avec des récentes études pendant la phase d'apprentissage [161, 162]. La principale référence est la Bit-Flip Attack (BFA) qui cherche à détériorer la performance moyenne d'un modèle en modifiant un nombre restreint de bits des paramètres. Dans le chapitre 4 (4.1), nous montrons que la définition même de cette attaque soulève plusieurs interrogations (plus particulièrement sur le modèle de menace initial) et que son évaluation peut s'avérer complexe en fonction des particularités du modèle étudié. La BFA et toutes les autres variantes (non-ciblées et ciblées) ayant été uniquement démontrées par simulations ou sur des cibles matérielles disposant de mémoires DRAM (Rowhammer), nous avons démontré qu'il était possible d'évaluer la robustesse d'un modèle déployé sur un microcontrôleur (Cortex-M) disposant de mémoire Flash contre des perturbations sur ses paramètres, induites par un tir laser produisant des fautes *bit-set*. Ces études ouvrent donc la voie à des protocoles d'évaluation mêlant simulations et caractérisations réelles sous banc de test. Ces deux études sont des premières dans l'état de l'art [147, 148].

Les dernières contributions s'intéressent à la menace d'*extraction de modèles*. Ce type d'attaques rassemble un nombre croissant de travaux et une variété remarquable d'approches. En considérant un modèle embarqué comme un secret (potentiellement très critique) à part entière, il est logique de s'intéresser aux attaques physiques comme les analyses *side-channel* ou par injection de fautes qui ont démontré des performances impressionnantes pour l'extraction de secrets

cryptographiques. Nous avons considéré le scénario d'extraction de modèle le plus exigeant ("fidélité"), à savoir la création d'un modèle substitut qui se rapprocherait le plus possible d'un clone du modèle cible. Cela implique l'extraction, d'une part, de l'architecture du DNN et, d'autre part, de la valeur de ses paramètres intrinsèques. Nous avons démontré que l'analyse *side-channel* est un outil très puissant pour retrouver un maximum d'informations relatives à l'architecture d'un modèle. A partir d'une cible matérielle de l'état de l'art (Cortex-M7, STM32H7) et un minimum de traces EM, notre méthodologie permet de retrouver l'organisation des couches d'un MLP ou d'un CNN ainsi que la grande majorité de leurs hyperparamètres.

Dans la littérature, l'extraction de modèle est principalement étudiée à travers les paramètres. Or, nous avons montré que la principale référence pour l'extraction des paramètres par analyse *side-channel* sur microcontrôleurs [105] est sujette à de nombreuses questions ouvertes autant théoriques que pratiques et méthodologiques. Aussi, nous avons proposé une étude complète (de l'opération élémentaire de multiplication jusqu'à un modèle) permettant de définir le degré de précision qu'il était possible d'obtenir par une analyse de type CPA ainsi qu'un ensemble de défis et problèmes qu'il était nécessaire de résoudre pour pouvoir extraire les valeurs de paramètres de modèles de l'état de l'art.

Enfin, tirant profit d'une technique novatrice d'injection de fautes pour l'extraction de modèle sur plateforme DRAM (Rowhammer [90] nous avons proposé d'utiliser le principe de la *Safe-Error Attack* dans un scénario d'extraction partielle de paramètres. Cette méthode s'accorde parfaitement au modèle de fautes data-dépendant de type *bit-set* que l'on observe avec de l'injection laser dans des mémoires Flash. La performance de la SEA est fortement influencée par le choix des entrées et nous avons proposé une méthode simple permettant de générer des données d'entrée qui améliorent significativement le taux d'extraction des bits les plus significatifs des paramètres. Ensuite, ces informations permettent d'apprendre un modèle substitut avec un nombre très limité de données d'apprentissage.

6.2 Perspectives

On-device training. Toutes les contributions présentées dans ce manuscrit concernent des menaces à l'inférence. Cependant, le développement de nouvelles plateformes matérielles dédiées à l'IA embarquée rend aujourd'hui possible l'apprentissage embarqué pour de multiples cas d'usage. En parallèle, des nouveaux paradigmes d'apprentissage distribués sont proposés, comme le *Federated Learning* ou le *Gossip Learning*. Pour ces systèmes permettant l'apprentissage et l'inférence, la surface d'attaque est encore plus problématique, plus particulièrement avec les menaces par empoisonnement. Ainsi, de nouveaux vecteurs d'attaque ont été récemment démontrés [161–163], pendant l'apprentissage, exploitant des fautes directement appliquées sur les paramètres, selon des principes proches de la BFA et ses variantes. L'utilisation d'analyses *side-channel* à l'apprentissage est aussi une perspective intéressante, car le processus d'apprentissage (*forward*/*backward pass*) peut *fuiter* des informations sensibles sur la constitution du modèle victime.

Analyses side-channel avancées. Dans le contexte de l'extraction de modèle, nos études ont montré la pertinence, mais aussi les limites d'analyses *side-channel* standards comme les *Timing Analysis*, *Simple Power Analysis* ou *Correlation Power Analysis*. Une perspective intéressante serait de traiter des modèles quantifiés sur 8 bits, déployés avec une librairie comme CMSIS-NN sur des plateformes disposant d'instructions SIMD. Dans ces cas, l'exploitation des fuites est certainement plus difficile mais la quantification sur 8 bits permettrait de réduire la complexité de l'attaque (recherche exhaustive des hypothèses). Une piste de recherche serait aussi d'utiliser des attaques profilées (par exemple par *Deep Learning*) qui ont démontré d'excellentes performances même en présence d'une désynchronisation des traces [164]. Enfin, les modèles CNN (très peu étudiés pour l'extraction des paramètres) possèdent des paramètres qui sont partagés ce qui impliquerait la présence de nombreuses fuites associées à un seul paramètre et qu'un attaquant pourrait exploiter.

Security-by-design. La communauté de sécurité de l'IA a proposé de nombreuses protections qui visent soit à renforcer la robustesse des modèles (approches proactives), soit à détecter des attaques (approches réactives). Une perspective importante est de considérer les approches de type *security-by-design* au niveau algorithmique (e.g., *adversarial training*, *differential privacy*) *et* aux niveaux logiciel et matériel (e.g., intégrité des données, protections du flot d'instructions). Les contributions présentées apportent des connaissances indispensables à la compréhension de

certaines vulnérabilités algorithmiques et physiques, qui sont propres aux modèles embarqués, et doivent permettre de développer des protections appropriées et de mieux évaluer leurs apports réels. Cet axe de recherche est au coeur du projet ANR AI.MMUNITY sur le *security-by-design* pour les systèmes de *Federated Learning*, plus particulièrement sur des architectures RISC-V.

Vers le MLSecOps. Malgré les nombreux progrès réalisés sur la caractérisation des menaces et les protections associées, la communauté de la sécurité de l'IA s'accorde sur le fait qu'il est devenu urgent de considérer les systèmes d'IA dans leur globalité (et complexité) et non plus comme un seul et unique modèle. Un système d'IA réel, opérationnel, est toujours constitué de plusieurs processus et opérations qui sont (le plus souvent) fortement dynamiques (e.g., mise à jour des données, *fine-tuning* des modèles). De plus, les modalités d'utilisation des modèles d'IA évoluent fortement : nous sommes déjà dans l'ère des modèles pré-entraînés, généralement des grands modèles de fondation multi-tâches, disponibles sur des plateformes de dépôts (e.g., Huggingface, STM32AI Model-Zoo) que chacun peut *adapter* selon son usage. Afin de formaliser la diversité et la complexité des systèmes d'IA, plusieurs *frameworks* de *Machine Learning Operations* (MLOps) voient le jour ainsi que des premières tentatives de MLSecOps (*Machine Learning Security Operations*), qui essaient de cibler l'ensemble des opérations de sécurité des systèmes d'IA. Il est important que les menaces physiques et les bonnes pratiques de protections viennent alimenter ces actions pour couvrir au mieux la surface d'attaque de ces systèmes aussi complexes que critiques.

Bibliographie

[1] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 99–108, 2004.

[2] D. Lowd and C. Meek, "Adversarial learning," in Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 641–647, 2005.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in International Conference on Learning Representations, vol. 103, 2014.

[4] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III, (Berlin, Heidelberg), pp. 387–402, Springer, Springer-Verlag, 2013.

[5] J. Rando, J. Zhang, N. Carlini, and F. Tramèr, "Adversarial ml problems are getting harder to solve and to evaluate," arXiv preprint arXiv :2502.02260, 2025.

[6] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack : Crushing neural network with progressive bit search," in Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1211–1220, 2019.

[7] T. M. Mitchell, Machine learning, vol. 1. McGraw-hill New York, 1997.

[8] J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum, "A cookbook of self-supervised learning," arXiv preprint arXiv :2304.12210, 2023.

[9] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning, vol. 1. MIT press Cambridge, 2016.

[10] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," IEEE signal processing magazine, vol. 29, no. 6, 2012.

[11] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist : a novel image dataset for benchmarking machine learning algorithms," arXiv preprint arXiv :1708.07747, 2017.

[12] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al., "Reading digits in natural images with unsupervised feature learning," in NIPS workshop on deep learning and unsupervised feature learning, vol. 2011, p. 4, Granada, 2011.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet : A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255, Ieee, 2009.

[15] R. Bernhard, P.-A. Moëllic, and J.-M. Dutertre, "Impact of low-bitwidth quantization on the adversarial robustness for embedded neural networks," in 2019 International Conference on Cyberworlds (CW), pp. 308–315, IEEE, 2019.

[16] C. Gaine, P.-A. Moëllic, O. Potin, and J.-M. Dutertre, "Fault injection on embedded neural networks : Impact of a single instruction skip," in 2023 26th Euromicro Conference on Digital System Design (DSD), pp. 317–324, IEEE, 2023.

[17] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in IEEE European Symposium on Security and Privacy, pp. 372–387, IEEE, 2016.

[18] H. Machiraju, O.-H. Choung, P. Frossard, M. Herzog, et al., "Bio-inspired robustness : A review," arXiv preprint arXiv :2103.09265, 2021.

[19] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer : Evaluating and testing unintended memorization in neural networks," in 28th USENIX Security Symposium (USENIX Security 19), pp. 267–284, 2019.

[20] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis.," in USENIX security symposium, vol. 16, pp. 601–618, 2016.

[21] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," Advances in Neural Information Processing Systems, vol. 33, pp. 1633–1645, 2020.

[22] A. C. Serban and E. Poll, "Adversarial examples-a complete characterisation of the phenomenon," arXiv preprint arXiv :1810.01185, 2018.

[23] Y. Bengio, S. Mindermann, D. Privitera, T. Besiroglu, R. Bommasani, S. Casper, Y. Choi, P. Fox, B. Garfinkel, D. Goldfarb, et al., "International ai safety report," arXiv preprint arXiv :2501.17805, 2025.

[24] B. Biggio and F. Roli, "Wild patterns : Ten years after the rise of adversarial machine learning," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2154–2156, 2018.

[25] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets : Identifying vulnerabilities in the machine learning model supply chain," arXiv preprint arXiv :1708.06733, 2017.

[26] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 27–38, ACM, 2017.

[27] N. Kandpal, M. Jagielski, F. Tramèr, and N. Carlini, "Backdoor attacks for in-context learning with language models," in The Second Workshop on New Frontiers in Adversarial Machine Learning.

[28] Y. Zhang, J. Rando, I. Evtimov, J. Chi, E. M. Smith, N. Carlini, F. Tramèr, and D. Ippolito, "Persistent pre-training poisoning of llms," arXiv preprint arXiv :2410.13722, 2024.

[29] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli, "Wild patterns reloaded : A survey of machine learning security against training data poisoning," ACM Computing Surveys, vol. 55, no. 13s, pp. 1–39, 2023.

[30] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in Asian conference on machine learning, pp. 97–112, PMLR, 2011.

[31] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," 2018.

[32] N. Carlini, M. Jagielski, C. A. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr, "Poisoning web-scale training datasets is practical," in 2024 IEEE Symposium on Security and Privacy (SP), pp. 407–425, IEEE, 2024.

[33] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical Black-Box Attacks against Machine Learning," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, (New York, NY, USA), pp. 506–519, ACM, Association for Computing Machinery, apr 2017.

[34] J. Chen, M. I. Jordan, and M. J. Wainwright, "Hopskipjumpattack : A query-efficient decision-based attack," in 2020 IEEE Symposium on Security and Privacy (SP), pp. 1277–1294, IEEE, 2020.

[35] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks : Reliable attacks against black-box machine learning models," in International Conference on Learning Representations, 2018.

[36] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in International conference on machine learning, pp. 2206–2216, PMLR, 2020.

[37] C. Dwork, "Differential privacy : A cryptographic approach to private data analysis," Privacy, Big Data, and the Public Good, p. 296, 2014.

[38] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 202–210, 2003.

[39] S. V. Dibbo, "Sok : Model inversion attack landscape : Taxonomy, challenges, and future roadmap," in 2023 IEEE 36th Computer Security Foundations Symposium (CSF), pp. 439–456, IEEE, 2023.

[40] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in Security and Privacy (SP), 2017 IEEE Symposium on, pp. 3–18, IEEE, 2017.

[41] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," in 2022 IEEE Symposium on Security and Privacy (SP), pp. 1897–1914, IEEE Computer Society, 2022.

[42] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in 29th USENIX security symposium (USENIX Security 20), pp. 1345–1362, 2020.

[43] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in Annual international cryptology conference, pp. 189–218, Springer, 2020.

[44] I. A. Canales-Martínez, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, N. Satpute, and A. Shamir, "Polynomial time cryptanalytic extraction of neural network models," in Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 3–33, Springer, 2024.

[45] N. Carlini, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, and A. Shamir, "Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting," arXiv preprint arXiv :2410.05750, 2024.

[46] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, "Sponge examples : Energy-latency attacks on neural networks. http s," arxiv. org/abs/2006.03463, 2020.

[47] A. E. Cinà, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "Energy-latency attacks via sponge poisoning," Information Sciences, vol. 702, p. 121905, 2025.

[48] K. Grosse, T. A. Trost, M. Mosbach, M. Backes, and D. Klakow, "On the security relevance of initial weights in deep neural networks," in International Conference on Artificial Neural Networks, pp. 3–14, 2020.

[49] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson, "Manipulating sgd with data ordering attacks," Advances in Neural Information Processing Systems, vol. 34, pp. 18021–18032, 2021.

[50] A. G. Howard, "Mobilenets : Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv :1704.04861, 2017.

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2 : Inverted residuals and linear bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[52] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al., "Fp8 formats for deep learning," arXiv preprint arXiv :2209.05433, 2022.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[54] M. Tan and Q. Le, "Efficientnet : Rethinking model scaling for convolutional neural networks," in International conference on machine learning, pp. 6105–6114, PMLR, 2019.

[55] B. Nguyen, P.-A. Moëllic, and S. Blayac, "Evaluation of convolution primitives for embedded neural networks on 32-bit microcontrollers," in International Conference on Intelligent Systems Design and Applications, pp. 427–437, Springer, 2022.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, pp. 1097–1105, 2012.

[57] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep roots : Improving cnn efficiency with hierarchical filter groups," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1231–1240, 2017.

[58] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9, 2015.

[59] Y. Jeon and J. Kim, "Constructing fast network through deconstruction of convolution," Advances in neural information processing systems, vol. 31, 2018.

[60] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Addernet : Do we really need multiplications in deep learning?," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1468–1477, 2020.

[61] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8697–8710, 2018.

[62] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, et al., "Mcunet : Tiny deep learning on iot devices," Advances in Neural Information Processing Systems, vol. 33, pp. 11711–11722, 2020.

[63] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Mcunetv2 : Memory-efficient patch-based inference for tiny deep learning," in Annual Conference on Neural Information Processing Systems (NeurIPS), 2021.

[64] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," 2022.

[65] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv :1503.02531, 2015.

[66] A. Mishra and D. Marr, "Apprentice : Using knowledge distillation techniques to improve low-precision network accuracy," arXiv preprint arXiv :1711.05852, 2017.

[67] S. Han, H. Mao, and W. J. Dally, "Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv :1510.00149, 2015.

[68] N. Lee, T. Ajanthan, and P. Torr, "Snip : single-shot network pruning based on connection sensitivity," in International Conference on Learning Representations, Open Review, 2019.

[69] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in International Conference on Learning Representations, 2017.

[70] B. Nguyen, P.-A. Moëllic, and S. Blayac, "Domain generalization on constrained platforms : On the compatibility with pruning techniques," in Global IoT Summit, pp. 250–261, Springer, 2022.

[71] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect : Training deep neural networks with binary weights during propagations," in Advances in neural information processing systems, vol. 28, pp. 3123–3131, 2015.

[72] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks : Training deep neural networks with weights and activations constrained to+ 1 or-1," arXiv preprint arXiv :1602.02830, 2016.

[73] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net : Training low bitwidth convolutional neural networks with low bitwidth gradients," arXiv preprint arXiv :1606.06160, 2016.

[74] H. V. Habi, R. H. Jennings, and A. Netzer, "Hmq : Hardware friendly mixed precision quantization block for cnns," in Computer Vision–ECCV 2020 : 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16, pp. 448–463, Springer, 2020.

[75] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14095–14103, June 2020.

[76] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn : Efficient neural network kernels for arm cortex-m cpus," arXiv preprint arXiv :1801.06601, 2018.

[77] J. Ma, "A higher-level Neural Network library on Microcontrollers (NNoM)." `https://github.com/majianjia/nnom`, oct 2020.

[78] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al., "Deepseek-v3 technical report," arXiv preprint arXiv :2412.19437, 2024.

[79] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," in International Conference on Learning Representations, 2018.

[80] E. B. Khalil, A. Gupta, and B. Dilkina, "Combinatorial attacks on binarized neural networks," arXiv preprint arXiv :1810.03538, 2018.

[81] S. Mangard, E. Oswald, and T. Popp, Power analysis attacks : Revealing the secrets of smart cards, vol. 31. Springer Science & Business Media, 2007.

[82] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices : Theory, practice, and countermeasures," Proceedings of the IEEE, vol. 100, no. 11, pp. 3056–3076, 2012.

[83] J. Breier and X. Hou, "How practical are fault injection attacks, really?," IEEE Access, vol. 10, pp. 113122–113130, 2022.

[84] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them : An experimental study of dram disturbance errors," ACM SIGARCH Computer Architecture News, vol. 42, pp. 361–372, jun 2014.

[85] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "Voltjockey : Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, (New York, NY, USA), pp. 195–209, Association for Computing Machinery, 2019.

[86] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?," in 2010 IEEE 16th International On-Line Testing Symposium, pp. 235–239, IEEE, 2010.

[87] J.-M. Dutertre, V. Beroulle, P. Candelier, S. De Castro, L.-B. Faber, M.-L. Flottes, P. Gendrier, D. Hély, R. Leveugle, P. Maistri, G. Di Natale, A. Papadimitriou, and B. Rouzeyre, "Laser fault injection at the cmos 28 nm technology node : an analysis of the fault model,," in 2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 14th Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 1–6, IEEE, Sep. 2018.

[88] B. Colombier, A. Menu, J. M. Dutertre, P. A. Moëllic, J. B. Rigaud, and J. L. Danger, "Laser-induced Single-bit Faults in Flash Memory : Instructions Corruption on a 32-bit Microcontroller," IEEE International Symposium on Hardware Oriented Security and Trust, HOST, pp. 1–10, 2019.

[89] T. Nayan, Q. Guo, M. Al Duniawi, M. Botacin, S. Uluagac, and R. Sun, "Sok : All you need to know about on-device ml model extraction-the gap between research and practice," in 33rd USENIX Security Symposium (USENIX Security 24), pp. 5233–5250, 2024.

[90] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "Deepsteal : Advanced model extractions leveraging efficient weight stealing in memories," in IEEE Symposium on Security and Privacy (SP), pp. 1157–1174, IEEE, arXiv, 2021.

[91] H. Chabanne, J.-L. Danger, L. Guiga, and U. Kühne, "Side channel attacks for architecture extraction of neural networks," CAAI Transactions on Intelligence Technology, vol. 6, no. 1, 2021.

[92] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," arXiv preprint arXiv :1812.11720, 2018.

[93] Y. Xiang, Z. Chen, et al., "Open DNN Box by Power Side-Channel Attack," IEEE Transactions on Circuits and Systems II : Express Briefs, vol. 67, no. 11, pp. 2717–2721, 2020.

[94] Y. Liu and A. Srivastava, "Ganred : Gan-based reverse engineering of dnns via cache side-channel," in Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2020.

[95] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitraş, "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," arXiv preprint arXiv :1810.03487, 2018.

[96] S. Hong, M. Davinroy, Y. Kaya, D. Dachman-Soled, and T. Dumitraş, "How to 0wn nas in your spare time," arXiv preprint arXiv :2002.06776, 2020.

[97] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, "Can one hear the shape of a neural network ? : Snooping the gpu via magnetic side channel," arXiv preprint arXiv :2109.07395, 2021.

[98] Z. Wang, X. Zeng, X. Tang, D. Zhang, X. Hu, and Y. Hu, "Demystifying arch-hints for model extraction : An attack in unified memory system," arXiv preprint arXiv :2208.13720, 2022.

[99] M. Yan, C. Fletcher, and J. Torrellas, "Cache telepathy : Leveraging shared resource attacks to learn dnn architectures," in USENIX Security Symposium, 2020.

[100] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, et al., "Deepsniffer : A dnn model extraction framework based on learning architectural hints," in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020.

[101] M. M. Ahmadi, L. Alrahis, A. Colucci, O. Sinanoglu, and M. Shafique, "Neurounlock : Unlocking the architecture of obfuscated deep neural networks," in 2022 International Joint Conference on Neural Networks (IJCNN), IEEE, 2022.

[102] W. Hua, Z. Zhang, and G. E. Suh, "Reverse-engineering cnn models using side-channel attacks," IEEE Design & Test, vol. 39, no. 4, 2022.

[103] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack : Steal dnn models with lossless inference accuracy.," in USENIX Security Symposium, 2021.

[104] X. Hu, L. Liang, L. Deng, S. Li, X. Xie, Y. Ji, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Neural network model extraction attacks in edge devices by hearing architectural hints," arXiv preprint arXiv :1903.03916, 2019.

[105] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN : Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel," in 28th USENIX Security Symposium, pp. 515–532, USENIX Association, [Sl] : USENIX, 2019.

[106] R. Joud, P.-A. Moëllic, S. Pontié, and J.-B. Rigaud, "Like an open book ? read neural network architecture with simple power analysis on 32-bit microcontrollers," in International Conference on Smart Card Research and Advanced Applications, pp. 256–276, Springer, 2023.

[107] Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks," in 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 657–662, IEEE, 2019.

[108] J. Li, Z. He, A. S. Rakin, D. Fan, and C. Chakrabarti, "Neurobfuscator : A full-stack obfuscation tool to mitigate neural architecture stealing," in 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 248–258, IEEE, 2021.

[109] T. Zhou, S. Ren, and X. Xu, "Obfunas : A neural architecture search-based dnn obfuscation approach," in Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, pp. 1–9, 2022.

[110] H. Chabanne, J.-L. Danger, L. Guiga, and U. Kühne, "Telepathic headache : Mitigating cache side-channel attacks on convolutional neural networks," in International Conference on Applied Cryptography and Network Security, pp. 363–392, Springer, 2021.

[111] Y. Luo, S. Duan, C. Gongye, Y. Fei, and X. Xu, "Nnrearch : A tensor program scheduling framework against neural network architecture reverse engineering," in IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2022.

[112] C. Gongye, Y. Fei, and T. Wahl, "Reverse-Engineering Deep Neural Networks Using Floating-Point Timing Side-Channels," in IEEE Design Automation Conference (DAC), pp. 1–6, IEEE, jul 2020. ISSN : 0738-100X.

[113] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky nets : Recovering embedded neural network models and inputs through simple power and timing side-channels–attacks and defenses," IEEE Internet of Things Journal, vol. 8, no. 15, pp. 12079–12092, 2021.

[114] D. Stutz, N. Chandramoorthy, M. Hein, and B. Schiele, "Random and adversarial bit error robustness : Energy-efficient and secure dnn accelerators," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.

[115] F. Yao, A. S. Rakin, and D. Fan, "{DeepHammer} : Depleting the intelligence of deep neural networks through targeted chain of bit flips," in 29th USENIX Security Symposium (USENIX Security 20), pp. 1463–1480, 2020.

[116] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning ?," Proceedings of machine learning and systems, vol. 2, pp. 129–146, 2020.

[117] B. Ghavami, M. Sadati, M. Shahidzadeh, Z. Fang, and L. Shannon, "Blind data adversarial bit-flip attack against deep neural networks," in 2022 25th Euromicro Conference on Digital System Design (DSD), pp. 899–904, 2022.

[118] D. Park, K.-W. Kwon, S. Im, and J. Kung, "Zebra : Precisely destroying neural networks with zero-data based repeated bit flip attack," arXiv preprint arXiv :2111.01080, 2021.

[119] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in 2018 IEEE Symposium on Security and Privacy (SP), pp. 245–261, IEEE, 2018.

[120] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA} : Exploiting {DRAM} addressing for {Cross-CPU} attacks," in 25th USENIX security symposium (USENIX security 16), pp. 565–581, 2016.

[121] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "Rowpress : Amplifying read disturbance in modern dram chips," in Proceedings of the 50th Annual International Symposium on Computer Architecture, pp. 1–18, 2023.

[122] R. Zhou, J. T. Liu, S. Ahmed, S. Angizi, and A. S. Rakin, "Compromising the intelligence of modern dnns : On the effectiveness of targeted rowpress," arXiv preprint arXiv :2412.02156, 2024.

[123] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-bfa : Targeted bit-flip adversarial weight attack," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2021.

[124] Y. Khare, K. Lakara, M. S. Inukonda, S. Mittal, M. Chandra, and A. Kaushik, "Design and analysis of novel bit-flip attacks and defense strategies for dnns," in 2022 IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8, 2022.

[125] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank : Filter pruning using high-rank feature map," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1529–1538, 2020.

[126] P. Zhao, S. Wang, C. Gongye, Y. Wang, Y. Fei, and X. Lin, "Fault sneaking attack : A stealthy framework for misleading deep neural networks," in 56th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, 2019.

[127] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 131–138, IEEE, 2017.

[128] B. Ghavami, S. Movi, Z. Fang, and L. Shannon, "Stealthy attack on algorithmic-protected dnns via smart bit flipping," in 2022 23rd International Symposium on Quality Electronic Design (ISQED), pp. 1–7, IEEE, 2022.

[129] Y. Guo, L. Liu, Y. Cheng, Y. Zhang, and J. Yang, "Modelshield : A generic and portable framework extension for defending bit-flip based adversarial weight attacks," in 2021 IEEE 39th International Conference on Computer Design (ICCD), pp. 559–562, 2021.

[130] M. Javaheripi and F. Koushanfar, "Hashtag : Hash signatures for online detection of fault-injection attacks on deep neural networks," in IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9, IEEE, 2021.

[131] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, "Radar : Run-time adversarial weight attack detection and accuracy recovery," in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 790–795, IEEE, 2021.

[132] F. S. Hosseini, Q. Liu, F. Meng, C. Yang, and W. Wen, "Safeguarding the intelligence of neural networks with built-in light-weight integrity marks (lima)," in 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1–12, IEEE, 2021.

[133] Q. Liu, J. Yin, W. Wen, C. Yang, and S. Sha, "{NeuroPots} : realtime proactive defense against {Bit-Flip} attacks in neural networks," in 32nd USENIX Security Symposium (USENIX Security 23), pp. 6347–6364, 2023.

[134] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, 2020.

[135] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in International Conference on Learning Representations, 2018.

[136] R. Zhou, S. Ahmed, A. S. Rakin, and S. Angizi, "Dnn-defender : An in-dram deep neural network defense mechanism for adversarial weight attack," arXiv preprint arXiv :2305.08034, 2023.

[137] R. Zhou, S. Ahmed, A. Roohi, A. S. Rakin, and S. Angizi, "Dram-locker : A general-purpose dram protection mechanism against adversarial dnn weight attacks," arXiv preprint arXiv :2312.09027, 2023.

[138] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Practical fault attack on deep neural networks," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2204–2206, 2018.

[139] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Experimental evaluation of deep neural network resistance against fault injection attacks.," IACR Cryptol. ePrint Arch., p. 461, 2019.

[140] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Security Evaluation of Deep Neural Network Resistance against Laser Fault Injection," Proceedings of the International Symposium on the Physical and Failure Analysis of Integrated Circuits, IPFA, pp. 1–6, 2020.

[141] Y. Fukuda, K. Yoshida, and T. Fujino, "Fault injection attacks utilizing waveform pattern matching against neural networks processing on microcontroller," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. 105, no. 3, pp. 300–310, 2022.

[142] W. Liu, C.-H. Chang, F. Zhang, and X. Lou, "Imperceptible misclassification attack on deep learning accelerator by glitch injection," in 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, IEEE, 2020.

[143] C. Gaine, D. Aboulkassimi, S. Pontie, J.-P. Nikolovski, and J.-M. Dutertre, "Electromagnetic Fault Injection as a New Forensic Approach for SoCs," in 2020 IEEE International Workshop on Information Forensics and Security (WIFS), (New York, United States), pp. 1–6, IEEE, Dec 2020.

[144] J.-M. Dutertre, A. Menu, O. Potin, J.-B. Rigaud, and J.-L. Danger, "Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures," Microelectronics Reliability, vol. 121, p. 114133, 2021.

[145] A. Zgheib, O. Potin, J.-B. Rigaud, and J.-M. Dutertre, "A cfi verification system based on the risc-v instruction trace encoder," in 2022 25th Euromicro Conference on Digital System Design (DSD), pp. 456–463, IEEE, 2022.

[146] R. De Clercq, J. Götzfried, D. Übler, P. Maene, and I. Verbauwhede, "Sofia : software and control flow integrity architecture," Computers & Security, vol. 68, pp. 16–35, 2017.

[147] K. Hector, P.-A. Moëllic, M. Dumont, and J.-M. Dutertre, "A closer look at evaluating the bit-flip attack against deep neural networks," in IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp. 1–5, IEEE, 2022.

[148] M. Dumont, K. Hector, P.-A. Moëllic, J.-M. Dutertre, and S. Pontié, "Evaluation of parameter-based attacks against embedded neural networks with laser injection," in International Conference on Computer Safety, Reliability, and Security, pp. 259–272, Springer, 2023.

[149] N. Carlini, "Cutting through buggy adversarial example defenses : fixing 1 line of code breaks sabre," arXiv preprint arXiv :2405.03672, 2024.

[150] J. Zhang, K. Nikolić, N. Carlini, and F. Tramèr, "Gradient masking all-at-once : Ensemble everything everywhere is not robust," arXiv preprint arXiv :2411.14834, 2024.

[151] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in International Conference on Learning Representations, 2015.

[152] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[153] R. Joud, P.-A. Moëllic, S. Pontié, and J.-B. Rigaud, "A practical introduction to side-channel extraction of deep neural network parameters," in International Conference on Smart Card Research and Advanced Applications, pp. 45–65, Springer, 2022.

[154] K. Hector, P.-A. Moëllic, M. Dumont, and J.-M. Dutertre, "Fault injection and safe-error attack for extraction of embedded neural network models," in Workshop on Security and Artificial Intelligence, pp. 644–664, Springer, 2023.

[155] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed : Reading bits in memory without accessing them," in 2020 IEEE Symposium on Security and Privacy (SP), pp. 695–711, IEEE, 2020.

[156] A. Barbalau, A. Cosma, R. T. Ionescu, and M. Popescu, "Black-box ripper : Copying black-box models using generative evolutionary algorithms," Advances in Neural Information Processing Systems, vol. 33, pp. 20120–20129, 2020.

[157] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," in 29th USENIX Security Symposium (USENIX Security 20), pp. 1309–1326, 2020.

[158] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets : Stealing functionality of black-box models," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4954–4963, 2019.

[159] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," IEEE Transactions on computers, vol. 49, no. 9, pp. 967–970, 2000.

[160] L. Wulfert, J. Kühnel, L. Krupp, J. Viga, C. Wiede, P. Gembaczka, and A. Grabmaier, "Aifes : A next-generation edge ai framework," IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–16, 2024.

[161] K. Cai, M. H. I. Chowdhuryy, Z. Zhang, and F. Yao, "Deepvenom : Persistent dnn backdoors exploiting transient weight perturbations in memories," in 2024 IEEE Symposium on Security and Privacy (SP), pp. 2067–2085, IEEE, 2024.

[162] K. Cai, Z. Zhang, Q. Lou, and F. Yao, "Wbp : Training-time backdoor attacks through hardware-based weight bit poisoning," in European Conference on Computer Vision, pp. 179–197, Springer, 2024.

[163] J. Dong, H. Qiu, Y. Li, T. Zhang, Y. Li, Z. Lai, C. Zhang, and S.-T. Xia, "One-bit flip is all you need : When bit-flip attack meets model training," in Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 4688–4698, 2023.

[164] L. Masure, C. Dumas, and E. Prouff, "A comprehensive study of deep learning for side-channel analysis," IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 348–375, 2020 url.

Table des figures

Liste des tableaux